WebSphere® software

IBM

Best Practices for Implementing WebSphere Extended Deployment

Optimize for service levels, health management, and application hosting

Deploy long-running applications in WebSphere

Enhance session data sharing capabilities

Carla Sadtler
David Buchanan
Naveen Deshwal
Tomonori Iwashina
Edward McCarthy
Hendrik Van Run

Redbooks

ibm.com/redbooks

International Technical Support Organization

Best Practices for Implementing WebSphere Extended Deployment

February 2007

SG24-7343-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xi.

First Edition (February 2007)

This edition applies to WebSphere Extended Deployment V6.0.2.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

CICS®	IMS™	WebSphere®
DB2 Universal Database™	Rational®	z/OS®
DB2®	Redbooks™	
Extended Services®	Redbooks (logo) ™	
IBM®	Tivoli®	

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates.

EJB, Java, JDBC, JDK, JVM, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Excel, Internet Explorer, Microsoft, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This IBM® Redbook looks at scenarios for using WebSphere® Extended Deployment and outlines procedures and best practices for these scenarios. Scenarios for operations optimization, long-running application extenders, and data-intensive application extenders are included.

This book focuses on process, design, and usage guidelines, complimenting the “how to” information found in *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153. In addition, the business grid (batch and compute-intensive) capabilities are covered extensively including *how to* and *best practice* information.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Carla Sadtler is a certified IT Specialist at the ITSO, Raleigh Center. She writes extensively about the WebSphere and Patterns for e-business areas. Before joining the ITSO in 1985, Carla worked in the Raleigh branch office as a Program Support Representative. She holds a degree in mathematics from the University of North Carolina at Greensboro.

David Buchanan is a Consulting IT Specialist working as a WebSphere technical sales specialist in the finance sector in the United Kingdom. With over 30 years experience in the IT industry, he is an IBM Certified IT Specialist, a chartered member of the British Computer Society, and has worked for IBM for more than 13 years. He has a BSc and an MSc from Heriot-Watt University, Edinburgh.

Naveen Deshwal is a Sr. IT Specialist in IBM US. He has 12 years of experience in the IT field. He has worked at IBM for 7 years in various positions at the Web Hosting Delivery Center. He holds a graduate degree in eBusiness from the University of Phoenix and degree in Electronics and Communication from IIT in India. His area of expertise includes Web Hosting Infrastructure software and devices.

Tomonori Iwashina is an IT Specialist in IBM Japan Systems Engineering Co.,Ltd.(ISE), Japan. He has five years of experience with WebSphere

Application Server. He holds a Bachelor of Arts in Philosophy from the University of Tokyo.

Edward McCarthy currently works in the e-business Enablement Services team for IBM Global Services Australia. The team is responsible for covering all aspects of WebSphere Application Server and WebSphere MQ across all platforms. For the last five years, he has specialized in supporting the WebSphere range of products. He worked as a senior CICS® and WebSphere MQ systems programmer for over eight years with a large IBM client. He also participated in writing the IBM Redbooks™ *WebSphere Business Integration Server Foundation V5.1 for z/OS*, SG24-6382 and *WebSphere for z/OS V6 Connectivity Handbook*, SG24-7064.

Hendrik Van Run is a Senior IT Specialist within the pan-IOT Software Lab Services team in Hursley. He holds a Masters degree in Physics from Utrecht University in the Netherlands. Hendrik has extensive experience in designing, architecting, and reviewing corporate J2EE™ infrastructure, usually based on WebSphere technology. He advised many of the IBM enterprise customers on issues such as high availability, performance, and queuing problems across the WebSphere Application Server product families. Prior to joining the pan-IOT Software Lab Services team, Hendrik worked for IBM Global Services in the Netherlands. He has worked with WebSphere Application Server since Version 3.5 and is co-author of the redbook called *IBM WebSphere V5.0 Performance, Scalability, and High Availability*, SG24-6198-00.

Thanks to the following people for their contributions to this project:

Suzanne Dewitt
IBM US

Ann Black
IBM US

Snehal Antani
IBM US

Billy Newport
IBM US

Christopher Vignola
IBM US

Aditya A Desai
IBM US

Jared Anderson
IBM US

Brian K Martin
IBM US

Keith B Smith
IBM US

Sajan Sankaran
IBM US

Joshua Dettinger
IBM US

Gary Miller
IBM US

George Bonifant
IBM US

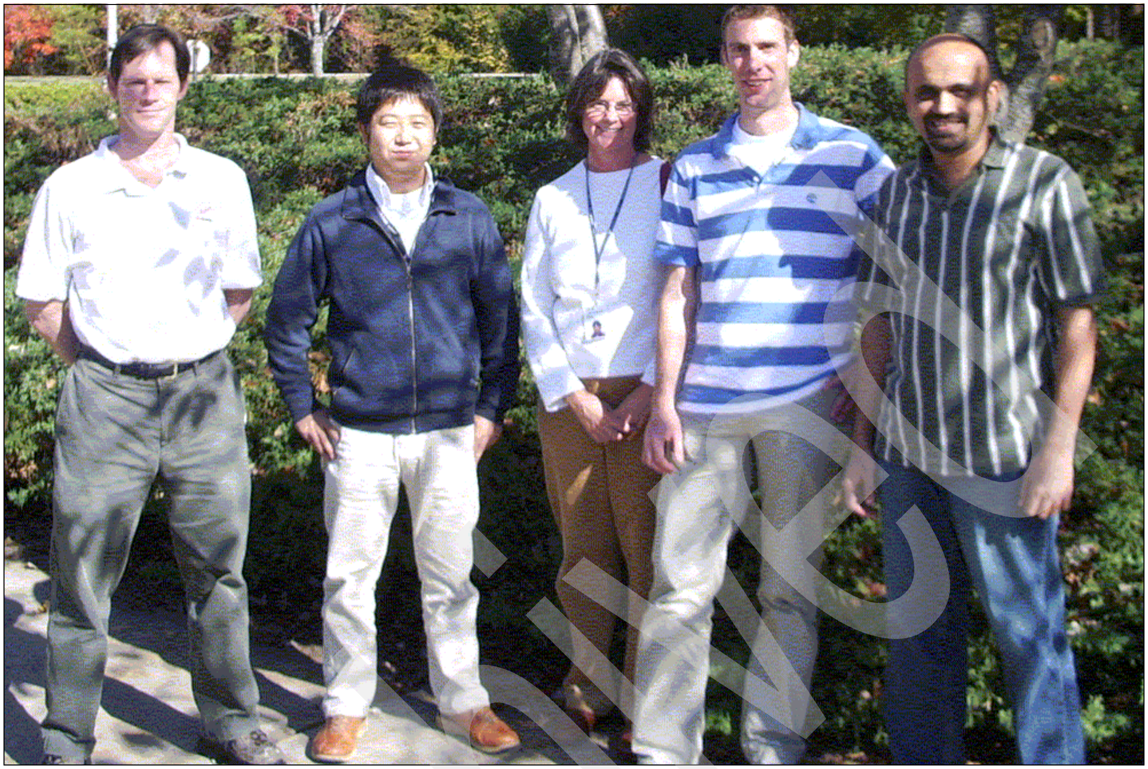


Figure 1 Authors (left to right): Edward McCarthy, Tomonori Iwashina, Carla Sadtler, Hendrik Van Run, Naveen Deshwal. Not pictured: David Buchanan

Become a published author

Join us for a two-to-six week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at the following Web site:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our IBM Redbooks to be as helpful as possible. Send us your comments about this or other redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at the following Web site:

  ibm.com/redbooks

- Send your comments in an e-mail to the following Web site:

  redbooks@us.ibm.com

- Mail your comments to the following address:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

Operations optimization

WebSphere Extended Deployment provides features that can help simplify management of complex environments while optimizing existing resources.

This part of the book contains the following chapters:

1

Service level optimization

This chapter focuses on service level optimization. It shows how the dynamic operations feature of WebSphere Extended Deployment can assist organizations in defining service level agreements and ensuring that their applications meet those agreements. Specifically, it will focus on the implementation of WebSphere Extended Deployment for a number of applications that all belong to a single-business area in an organization.

We describe the factors to consider and the methodology for implementing Extended Deployment in an already established WebSphere Network Deployment environment. We also provide an example of applying the methodology in an actual, albeit fictitious, WebSphere environment.

This chapter contains the following topics:

- What is service level optimization?
- Planning for service level optimization
- Implementation - hands on example

Chapter 2, “Application hosting and chargeback” on page 61 extends this example scenario to show the implementation of service level optimization across a WebSphere environment running applications from a number of business areas, which may belong to the same or different organizations.

1.1 What is service level optimization?

The term *service level agreement* (SLA) is commonly used in the industry to describe an agreement between a service provider and user detailing the level of service to be provided. This includes things like hardware configuration, services (libraries, and so on.), and response time goals for applications. When SLAs are in place, service level optimization becomes an important factor in meeting those agreements.

What do we mean by saying we want to have *service level optimization*? In brief we are talking about changing existing WebSphere environments from a static state to one where the WebSphere environment can dynamically manage itself to make the most efficient use of its resources.

Reasons to optimize for service levels:

If any of the following apply to your situation, you should consider service level optimization:

- You have service level objectives but meeting them requires manual, complex system management to be performed.
- You need a better way to guarantee certain application performance levels.
- You have to deliver consistent response times to external applications to ensure customer and business partner satisfaction.
- You want to deliver the best possible response times to internal users but without compromising the performance of critical externally facing transactions.
- Your applications must be highly available but you do not want to duplicate everything.
- You often need to deploy new applications and want help placing them on servers for optimum performance.

1.1.1 Static WebSphere environments

WebSphere Application Server is used in many organizations across the world. While no two WebSphere Application Server implementations are exactly the same, it is probably fair to say that most implementations fall into one of the following broad configurations:

- Each application at an organization runs in its own set of servers on its own hardware.

- A number of applications run in a server or cluster.
- Applications are split across servers.
- Some combination of the above.

Regardless of how many applications are mapped to WebSphere servers, it is probably fair to say that the environment is very static. There will probably be much more physical capacity in terms of CPU and memory available than is required to run the total workload. This tends to come about, especially in larger organizations, because each business area wants their own dedicated hardware and software environment for their application. Each business area will claim they require this to ensure their application performs well by having enough resources.

However it is also not uncommon for a business area to over-estimate how much their application is going to be used. The end result is that they double the expected load they think they will get to ensure they get more than enough capacity.

The more this happens, the more often the end result is a large server farm with more capacity than is actually required to run the total business workload. This results in the organization paying for much more expensive hardware and software then it needs.

The characteristic of these static environments is that they are not making the best use of the overall capacity and the configuration in terms of numbers of servers. Additionally these environments cannot quickly respond to unexpected changes in workload. For example if an application has a dramatic increase in load, there may be insufficient capacity in the servers set aside for that application to meet the demand. However, there may be sufficient capacity in other servers running other applications that cannot be used.

While it is possible to use the normal WebSphere Application Server product to manage a large WebSphere environment in a more dynamic manner, it would require the use of skilled WebSphere staff making decisions by looking at response time and CPU monitors and so on. Such an approach is costly and difficult to implement.

The z/OS exception

The above discussion applies specifically to the distributed platforms such as Windows® and Unix. Matters are different with WebSphere Application Server on z/OS®. The z/OS operating system has a workload management (WLM) feature that allows work to be classified and policies defined that specify the service levels different workloads are to meet.

WebSphere Application Server on z/OS has always taken advantage of WLM. Organizations that use WebSphere Application Server on z/OS typically run all the business load in some number of LPARs. The result is that much better utilization of total available CPU and memory is achieved. It is beyond the scope of this redbook to explain in full the details of WebSphere Application Server on z/OS and WLM; however, there are several redbooks available as well as the WebSphere Information Center.

There is still benefit to be gained from using WebSphere Extended Deployment on z/OS with regard to managing workloads. Configuring the on demand router (ODR) within the z/OS environment provides for more intelligent routing of requests across LPARs than can be done with the Sysplex Distributor. The Sysplex Distributor can load balance requests across LPARs hosting the same application but does not provide a way to maintain affinity. Maintaining affinity in this scenario would require the use of some product in front of the LPARs, such as the IBM HTTP Server and the Web server plug-in. The ODR, when run on z/OS, works with the standard z/OS WLM to determine where requests should be routed.

1.1.2 Dynamically managed environment

Using the dynamic operations features of WebSphere Extended Deployment you can change the way a typical WebSphere environment is configured today to one that has the following features:

- Improves the utilization of available resources such as CPU and memory
- Classifies and monitors the workload
- Provides a business-centric view of the workload and how it is performing
- Responds in real time to changes in the workload mix (without human intervention if so desired), using business guidelines that the organization specified

This is what we mean when we say that WebSphere Extended Deployment can be used to achieve service level optimization. The primary features that you will see used are shown in Figure 1-1 on page 7.

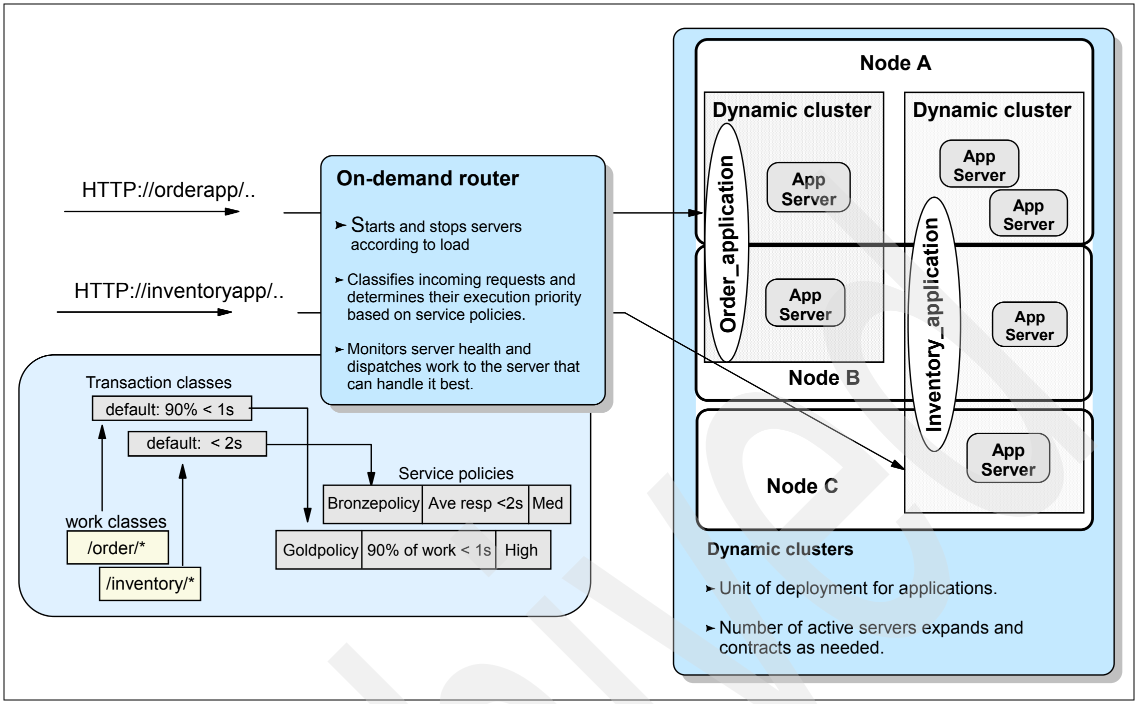


Figure 1-1 Dynamic operations

On demand router A component that sits in front of your application servers, responsible for managing the flow of requests into the WebSphere environment.

Service policies Defines performance goals and assigns relative importance for the applications being run. Service policies are a technical implementation of SLAs in place between the business area and the IT area running their applications.

Dynamic clusters Consist of a number of servers that WebSphere Extended Deployment can stop or start in response to changing workload.

1.2 Planning for service level optimization

When it comes to implementing WebSphere Extended Deployment, you will need to carefully plan and prepare its implementation.

1.2.1 Overview

Implementing WebSphere Extended Deployment includes several aspects:

- Gathering information to feed into the technical implementation
- The technical implementation
- Education for those who support the WebSphere environment

The technical implementation

At the technical level, you need to consider the following:

- How you are going to install the WebSphere Extended Deployment software into the existing WebSphere environment
- What your current topology is, and how the updated topology should look
- How you will implement the on demand router, for example, where you wall place it in the environment
- How to define components related to service level optimization such as transaction classes and service policies for your environment
- How you will move applications from a static environment to a dynamically managed one

Gathering information about the current environment

For a successful WebSphere Extended Deployment implementation, you need to gather as much detailed information as possible about the environment, such as the following:

- Application usage characteristics
- Application load characteristics
- Application interdependencies
- Service level goals that the business areas require for their applications

It is this information that feeds into the technical process in terms of defining workload policies that WebSphere Extended Deployment will use to manage the workload.

Education

You are going to need to educate different areas with respect to the implementation of WebSphere Extended Deployment such as the following:

- Operation staff who look after WebSphere
- Business areas

- Application development areas
- Test teams who test applications

1.2.2 Gathering information

Gathering information is probably the key step to a successful WebSphere Extended Deployment implementation.

Environment layout

A first step is to obtain a diagram of the existing WebSphere cell configuration. At a minimum, this diagram should show the following:

- The physical servers that are used
- The deployment manager and application servers' locations

Identify your applications

The next step is to identify the applications running in the WebSphere environment. As each organization is different in how they set up and manage their WebSphere environments, it is impossible to lay down exact rules for gathering application information. Following are suggested guidelines you could use:

Produce a report that shows the following for each application:

- The WebSphere cell in which the application is deployed
- The WebSphere servers and clusters to which it is deployed
- The physical servers on which these application servers are running
- The business area owner
- The application support contact

For each application, contact the business owner and determine the following:

- Required hours of availability
- Some relative importance indicator of this application to the organization as a whole
- A first cut of a service level required
- Whether the application can be broken into components. For example, are there some requests that should get higher priority then others.
- Any special considerations the business owner believes exist
- Whether they are willing to have their application dynamically managed by WebSphere Extended Deployment

For each application, contact the application support area and determine the following:

- What sort of resources the application uses, for example, data sources and JMS definitions
- What type of dependencies exist on other applications

For each application, contact an area that can provide current performance information, such as the following:

- Typical daily profile of application usage
- Typical daily response time breakdown
- Typical weekly profile of usage

After you gather the above information, you are ready to start defining what definitions you will set up for dynamic operations.

Determine application suitability

WebSphere Extended Deployment is specifically designed to help manage applications that are *compute-bound* and have hit rates with well-defined spikes. Compute-bound means that increasing the CPU available to the application will reduce the response time. When there are a mixture of these types of applications in the environment, WebSphere Extended Deployment can balance the overall workload using the goals specified in the service policies.

Some applications can be considered *memory bound*, for example, due to having a large number of session objects. WebSphere Extended Deployment can also assist these types of applications by making dynamic decisions in terms of how many servers are started and where requests are sent.

There could also be applications that are receiving poor response time due to overloaded back-end systems, for example, when SQL queries cannot be handled fast enough by a database server. If WebSphere Extended Deployment was to start additional servers to try and provide more capacity for the application, the result would be even more SQL requests being sent to the already overloaded database. For applications in this situation, it is best to limit what WebSphere Extended Deployment can do in terms of starting additional servers when the response time goal is not being met.

1.2.3 Dynamic operations - brief recap

Before describing how to go about determining what definitions are required, we will describe briefly the relationship between the key components in the dynamic operations feature of WebSphere Extended Deployment.

Figure 1-2 shows the relationship between the three components of the dynamic operations feature.

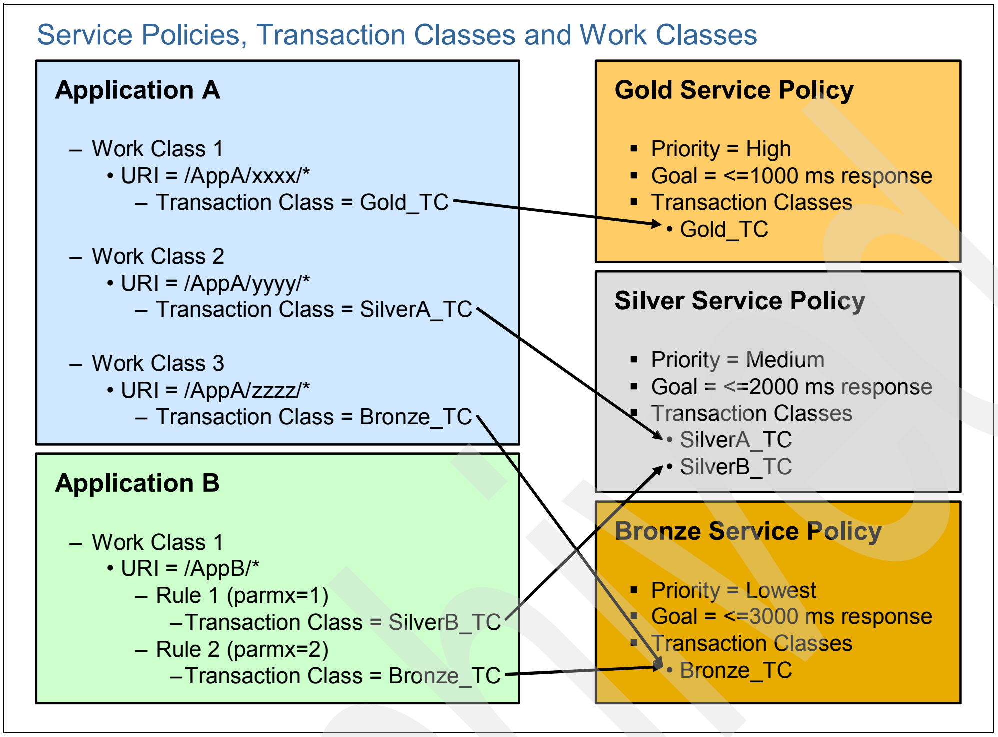


Figure 1-2 Dynamic operations overview

1.2.4 Service policies

Service policies define performance goals and relative importance of a request. The performance goal defines how requests are managed to ensure that they meet the assigned service level. The relative importance setting is used when there is resource contention to identify the most important work to ensure it is dispatched first.

Transaction classes

Transaction classes provide the link between applications and service policies. They are defined in service policies.

Work classes

Work classes define a grouping of work intended for a service policy by mapping all or a part of an application to a transaction class.

1.2.5 Defining service policies

For discussion purposes, we are assuming that we want to add WebSphere Extended Deployment into a WebSphere cell that is running a number of applications for a single business area.

Having obtained information about the applications, the first step is to determine what service policies are required.

When starting to consider what service policies to create, keep in mind that service policies are really going to come into play when your system is under load or when the mix of requests being processed by the system changes significantly. When the system is running smoothly, there is likely to be little queuing of requests in the ODR, and every request that arrives is immediately dispatched to a server to run in. However when the mix of requests changes or the load changes in some significant way, WebSphere Extended Deployment is going to use the service policies you defined to make decisions about which requests get dispatched to a server first from the ODR, and whether to stop or start servers.

Applications are mapped indirectly to service policies. Thus a single-service policy could be used to manage a number of applications, or parts of applications. For example if two applications had the same goal of achieving 90% of responses in less then 1 second, then you only need to define one service policy to manage both applications.

Another important point to note is that service policies are a goal. They are not some form of super WebSphere option that can actually make application code run faster.

How to determine service policies

After you gather information about the applications, you may end up with conflicting information. The business area may have told you they require 100% of requests to complete in less then 100 milliseconds. However your response time monitor might show you that the average response time the application actually gets in production is of the order of 1 second.

To be sure that the service policy that ends up being used for the application specifies an achievable goal, the following process should be followed for each application:

- Load test the application on a single server. This shows its response time under load and identifies where its breaking point is. It also lets you determine the characteristics of the application, for example, whether it uses a lot of CPU or spends a lot of its time waiting for IO to complete.

- Load test the application in a normal cluster over a number of servers. This determines how well the application scales.

The results of this testing should provide accurate information about how the application really performs under load and what response time it can deliver. With this information you can map the application to a realistic service policy. Additionally you can go back to the business area and advise them of what is realistic in terms of setting a service policy.

The Importance setting in service policies

The *Importance* setting can be set with values ranging from lowest to highest. We recommend that when you start defining service polices for a new environment, you do not use the values at the extreme ends; otherwise, you do not give yourself any room to maneuver if you need to add additional service policies later on.

1.2.6 Determining transaction classes

Transaction classes are a subcontainer of the service policy and provide finer grained monitoring of applications with respect to service policies.

Each application should have at least one transaction class defined for it. You can use the default transaction classes, but in an environment where you have a number of applications, using the same the transaction class does not provide a benefit.

By defining at least one transaction class specific to each application, you can use WebSphere Extended Deployment to display information showing how each application is performing.

It is possible to have more then one transaction class for an application. For example an application may have two parts, one that does queries and one that does purchases. In this case, you could define two transaction classes: one to which the query requests are mapped and one to which the purchase requests are mapped. The purchase transaction class can then be associated with a service policy that has a higher relative importance. The result is that in times of load, the ODR gives priority to the purchase requests over the query requests.

In most cases it is probably best to keep the number of transaction classes that are defined for an application to a relatively small number, perhaps two-to-four. In other words, avoid extremes. If, for example, your application had 100 distinct URLs, do not define a transaction class for each one.

When putting WebSphere Extended Deployment into place initially, we recommend defining at most two transaction classes to be used for each application: one to cover most of the application and one for some more important part of the application. Once you have this implemented in your environment, observe how WebSphere Extended Deployment manages your application, and then consider if you need to add more transaction classes for an application.

1.2.7 Determining work classes

Having determined what transaction classes you need for your applications, you then need to determine if you will require work classes in addition to the default work classes. When WebSphere Extended Deployment is implemented in a WebSphere environment, it automatically assigns a default work class to each application. This default work class uses an HTTP matching pattern of *<context root>/** as shown in Figure 1-3.

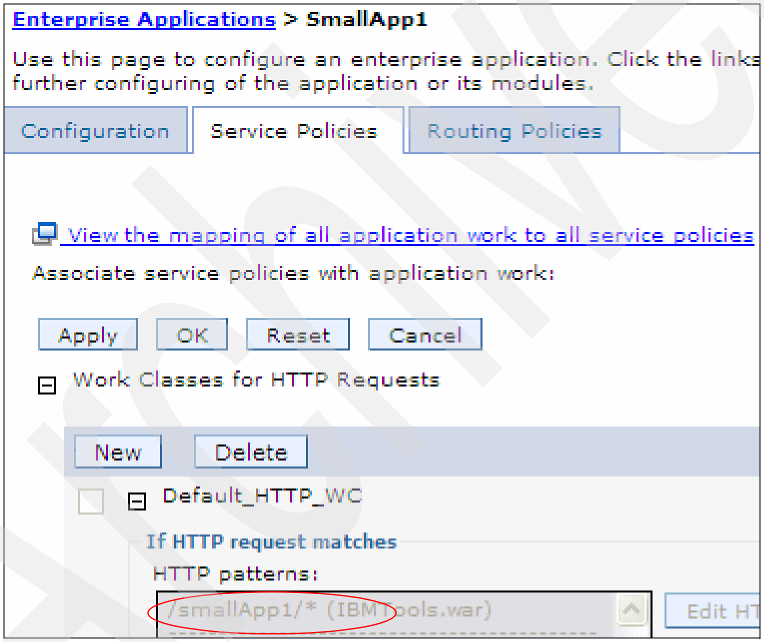


Figure 1-3 Example of a default mapping class

You cannot assign this same pattern to any new work class you create. By default, the default work class is assigned to the default transaction class and thus to the default service policy. This means that when you are planning for work classes, you only need additional work classes if you plan to split an application into multiple transaction classes so that they can be assigned to different service policies. If there is only one transaction class for an application, then you only need the default work class.

If you plan to have more then one transaction class for an application, then you need to have a work class to correspond to each transaction class. In such a case, you use the work class to define how to split up the application into the different transaction classes you defined.

1.2.8 Planning an ODR implementation

Having determined the service polices, transaction classes, and work classes you plan to define, the next step is to plan how you will implement the ODR in the environment.

For WebSphere Extended Deployment to be able to dynamically control the environment, it is necessary to have requests flow into the WebSphere environment via the ODR.

In most environments there will be some component that receives the requests and sends them to the various WebSphere application servers. There are many possible ways real WebSphere sites may control how work flows from end users to WebSphere servers, and it is impossible to describe how to implement ODR for each one. We will discuss implementing ODR for the following typical IBM solutions:

- IBM HTTP Server in conjunction with the WebSphere plug-in
- IBM Tivoli® WebSEAL
- IBM Caching Proxy, possibly in conjunction with Content Based Routing

ODR Implementation impact

It is recommended that you implement ODR in your WebSphere environment before implementing dynamic clusters or even new service policies and so on. All requests are treated as equal, which is probably how they are being treated before you implemented ODR.

The aim is to get this key piece of the WebSphere Extended Deployment solution in place in your WebSphere environment and to build confidence in its use before moving onto the next steps.

The net effect of implementing the ODR this way should be negligible. As you have not yet defined other dynamic operation definitions that would allow WebSphere Extended Deployment to use the ODR to dynamically manage the environment, the workload is handled in much the same way as it was before the ODR was implemented. Having the ODR implemented then allows you to start implementing the service policies (and other items) that you determined from your planning.

You should also implement and load test ODR in a pre-production environment before implementing ODR in your production environment.

IBM HTTP Server and plug-in

Figure 1-4 shows a typical WebSphere environment that uses the IBM HTTP Server (IHS) and the Web server plug-in.

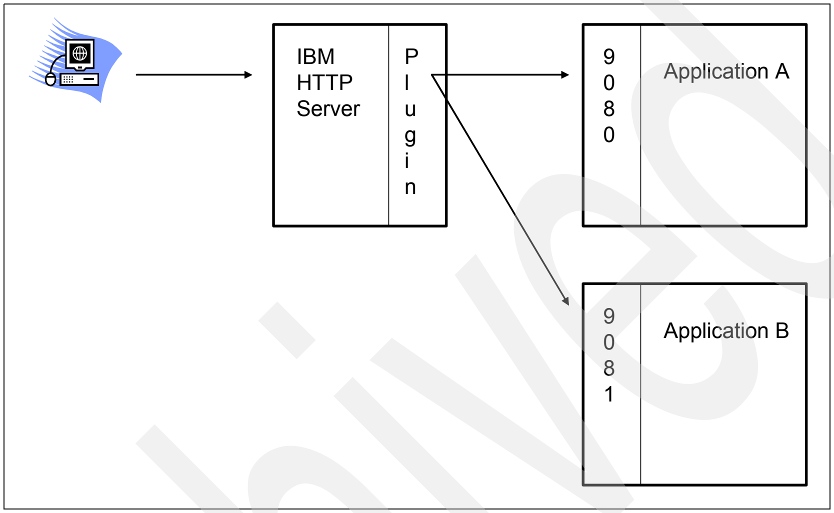


Figure 1-4 Diagram showing typical IHS setup

A larger production environment may have several IHSs with a load balancer spraying requests across them but, for purposes of simplicity, leaves that to one side and just uses the simple case of a single IHS.

The simplest approach is to use the following process:

1. If automatic propagation of the plug-in file to the IHS is enabled, disable this functionality. You need to do this since you will install a new plug-in file that will send all requests to the ODR.
2. Propagate the ODR enabled plug-in to the IHS.
3. You may need to restart the IHS for the new ODR enabled plug-in to be picked up.

The end result is shown in Figure 1-5 on page 17.

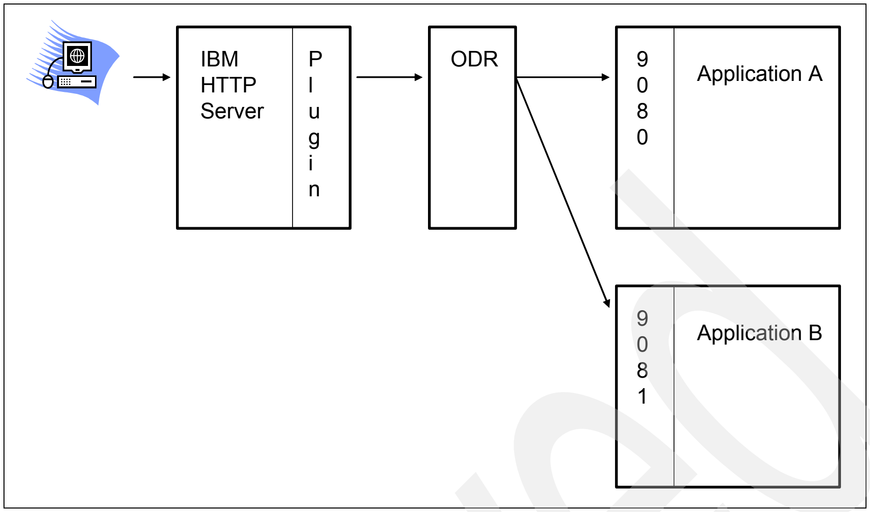


Figure 1-5 WebSphere environment with ODR implemented

IBM Tivoli WebSEAL

WebSEAL is part of the IBM Tivoli Access Manager product. WebSEAL typically runs in DMZ performing an authentication and authorization function. It is also a proxy.

WebSEAL uses the concept of *junctions* as a mechanism to control routing of requests it receives to backend servers.

A commonly used configuration is shown in Figure 1-6 on page 18.

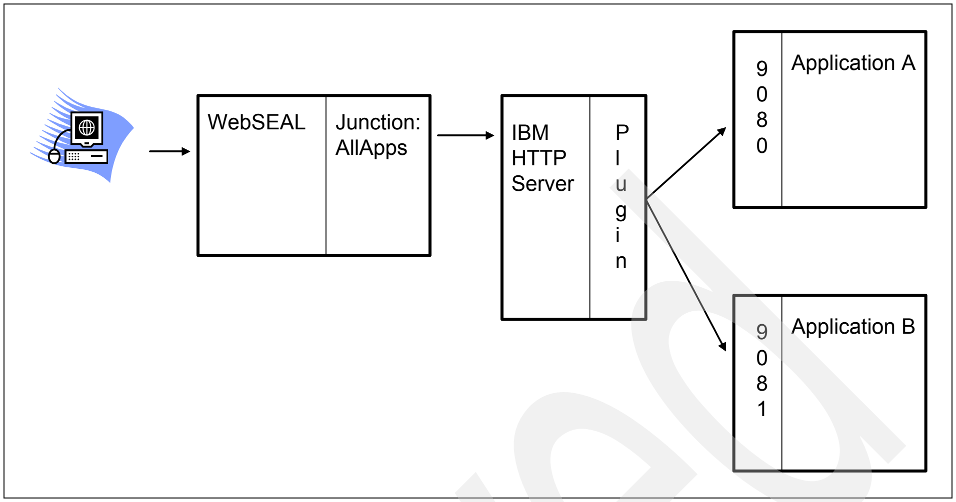


Figure 1-6 WebSEAL configuration with IBM HTTP Server

In this configuration, there is only one junction used by applications. This junction is configured to send all requests to an IBM HTTP Server that has the standard Web server plug-in. For these type of configurations, the ODR would be added in the manner described in "IBM HTTP Server and plug-in" on page 16.

Figure 1-7 shows another typical WebSEAL environment in use with WebSphere.

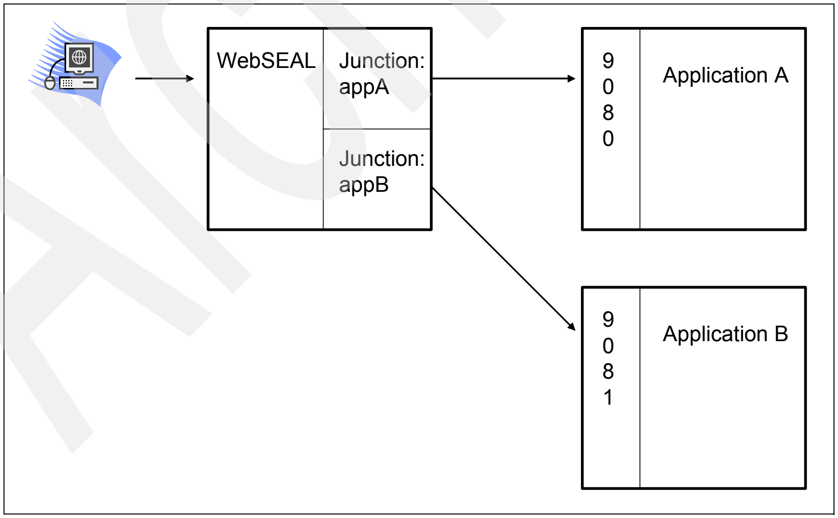


Figure 1-7 WebSEAL direct to WebSphere configuration

In Figure 1-7 on page 18, there are two junctions. The junction appA is configured to send requests to the WebSphere application server where application A is running. The junction appB is configured to send requests to the WebSphere application server where application B is running. This configuration is used when there is no requirement for an HTTP server to be placed between WebSEAL and WebSphere servers.

When adding the ODR to this type of configuration, consider the way the multiple junctions are being used as this may well influence how you intergrate the ODR.

Reduce many junctions to one junction

In this approach, you would replace all the different junctions with just one junction, and add the ODR as shown in Figure 1-8.

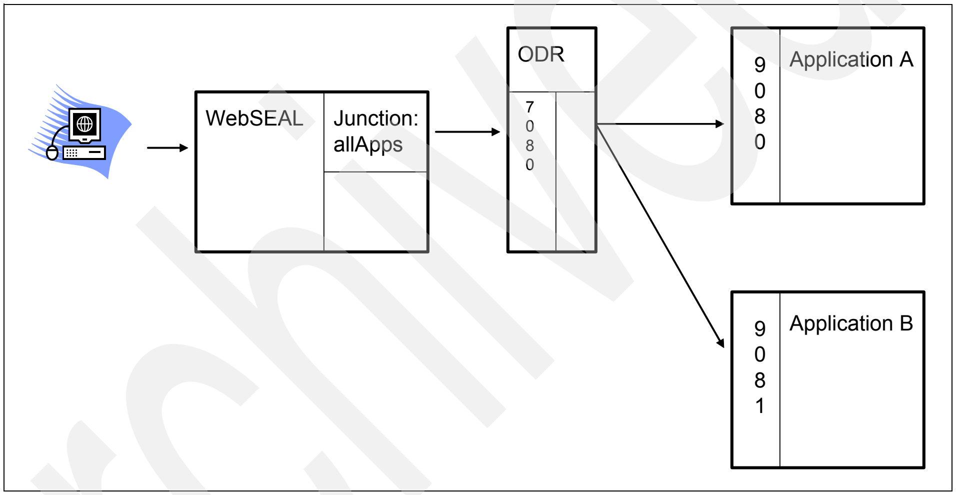


Figure 1-8 Reducing many junctions to just one

Following are the advantages of this approach:

- Simplest configuration.
- Only one junction to administer.
- As new applications are added, you no longer need to add more junctions.

Following are the disadvantages of this approach:

- The amount of effort to move from many junctions to one junction are directly proportional to how many junctions there are to start with, as will require the changing of links that have the old junction names and so forth.

- You need to review and modify the Tivoli Access Manager access rules being applied to ensure that they are applied in the same manner when there is only one junction.

Map all junctions to the one ODR proxy port

In this approach, all the current junctions are maintained, but all the junctions are modified to send their requests to the one port on the ODR as shown in Figure 1-9.

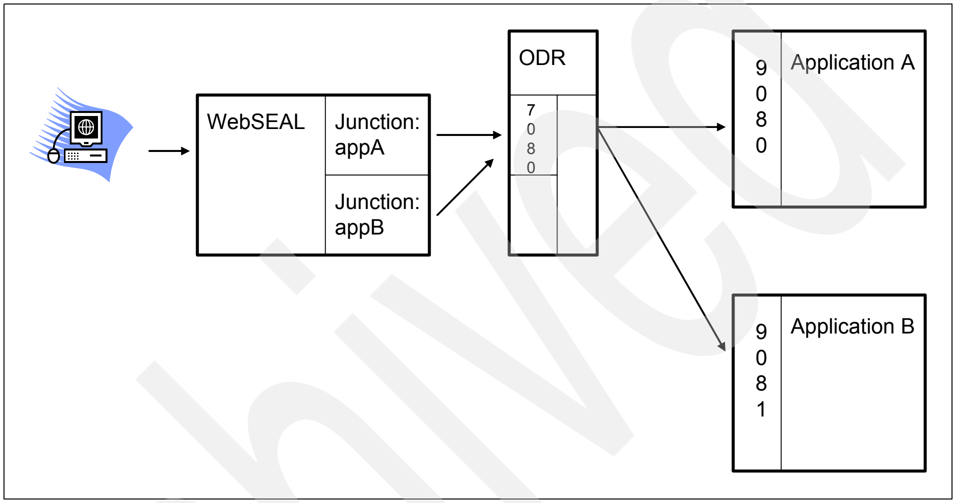


Figure 1-9 All junctions sending to the one ODR proxy port

The advantages of this approach include the following:

- There is no need to modify any links used to access the applications as the junction names are unchanged.
- It is relatively easy to implement.

Following are the disadvantages of this approach:

- It is a less than elegant overall solution because with ODR in place there is no real need for numerous junctions.
- If you have an older version of WebSEAL, there can be unexpected problems caused by having all junctions pointing to the same TCP/IP and port of the ODR. Typically these are problems to do with WebSEAL modifying the replies being sent to the user where the junction name had to be added in.

Other options

If you have concerns about configuring multiple WebSEAL junctions to the same TCP/IP and port combination where the ODR is located, then it is possible to configure the ODR to have multiple proxy ports. You could then configure the junctions to point to the same TCP/IP address of the ODR, but different ports in the ODR. We did some basic tests on this. We configured the ODR to have two proxy ports and then sent a small load to both proxy ports and all requests ran successfully. However this approach has not been submitted to any large scale testing by IBM.

Another approach is to have multiple ODR servers between WebSEAL and the WebSphere servers. An ODR could be set up for each WebSEAL junction. However, this would then create a more complicated environment to maintain.

IBM Caching Proxy and Content Based Routing

Figure 1-10 shows a typical WebSphere environment that uses the IBM Caching Proxy and Content Based Routing (CBR).

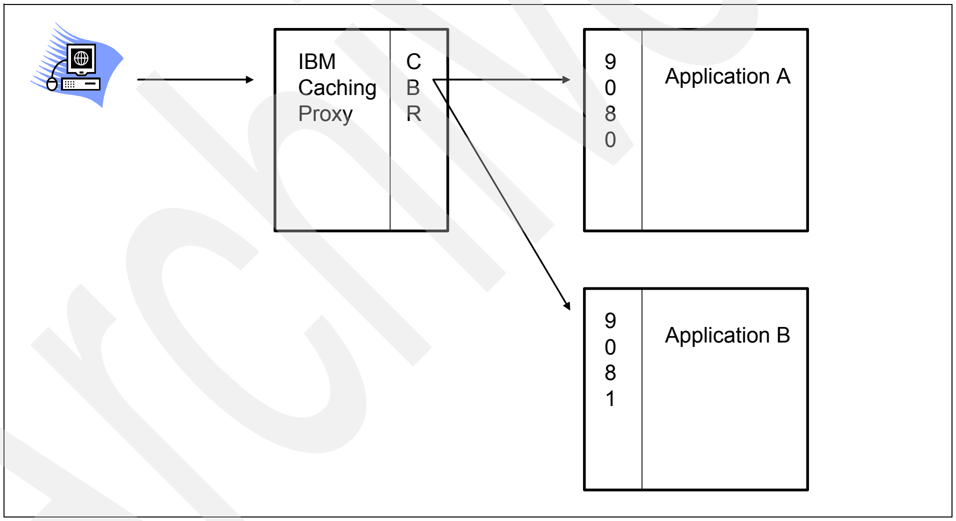


Figure 1-10 IBM Caching Proxy and WebSphere

A larger production environment may have several caching proxies with a load balancer spraying requests across them but, for purposes of simplicity, will leave that to one side and just use the simple case of a single IBM Caching Proxy installation.

The CBR would typically have several rules defined to control routing for the various applications. However we now want the caching proxy to send all requests to the ODR, as it will now be the role of the ODR to queue and route requests to the actual WebSphere servers. You could also migrate to the ODR on a more gradual basis, by first modifying a subset of application-specific CBR rules to route those requests to the ODR.

Figure 1-11 shows the implementation of ODR with IBM Caching Proxy.

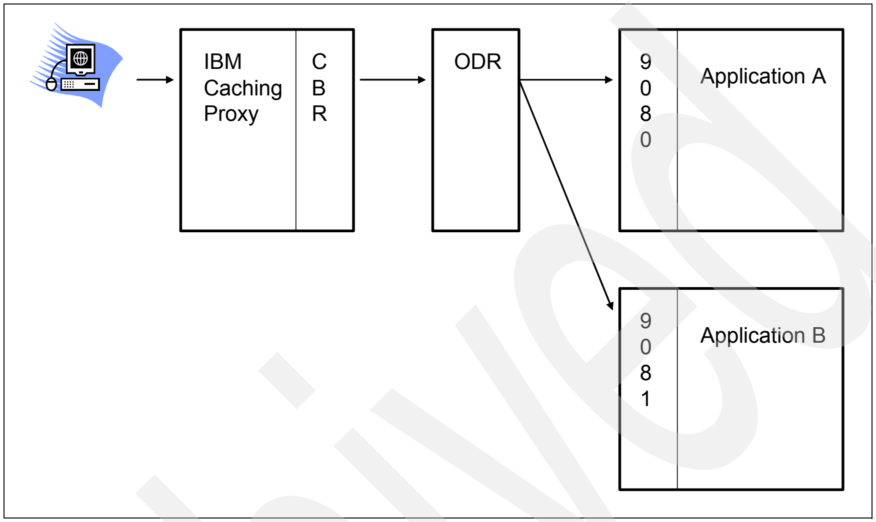


Figure 1-11 Implementation of ODR with IBM Caching Proxy

You could consider removing the use of the CBR all together. However you may want to retain the use of the CBR if you have other requests that you want to handle in some special way. For example, you may want to route requests for static pages to a set of HTTP servers.

1.2.9 Node group and dynamic clusters

Without WebSphere Extended Deployment, the typical approach is to have a fairly static environment. For example, there are typically a specific number of servers or clusters defined to run on a specific number of nodes. With WebSphere Extended Deployment the aim is to let the product determine how best to utilize the resources available. To achieve this you need to define the following two components:

- Node groups
- Dynamic clusters

Node groups

Node groups represent resource pools that workload can be spread over. The resource pool consists of the WebSphere nodes.

Dynamic clusters

Dynamic clusters are simply clusters that could contain any number of running servers, with the number of started servers being determined by WebSphere Extended Deployment in response to workload demand. A dynamic cluster can only start servers in the node group to which it is assigned.

1.2.10 Planning node groups

The ideal node group design is to have one node group in which all available WebSphere nodes of the cell are defined. This gives WebSphere Extended Deployment the greatest degree of flexibility, allowing it to start servers on any node in response to workload demands.

However there are valid reasons why you may not be able to have just one node group, such as the following:

- Your applications use a product that is only available on specific nodes in the cell.
- A hardware feature, such as a crypto card, is only available on some nodes.
- Political reasons, for example, a group may claim exclusive ownership of a set of nodes and refuse to let them be used by other applications.

You need to understand if your environment has any of these or other constraints and then plan what node groups you will need accordingly. When faced with these types of constraints you may well need to think if there is a way to remove them. For example where there is a group that will not let their nodes be shared, perhaps you can point out that their application is being constrained at peak times due to insufficient capacity. You can advise them that by agreeing to share hardware with other groups, they gain access to additional resources during peak time and improve the performance of their application.

Additionally it may be that you have a very large number of applications and nodes. In such a circumstance it may make sense to split your nodes into a few small groups and then spread the applications across these nodes. The reason it may make sense to do this is that it may make it simpler in terms of managing external resources used by the applications, for example, number of connections to databases.

1.2.11 Planning dynamic clusters

The ideal WebSphere Extended Deployment design is to have one application deployed per dynamic cluster. This gives WebSphere Extended Deployment the greatest flexibility in dynamically managing the workload. However, note that this is not mandatory.

Having each application in its own dynamic cluster means that if WebSphere Extended Deployment detects that an application is not meeting its service policy, it can dynamically manage the environment to improve the performance of the application. For example, it may make sure that those requests move to the top of the dispatch queue in the ODR or it may dynamically start another server. However, if the new application server has additional applications deployed to it, those applications are made available as well. This may or may not have any impact.

Sites that already have every application deployed to its own application server can easily migrate their configuration to Extended Deployment with the greatest level of flexibility in terms of dynamic operations.

Sites that have more than one application deployed to an application server need to consider the reasons they are currently using that approach and whether they want to split each application out into its own application server. This may not be such a big decision if there are only two applications in the same server. However where there is a much larger number of applications deployed to a single server, sites need to consider whether they have the resources to split each application into its own server. For example, suppose there are ten applications in a server. Creating nine new servers can result in significantly more memory being used since there is now the potential for ten JVMs to be up and running simultaneously, where as before there was only one. Note that the impact of this is reduced with WebSphere Application Server V6.1 because it loads common Java™ classes into shared memory, which reduces the impact of having multiple servers running.

If you have two applications deployed in the same server, both of which call EJBs within their own and the other application, we recommend that you configure WebSphere with the prefer local option for calling EJBs. If you were to split these tightly coupled applications into separate servers, then the inter-application EJB™ calls become a much more expensive operation than before. In such a case it may make more sense to keep the applications in the same dynamic cluster.

If you have a large number of applications in the environment it may be that you have some key applications that either handle a large proportion of the workload or are highly valued for some business reason. It makes sense to have these sorts of applications in their own servers so they gain the maximum benefit from WebSphere Extended Deployment. Where there are low-use applications or those with a relatively low business value sharing a server, it may not be worth the additional overhead of splitting them out into separate servers.

As with everything, look to find the appropriate balance in terms of deciding if you will put every application into its own server or not.

1.2.12 Planning the implementation of dynamic operations

Having planned all the components involved with dynamic operations, you have now reached the point where it is time to implement them.

One approach is the big bang approach. In this approach you define all the new associated definitions, implement them, and then check the result. The end result of the big bang approach could well be a big bang in your WebSphere environment. Not so much due to a product failure but due to the introduction of so many changes at once.

We recommend that you gradually introduce dynamic operations into your WebSphere environment. It goes without saying that you should first test your new dynamic operations implementation in a pre-production environment before moving to production.

Define service policies and transaction classes

The first step is to define your service policies and transaction classes. Until you change the work class definitions in the deployed applications, defining new service policies and transaction classes has no effect.

Change application work classes

Each application has a default work class associated with it that is mapped to the default service policy (a result of installing Extended Deployment). The next step is to start changing the work class in the deployed applications to map to the new transaction classes. We recommend that you start this process on your least used applications and work towards your most heavily used applications. As you make these changes, WebSphere Extended Deployment will start to queue and prioritize requests in the ODR based on the new service policies coming into effect.

Conversion of applications to dynamic clusters

Next, define your new node groups and dynamic clusters. Select one application and remap it from its static server to the dynamic cluster. Check how WebSphere Extended Deployment manages the application now that it can fully dynamically control it. Repeat this process for each application, one at a time.

1.2.13 Planning application deployment approach

The Application Edition Manager (AEM) feature of WebSphere Extended Deployment provides greater flexibility with regard to how applications are deployed into WebSphere.

Without AEM

Many organizations require that their applications be available on a 24x7 basis. When an updated version of an application needs to be deployed into a standard WebSphere production environment the level of control available to keep the application available during the deployment is not ideal. It can be done but typically involves manual intervention or the use of scripts. A greater issue is that it can be difficult to have both versions of the application available at the same time in order to have the new version tested by a limited set of users before releasing it to all users.

With AEM

AEM provides four methods of activating new versions of applications. These are described in detail in the IBM Redbook *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153.

Following are the four methods of activating new versions of applications:

Simple Marks an application edition as available to be started

Concurrent Allows you to activate the same edition of an application on different servers or clusters

Validation Activates an edition on a clone of the original deployment target

Rollout Activates one edition in place of another

Since you now have four ways of activating updated versions of applications, you need to plan what approach you will use. You need to discuss these options with the application developers and with the business areas that these new versions affect. Some of the new options will allow test users to test the deployed version of the new application before it is made generally available. This helps to prevent your real end users from accessing a new version that is still in test.

1.2.14 Monitoring the results

Once you have implemented dynamic operations into your WebSphere environment you should monitor the systems to ensure they are handling the workload in the manner you expected. You may find that you need to tune your service policies to better reflect the response times you see.

1.2.15 The benefit of dynamic operations

The benefit to the organization of implementing dynamic operations is that you are now managing the environment more at a business level, rather then just at the lower level of URLs and so forth.

Using service policies, you clearly identify and prioritize the applications that implement the business processes within the organization. You have a way of verifying that the SLAs you have with the business areas are being met.

Perhaps most importantly, you can let the WebSphere environment manage itself in terms of processing the workload so that the service policies you defined are met. This enables the WebSphere environment to respond much faster and in the manner you require then would otherwise be possible if left to manual human intervention. Additionally this reduces your support costs.

As a result, business areas can bring new initiatives to market much faster than they could have done otherwise. Consider for example a business area that decides to have a promotion they know will generate a temporary increase in load on the system. Before the implementation of dynamic operations, the extra load might have created the need to acquire a temporary server, even though there was spare capacity on other existing servers. This process would typically take considerable time, time which the business area might not be able to afford. With dynamic operations, the existing environment could most likely absorb the additional load by spreading it across existing servers, with no need to acquire an additional server.

1.3 Implementation - hands on example

This section demonstrates using the approach described in 1.2, “Planning for service level optimization” on page 7 to implement WebSphere Extended Deployment in an existing WebSphere Network Deployment cell.

As we go through this example, we show you the results of our configuration activities. The details on how to do these can be found in the redbook *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153.

1.3.1 Sample scenario

As a sample scenario, lets follow the process taken by a fictitious customer called ITSO-Area-305 to implement WebSphere Extended Deployment into their existing WebSphere Application Server Network Deployment environment. The following personnel was involved in the implementation:

- Business manager
- Application manager
- WebSphere technical contact

The approach we used is as outlined in 1.2, “Planning for service level optimization” on page 7.

During the following sections, there are various displays showing response time etc. We used a load generation tool to put load onto the environment so that the various displays that show performance information have a semi-realistic look to them.

1.3.2 Obtaining the WebSphere configuration

The WebSphere technical contact provided the diagram shown in Figure 1-12 of the existing Network Deployment environment. It shows the cell layout and in which servers the customer applications were deployed. Note the deployment manager is not shown.

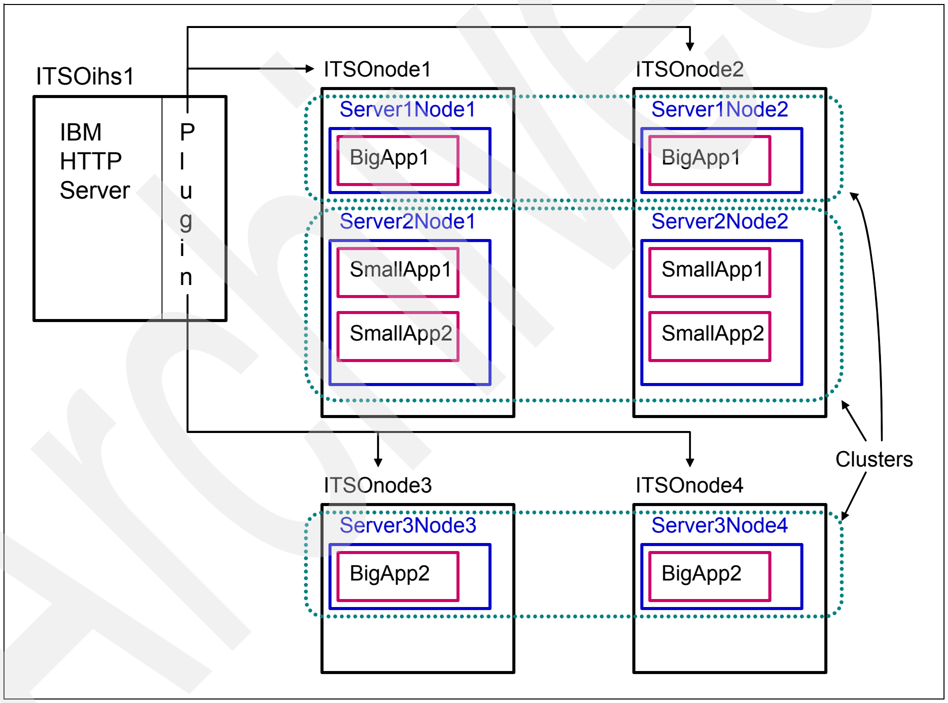


Figure 1-12 Starting WebSphere configuration

1.3.3 Gathering business requirements

The business manger then provided information about the applications that ran in the cell and about their service level requirements, including such things as when the applications needed to be available, what response time goals were expected, and the relative importance of the applications. This information took into account information about the user base and any future changes predicted for the usage of the applications.

The current environment included one set of users on the east coast and another set on the west coast. Daily reports showed that users of the BigApp1 application were getting slow response time during part of the day. Technical support determined that the servers running the BigApp1 application were getting overloaded during parts of the day.

The information obtained is summarized in Table 1-1:

Table 1-1 Business manager input on current conditions

Name	Required availability	Desired response time	Other comments	Relative importance
BigApp1	24x7	less then 20 msec	Getting slow response at times during the day	1
BigApp2	24x7	less then 50 msec	----------------	2
SmallApp1	24x7	less then 100 msec	----------------	3
SmallApp2	24x7	less then 100 msec	----------------	3

1.3.4 Gathering application information

The application development manager provided information regarding the technical details of the applications. This would normally include external dependencies the applications have such as crypto cards, back-end databases and so forth; however, in this case there were none. The information is summarized in Table 1-2.

Table 1-2 Application development manager input

Name	Dependence
BigApp1	None
BigApp2	None
SmallApp1	None

Name	Dependence
SmallApp2	None

1.3.5 Gathering performance information

Next, the information about the response time data for the applications on a typical working day was collected. The response time and request rate data appear in Table 1-3.

Table 1-3 Response time and request rate

Time	**Big App 1**		**Big App 2**		**Small App 1**		**Small App 2**	
	Avg Resp Time	Req Rate	Avg Resp Time	Req Rate	Avg Resp Time	Req Rate	Avg Resp Time	Req Rate
14	13	55	44	34	78	11	99	7
15	15	66	48	48	105	7	112	2
16	17	60	66	66	65	4	77	4
17	25	44	56	132	89	2	86	2
18	16	23	47	76	46	1	91	3

Note the data in the above table is fictional and used only for demonstration purposes. They may not necessarily line up with subsequent response time pictures. Again the purpose here is to give a hypothetical example of the process only.

1.3.6 Determining the service polices

Having gathered the information, the process of determining what was needed for defining dynamic operations could begin. The data in Table 1-3 was used to determine what the different applications were achieving in the way of average response time. For the BigApp1 application the average response time was in the order of 20 msec. The BigApp2 application the average response time was abut 50msec. The SmallApp1 and SmallApp2 applications had average response times of about 100msec.

Using this data, the service policies were determined. Table 1-4 on page 31 shows the service policies to be defined based on the observed average response times and the relative importance of the applications.

Table 1-4 Service policy information

Name	Goal	Relative importance
Gold	less then 20 msec	1
Silver	less then 50 msec	2
Bronze	less then 100 msec	3

1.3.7 Determining the transaction classes

The transaction classes were defined next, as shown in Table 1-5.

Table 1-5 Transaction classes

Name	Transaction Class	Associated Service Policy
BigApp1	BigApp1-TC1	Gold
BigApp2	BigApp2-TC1	Silver
SmallApp1	SmallApp1-TC1	Bronze
SmallApp2	S,mallApp2-TC1	Bronze

1.3.8 Determining the work classes

At this stage there were no plans to split any of the applications into different transaction classes, which meant that we could use the default work classes created when WebSphere Extended Deployment was installed.

1.3.9 Installing WebSphere Extended Deployment

The WebSphere Extended Deployment product code was installed into the existing WebSphere Application Server Network Deployment environment.

The installation process consisted of the following two steps:

- Installing the WebSphere Extended Deployment product code
- Augmenting the existing profiles

We recommend that when installing WebSphere Extended Deployment on a server, and by that we mean a machine, that you stop all the WebSphere processes on that server before you start the install. Where you have a cell that consists of more then one machine, stop all WebSphere process on one machine, perform the install, restart the WebSphere processes, and then repeat on the next machine. The machine with the deployment manager should be done first.

The same recommendation applies to the augmentation of the profiles.

Checking the pre-req maintenance levels

Before installation, the prerequisites for WebSphere Extended Deployment were noted and the current environment checked to see if it met those requirements. WebSphere Extended Deployment can be installed on top of an existing V6.0.2 or V6.1 Network Deployment installation. In this case, V6.1 was installed and met the minimum maintenance level of 6.1.0.1. A check found that the system was at 6.1.0.2, which meant it met the minimum requirement.

Installing the product code

WebSphere Extended Deployment was then installed into the existing WebSphere environment. During installation, you have the option to augment the existing profiles to Extended Deployment; however, we recommend taking a cautious step-by-step process when it comes to implementing changes, thus the augmentation did not occur during the installation process.

The process followed was to perform the following steps on each node, one node at a time:

- Stop all WebSphere processes.
- Install the product code.
- Restart WebSphere processes.

After this process was completed, the administrative console was used to verify that the new code was installed successfully to each node, as shown in Figure 1-13 on page 33.

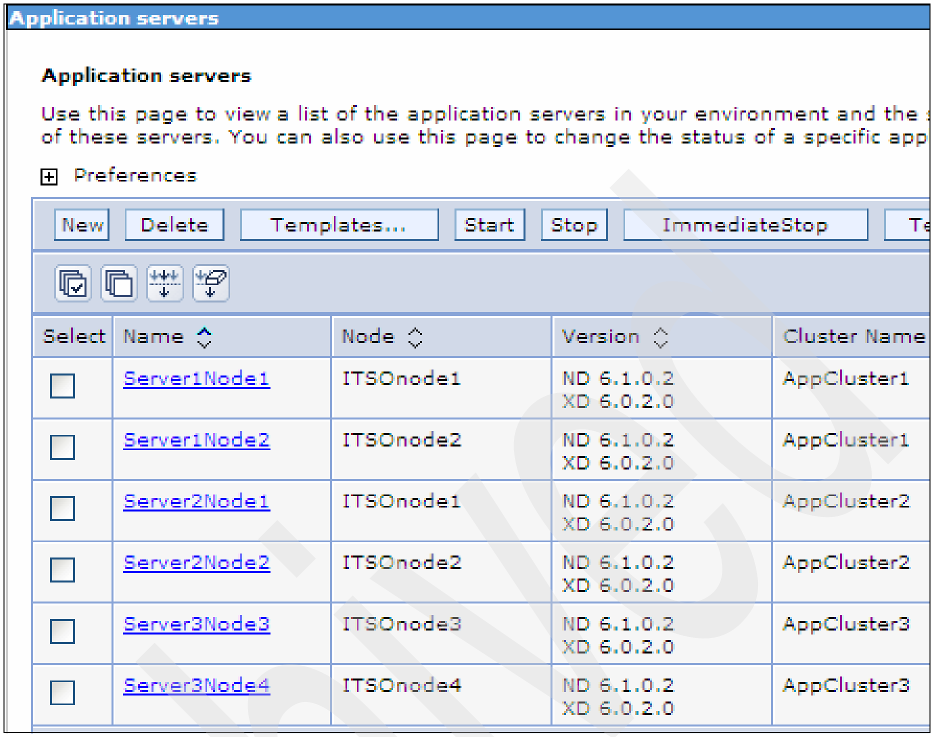


Figure 1-13 Display showing WebSphere Extended Deployment installed

Augmenting the profiles

Augmenting the profiles on a WebSphere node results in that node becoming WebSphere Extended Deployment enabled. Augment the node with the deployment manager first.

The process followed was to perform the following steps on each node, one node at a time:

- Stop all WebSphere processes.
- From the product bin directory issue the **xdaugment** command passing in the name of the profile to be augmented.
- Restart WebSphere processes.

Example 1-1 on page 34 shows the command issued for the deployment manager and the output produced. The augmentation process writes a log, in this case to the following location:

```
C:\WebSphere\AppServer61\logs\manageprofiles\Dmgr01_augment.log
```

It took about five minutes on a fairly reasonably sized machine to complete the augmentation process. Example 1-1 shows the output of a successful augment process. The process outputs an error message if you try to run it and the node agent or any WebSphere application server is running.

Example 1-1 Augmenting the deployment manager profile

```
C:\WebSphere\AppServer61\bin>xdaugment Dmgr01
INSTCONFSUCCESS: Profile augmentation succeeded.
```

After the deployment manager was restarted, new options appeared in the administrative console, including Runtime Operations and Operational Policies.

Note that the administrative console shows if the WebSphere Extended Deployment product code is installed on a node; however, it does not show if the profile on that node was augmented or not. You can verify if it has or not by looking in the file called *<install_root>*/properties/profileRegistry.xml for text similar to the following:

```
<augmentor
template="C:\WebSphere\AppServer61\profileTemplates\xd_augment"/>
```

Viewing the default dynamic operation policies

The administrative console was used to view the default policies related to dynamic operations. You can view the default topology by selecting **Operational Policies** → **Service Policy Topology**, which will produce the display shown in Figure 1-14 on page 35.

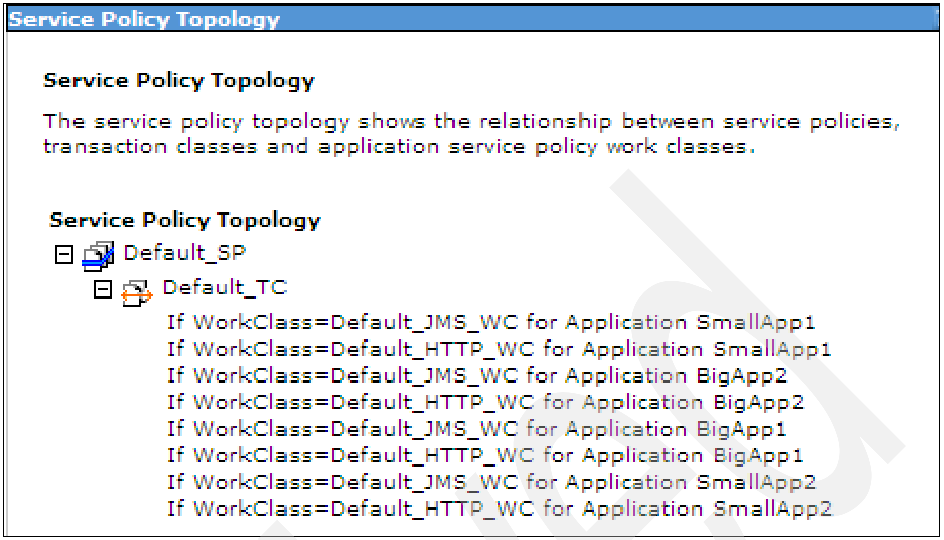


Figure 1-14 Default service policy topology

1.3.10 Integrating the ODR

With WebSphere Extended Deployment installed and all profiles updated, the next step was to define and integrate the ODR into the WebSphere environment.

ODR placement

In this example there are no firewalls or DMZs. We decided to run the ODR on its own server and that the ODR would be placed between the existing IBM HTTP Server and the application servers.

In brief remember the following:

- The ODR should not be put in the DMZ.
- You should still have some process that terminate the connection in the DMZ before proxying the requests to the ODR.

For a full discussion of determining the topology to use for on demand routing components, see the redbook *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153.

Installing the product code and defining the profile

Before defining the ODR on a new machine, the ODR must be added as a node to the existing WebSphere cell.

WebSphere Application Server Network Deployment and WebSphere Extended Deployment were installed on the new machine and a new profile, called odr1 was created and the node federated into the WebSphere cell.

The **xdaugment** command was run to augment the odr1 profile. The new ODR node then appeared as shown in Figure 1-15.

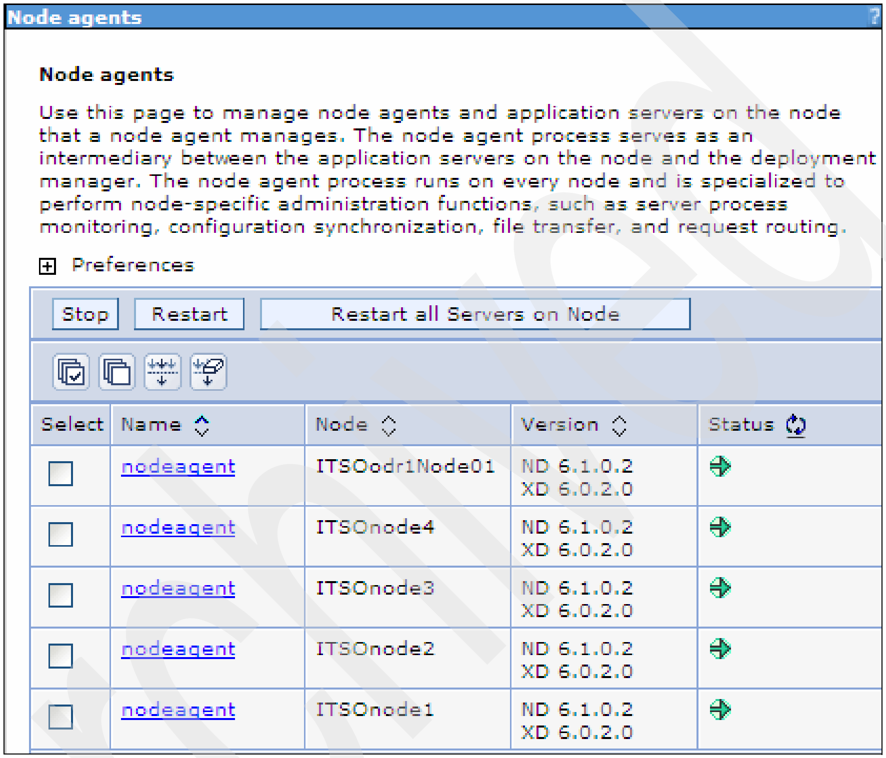


Figure 1-15 After the ODR node is added to the cell

Defining the ODR

An ODR called **odr1** was defined on the new node. In the administrative console the new ODR server could be viewed as shown in Figure 1-16 on page 37.

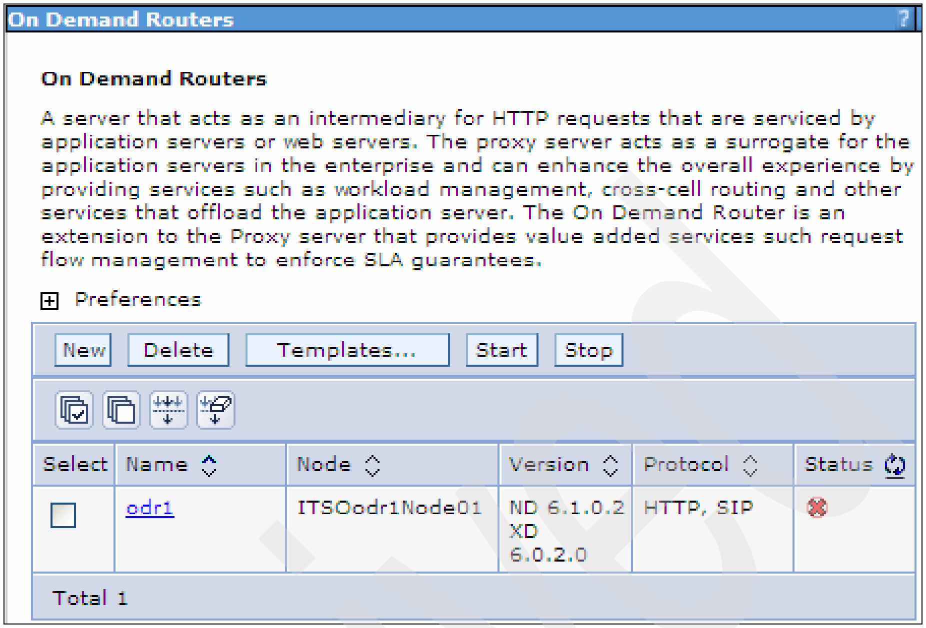


Figure 1-16 New ODR defined

The ODR was then started.

Verifying the ODR operation

Before changing the IHS to send requests to the ODR, it was verified that requests could be sent to the ODR and that they could be processed successfully.

To send requests to the ODR directly, you need to know which TCP/IP port the ODR is listening on when it comes to the proxying of requests. To find this port, select **Servers** → **On Demand Routers** → **odr1** → **Ports**. This display shows all the TCP/IP ports used by the ODR. A portion of this display is shown in Figure 1-17 on page 38.

PROXY_HTTPS_ADDRESS	*	443
PROXY_HTTP_ADDRESS	*	80
PROXY_SIPS_ADDRESS	*	5061
PROXY_SIP_ADDRESS	*	5060

Figure 1-17 ODR ports used to proxy requests

To test that the ODR was proxying standard HTTP requests, a request was sent to the ODR server on port 80, and then to test SSL requests, to port 443. Note the ODR server, since it is really a special form of the standard application server, also has the standard Web container ports. These are not used for the proxying of requests, so do not send requests to those ports.

In a real production environment, you may want to change the default ports assigned to the proxy ports.

ODR proxy logs

When an ODR server is created, by default it logs each request it proxies. These logs provide one way to investigate proxy issues dealing with the ODR. This logging is controlled by settings located by selecting **Servers** → **On Demand Routers** → **odr1** → **On Demand Router Properties** → **On Demand Router** settings. A portion of this display is shown in Figure 1-18.

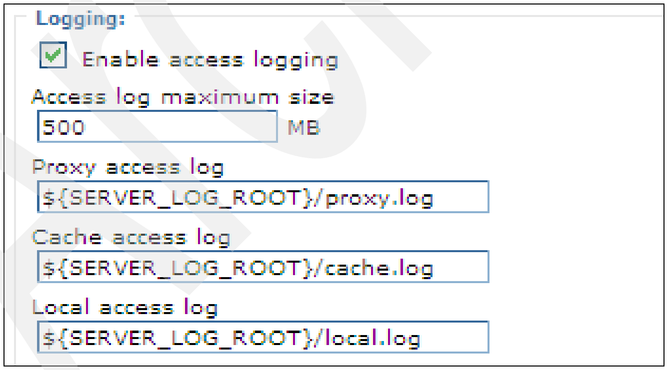


Figure 1-18 ODR Proxy log settings

Example 1-2 on page 39 shows an extract of the ODR proxy log from our system.

Example 1-2 Example of ODR proxy log contents

```
9.42.170.129 - - [23/Oct/2006:17:39:18 -0400] "GET /bigApp1/index.html
HTTP/1.1" 200 3345
9.42.170.129 - - [23/Oct/2006:17:39:18 -0400] "GET /bigApp1/rbhome.gif
HTTP/1.1" 200 3584
9.42.170.129 - - [23/Oct/2006:17:39:21 -0400] "GET /bigApp1/index.html
HTTP/1.1" 200 3345
9.42.170.129 - - [23/Oct/2006:17:39:21 -0400] "GET /bigApp1/rbhome.gif
HTTP/1.1" 200 3584
```

Disable the old automatic plug-in generation

It is necessary to disable the current automatic plug-in generation and propagation process that was in place. This is because a new process will be set in place to propagate a new plug-in that is ODR-aware to the IHS server. If the current plug-in management process was left in place, then it would interfere with the use of the ODR.

ODR proxy plug-in generation

Instead of using the standard Web server plug-in configuration file to control the routing of requests, the IHS will now use a new configuration file generated by the ODR that tells the plug-in, in IHS, to send all requests to the ODR.

The plug-in file can be generated at different scopes. Since the topology only had one IHS, the scope was set to the cell level. When an ODR is first defined, the scope defaults to `none` and no ODR plug-in is generated. Once the setting is changed to a scope, whenever the ODR is restarted, an ODR-aware plug-in file is generated. Any subsequent application deployments trigger the ODR plug-in file to be regenerated.

> **Note:** The ODR must be restarted whenever this scope setting is first set or changed. No restart is required when applications are added or removed.

A user-written script or Windows .bat file can be created and run automatically to copy the file from the ODR server to the IHS server whenever the ODR re-generates the plug-in file to copy the file.

The plug-in generated by the ODR is written to the following:

<*install_root*>/profiles/<*ODR-profile-name*>/etc/plugin-cfg.xml.

The scope and script settings can be configured by selecting **Servers → On Demand Routers → odr1 → On Demand Router Properties → On Demand Router**. Figure 1-19 shows a portion of the console page where the ODR plug-in generation scope level and script to invoke are set.

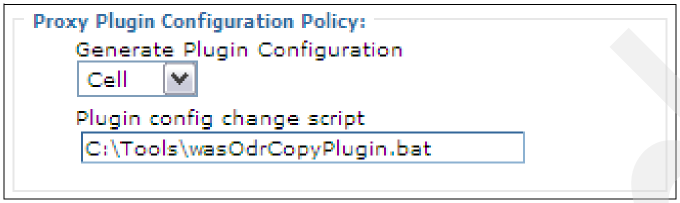


Figure 1-19 Setting plug-in scope generation and script to invoke

Note that the bat file runs on the machine where the ODR is running.

Copy the ODR plug-in file

During the initial setup of the ODR, the existing plugin-cfg.xml file was backed up and then manually replaced with the ODR-generated plug-in file.

The IHS was restarted to make sure it picked up the new plug-in file. Tests were run to ensure the applications could still be accessed successfully via IHS and the ODR proxy logs were checked to verify that requests where being proxied by the ODR.

All these tests were successful, meaning the ODR was implemented successfully.

Making the IHS a trusted server

Adding the ODR between the IHS server and application servers has an impact on applications that use the J2EE API getRemoteAddr(). Applications can use the J2EE API request.getRemoteAddr() to obtain the TCP/IP address of the client that sent the request. When we checked what value was now being received by the applications, we found that when an application used getRemoteAddr(), it received the TCP/IP address of the IHS instead of the browsers that sent the request.

To remedy this, it was necessary to tell the ODR that the IHS is a trusted server.

This was done by selecting **Servers → On Demand Routers → odr1 → On Demand Router Properties → On Demand Router Settings**, and then adding the host name of the IHS server in the text box with the heading **Trusted Security Proxies** as shown in Figure 1-20 on page 41.

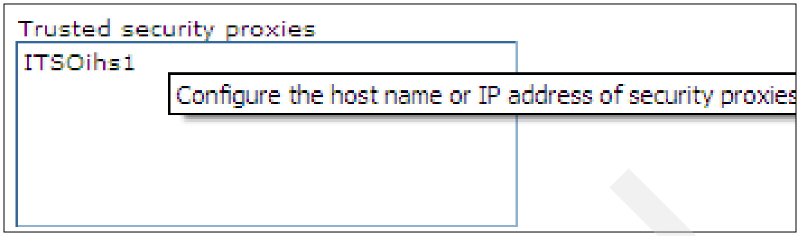


Figure 1-20 Adding the IHS node as a trusted server

After restarting the ODR, and sending some test requests, we verified that the applications now obtained the TCP/IP address of the browser again.

1.3.11 Checkpoint 1

At this stage, Extended Deployment was installed and the ODR was implemented in the existing environment. We wanted to explore how WebSphere Extended Deployment was managing the WebSphere environment at this stage, without having done any of the steps to move to dynamic operations.

Default service policy

The default service policies, transaction classes and work classes, can be viewed by selecting **Operational Policies** → **Service Policies** (Figure 1-21).

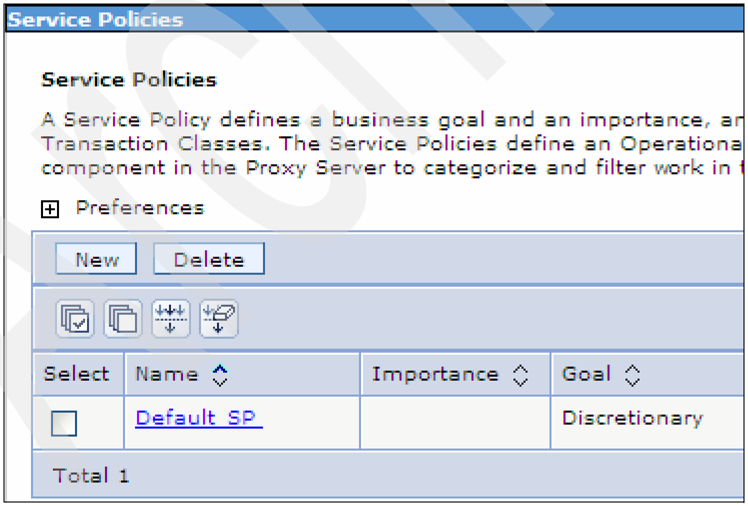


Figure 1-21 Default service policy

The default service policy has a goal of `discretionary`. This is usually used for work that does not have a significant value. However at this stage it is being applied to all HTTP requests entering the system, which means they are all treated the same, which is more or less what you had before.

Clicking on the service policy name will open a page that shows the transaction class definitions, as shown in Figure 1-22.

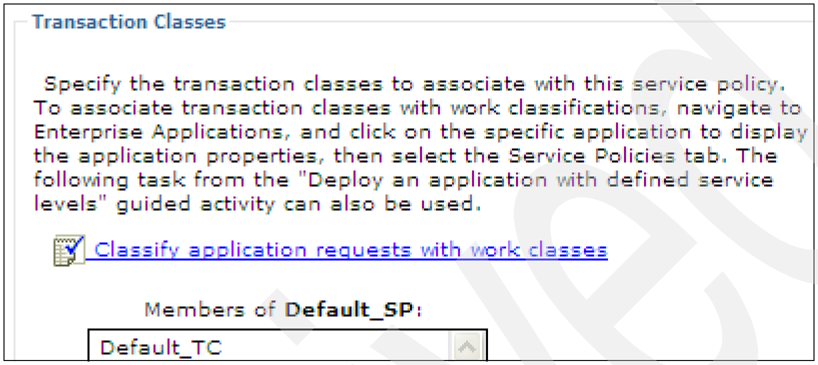


Figure 1-22 Default transaction class

Selecting **Operational Policies** → **Service Policy Topology** shows the connection of work classes for each application to transaction classes and then to service policies as shown in Figure 1-23.

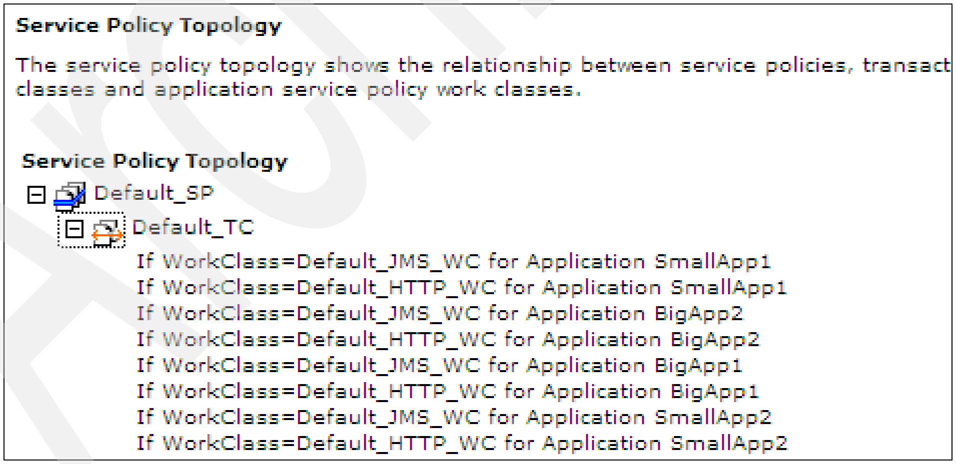


Figure 1-23 Default service policy topology

Monitoring

We also tried the enhanced monitoring capability of WebSphere Extended Deployment.

The first display was a view that showed the real time CPU usage on each node in the cell. We selected **Runtime Operations** → **Runtime Topology**, then selected **Node Group** from the drop down box labelled **Select a perspective**. This gave the display shown in Figure 1-24.

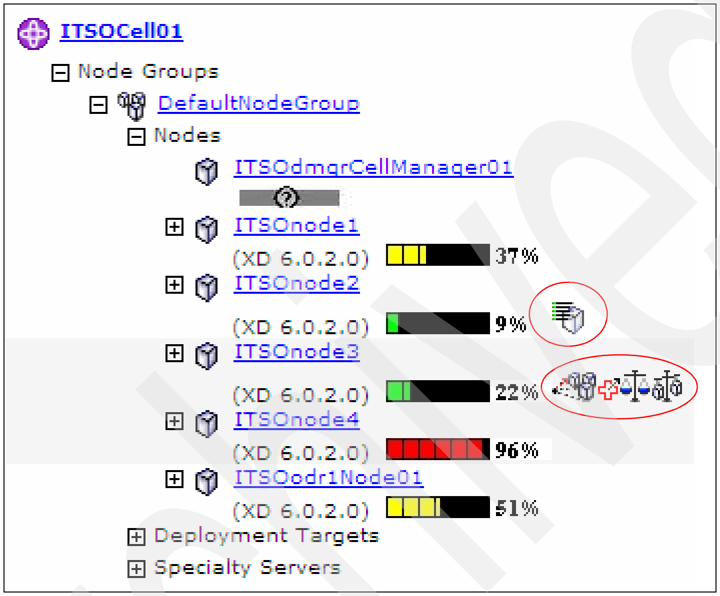


Figure 1-24 Node group CPU usage display

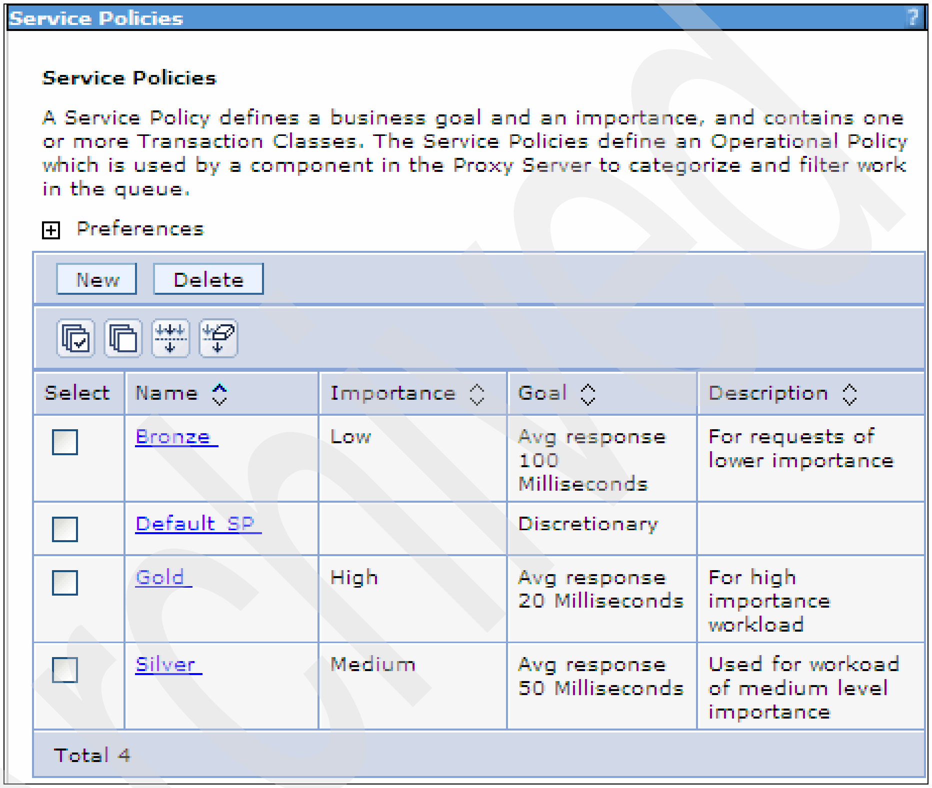
The symbols in the circle represent where various WebSphere Extended Deployment control functions are running, for example the ARFM Controller.

We also then viewed the runtime charting capability of WebSphere Extended Deployment. We experimented with this capability to view for example the average response time of all applications and how to display information about the default service policy and transaction classes.

1.3.12 Define new service policies

The next step was to start defining the service policies that were identified in 1.3.6, “Determining the service polices” on page 30 and transaction classes identified in 1.3.7, “Determining the transaction classes” on page 31. Defining these service policies and transaction classes has no effect on the operation of the WebSphere environment, since there is no change to the work class to which the applications are mapped.

New service policies and transaction classes can be created by selecting **Operational Policies** → **Service Policies**. Click **New** to start the wizard. The wizard is straightforward, asking for a service policy name, goal type, and transaction class. The results are shown in Figure 1-25.


Service Policies

Service Policies

A Service Policy defines a business goal and an importance, and contains one or more Transaction Classes. The Service Policies define an Operational Policy which is used by a component in the Proxy Server to categorize and filter work in the queue.

Preferences

New Delete

Select	Name	Importance	Goal	Description
	Bronze	Low	Avg response 100 Milliseconds	For requests of lower importance
	Default_SP		Discretionary	
	Gold	High	Avg response 20 Milliseconds	For high importance workload
	Silver	Medium	Avg response 50 Milliseconds	Used for workoad of medium level importance

Total 4

Figure 1-25 New service policies defined

Selecting **Operational Policies** → **Service Policy Topology** shows the mapping of the new transaction classes to the new service policies, as shown in Figure 1-26 on page 45.

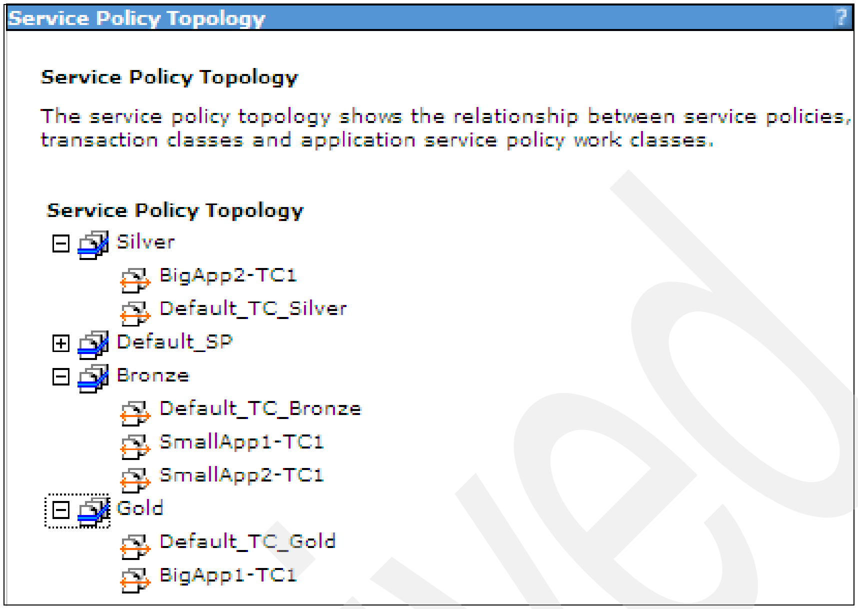


Figure 1-26 Updated service topology

When a new service policy is created, a default transaction class is automatically created as well. These new default transaction classes cannot be deleted.

1.3.13 Adding SmallApp1 to new service policy

With the service policies and transaction classes in place it was now time to start implementing dynamic operations. Using a policy of implementing change on a gradual basis, SmallApp1 was selected as the first app to use one of the new service policies.

Service policies are selected by selecting a work class and associating a transaction class to it. The transaction class is associated with a service policy.

Use the following steps to make this change:

1. Navigate to **Applications** → **Enterprise applications**.
2. Click the SmallApp1 application name.
3. Switch to the Service Policies tab.
4. Expand the Default_HTTP_WC work class, and select the SmallApp1-TC transaction class for the Bronze service policy as shown in Figure 1-27 on page 46.
5. Save the change.

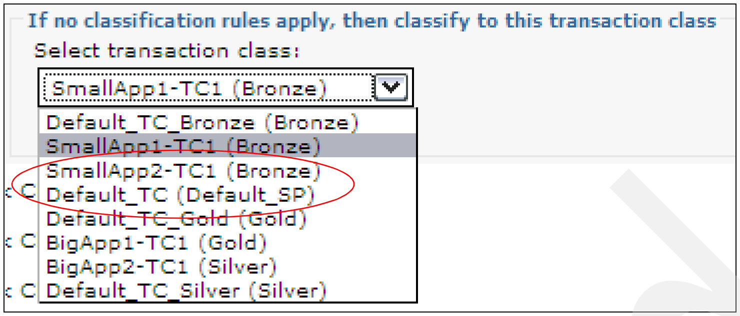


Figure 1-27 Selecting transaction class for the SmallApp1 application

To view information about the application and the service policy it is now associated with, select **Runtime Operations** → **Runtime Topology.** Figure 1-28 shows the Bronze service policy and the average response time for the SmallApp1 application.

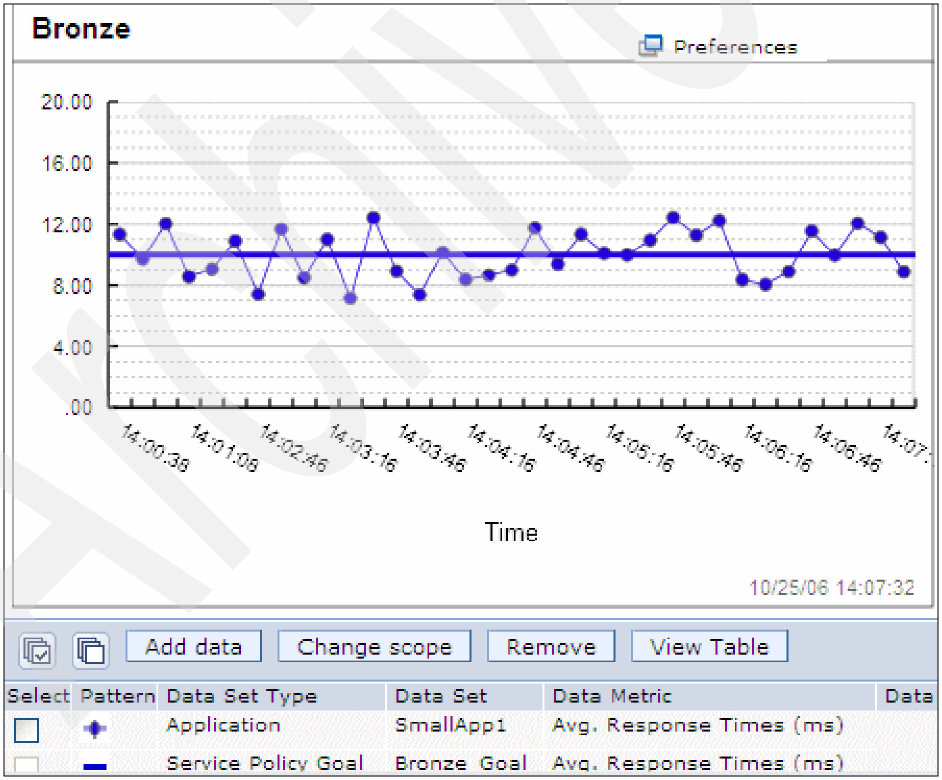


Figure 1-28 Bronze service policy and SmallApp1 response time

The graph in Figure 1-28 on page 46 shows the service policy goal of 100ms that was defined. This provides a clear indication of how the SmallApp1 application is performing against the service policy goal that was set.

Click the **Add data** button, and select the Percentile Response Time metric to be added to the graph (Figure 1-29).

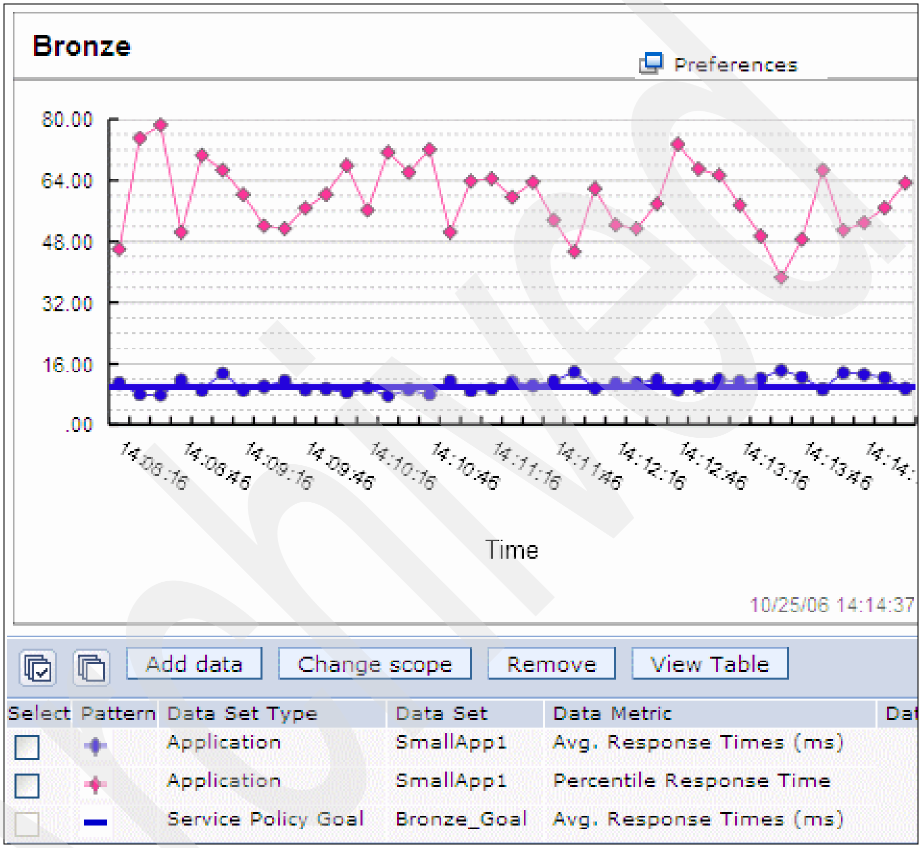


Figure 1-29 After adding the percentile response time

With this metric you can see at a glance what percentage of requests are meeting the service policy objective.

Implementing all service policies

After seeing that the WebSphere environment continued to function well after implementing the Bronze service policy for the SmallApp1 application, the following was then done:

- The default HTTP work class for the SmallApp2 application was associated to the SmallApp2-TC1 transaction class and by inference mapping it to the Bronze service policy.
- The default HTTP work class for the BigApp2 application was associated to the BigApp2-TC1 transaction class and by inference mapping it to the Silver service policy.
- The default HTTP work class for the BigApp1 application was associated to the BigApp1-TC1 transaction class and by inference mapping it to the Gold service policy.

The resulting service policy topology looked as shown in Figure 1-30.

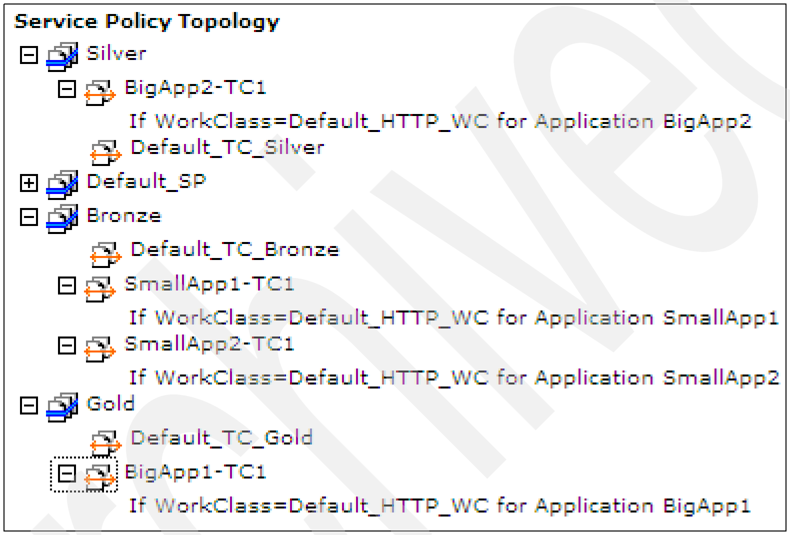


Figure 1-30 All applications now mapped to new service policies

1.3.14 Implementing the dynamic clusters

With the service policies now in use, the next step was to implement dynamic clusters. Since the impact of dynamic clusters is typically the greatest for applications that are highly used, we decided to implement dynamic clusters for the BigApp1 application.

Currently the BigApp1 application runs in the cluster topology, as shown in Figure 1-31 on page 49.

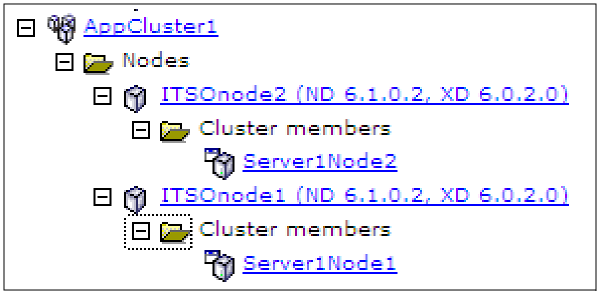


Figure 1-31 Cluster topology used to support the BigApp1 application

Defining a new node group

To implement dynamic clusters, a new node group must be defined, as dynamic clusters cannot be defined in the default node group. The best and simplest approach that gives WebSphere Extended Deployment the most flexibility in dynamically managing the workload is to put all nodes running application servers in the WebSphere cell into one new node group.

A new node group called ITSO-Area-305 was defined and the four nodes that ran applications were added to it. Node groups can be defined by selecting **System Administration** → **Node groups** and clicking **New**. Once the node group has been added, members for the node group can be identified by selecting **Node group members** under the Additional properties section. Figure 1-32 shows the nodes that are now members of the new node group.

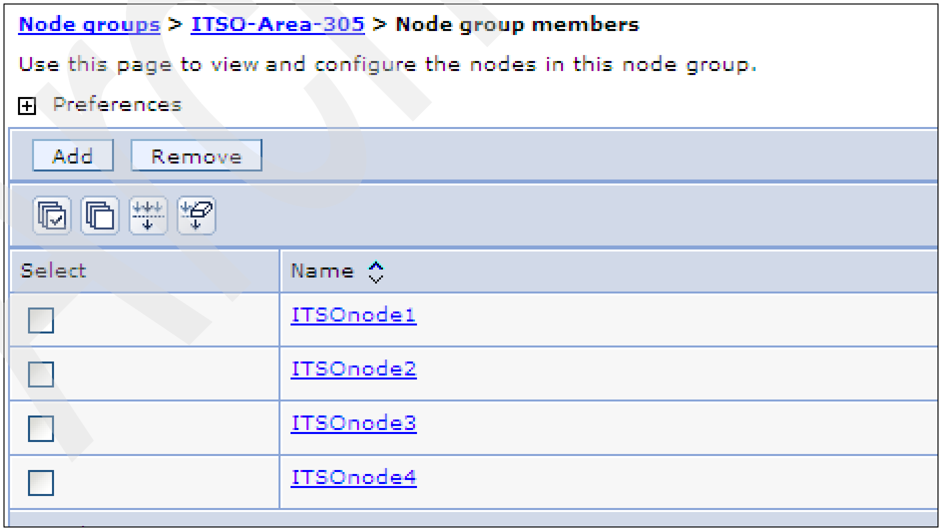


Figure 1-32 Nodes in the new node group

Migrating to a dynamic cluster - overview

The existing servers that run the BigApp1 application cannot be used in a dynamic cluster. The process to move an application to a dynamic cluster is the following:

- Create a new server template based on the server that currently runs the BigApp1 application.
- Create the dynamic cluster, referencing the new server template.
- Remap the BigApp1 application to the new cluster.

Creating the new server template

A new template based on an existing server can be made by navigating to **Servers** → **Application Servers**. Click **Templates...** A new template called BigApp1 was created as shown in Figure 1-33.

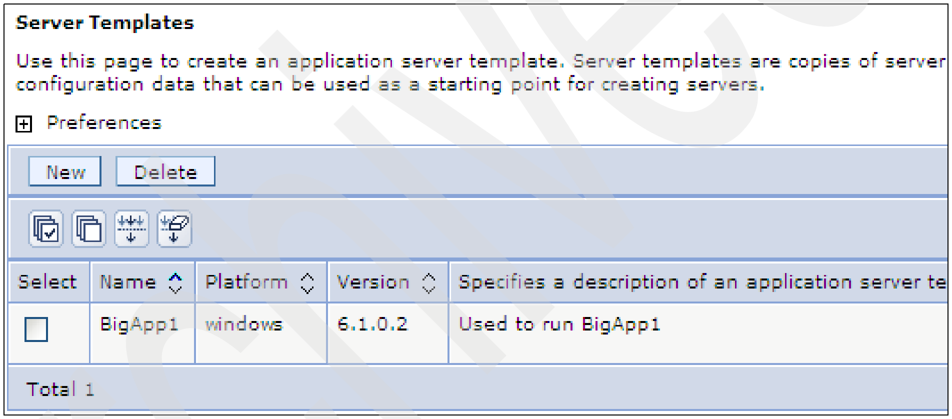


Figure 1-33 BigApp1 server template defined

Any changes in the future that need to be made to the servers that run in the dynamic cluster, should be made to this template. This ensures that they are propagated by WebSphere to any servers in the dynamic cluster.

Creating the dynamic cluster

Dynamic clusters can be created by navigating to **Servers** → **Dynamic Clusters**, and clicking **New**. A dynamic cluster called BigApp1 was created as shown in Figure 1-34 on page 51.

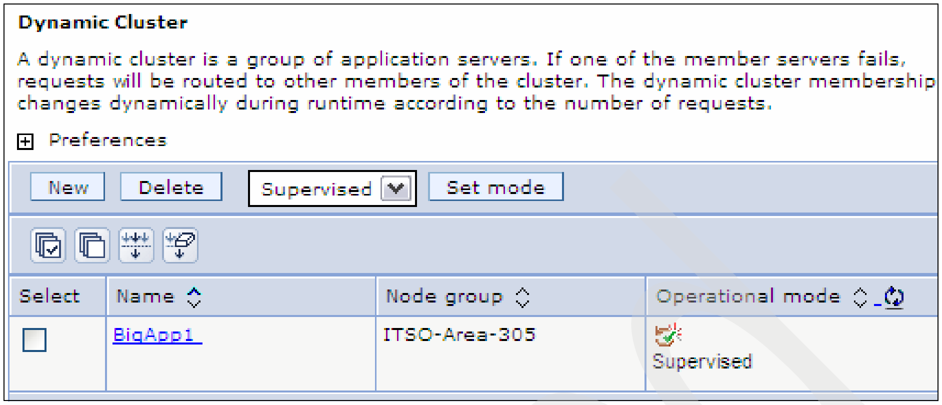


Figure 1-34 BigApp1 dynamic cluster

The operational mode was set to Supervised. This means that WebSphere Extended Deployment will raise runtime task alerts when it detects that an action related to the dynamic cluster should be done.

Additionally new servers are now defined for each node, as shown in Figure 1-35.

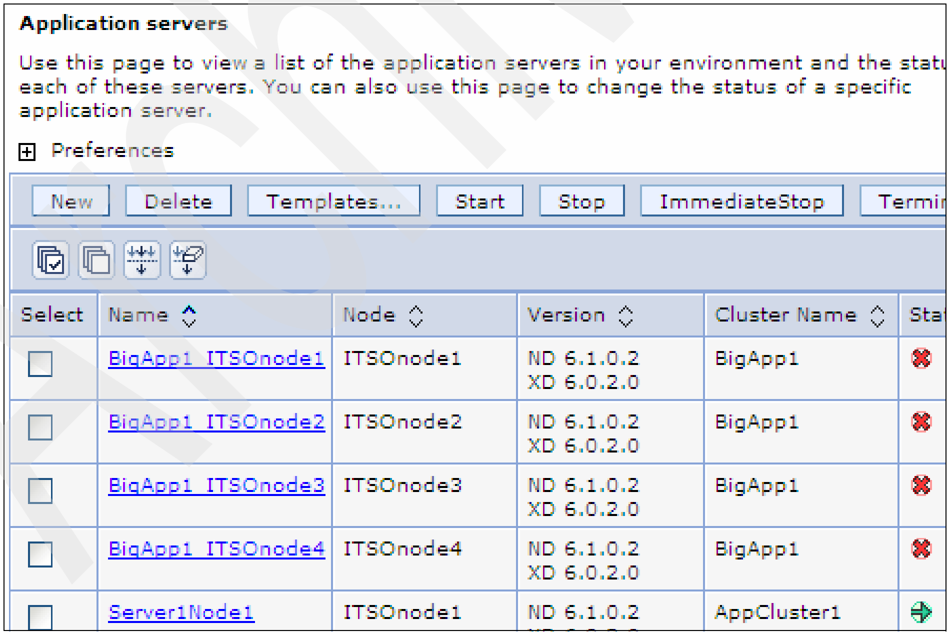


Figure 1-35 New servers for the BigApp1 application

In this case none of the applications had any associated resources such as data sources. Where there are resources associated with the existing servers or clusters, these resources should be defined at the new dynamic cluster level.

Map BigApp1 to the dynamic cluster

The final step was to map the BigApp1 application to the BigApp1 dynamic cluster. This was done using the **Manage Modules** option for an application (Figure 1-36).

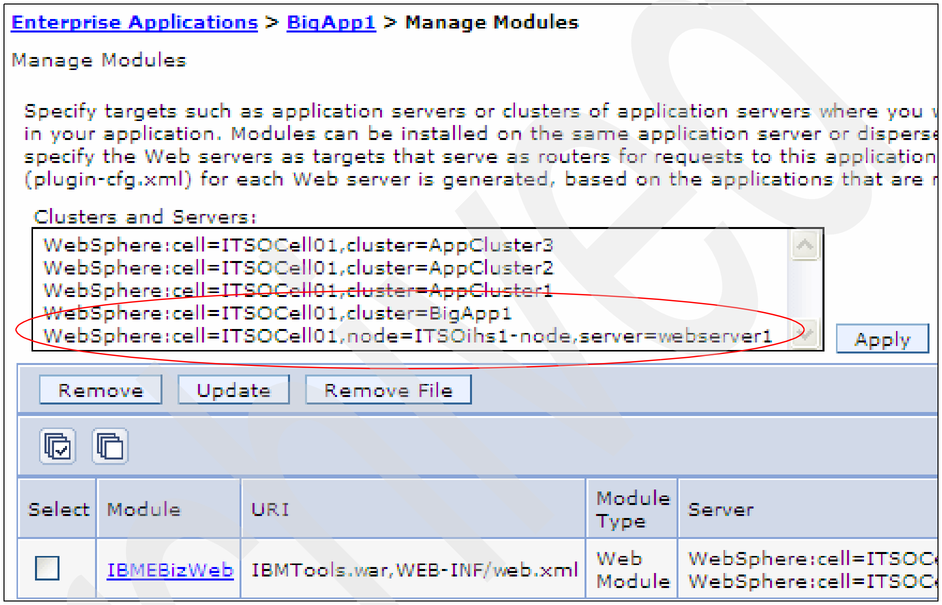


Figure 1-36 Mapping the BigApp1 application to the new BigApp1 dynamic cluster

When this change is saved to the WebSphere configuration, the following happens:

- The BigApp1 application is stopped in the servers in which it is deployed.
- The BigApp1 application is uninstalled from those servers.
- Any requests for the BigApp1 now fail.

The reason a request fails is because there are no servers in the dynamic cluster started. However, because we configured the dynamic cluster in the supervised mode, WebSphere Extended Deployment detects that there are no running servers to process requests and to raise a runtime task alert.

The changes were saved and matters proceeded as predicted. Selecting **System administration** → **Task Management** → **Runtime Tasks** showed that an alert was raised as expected (Figure 1-37).

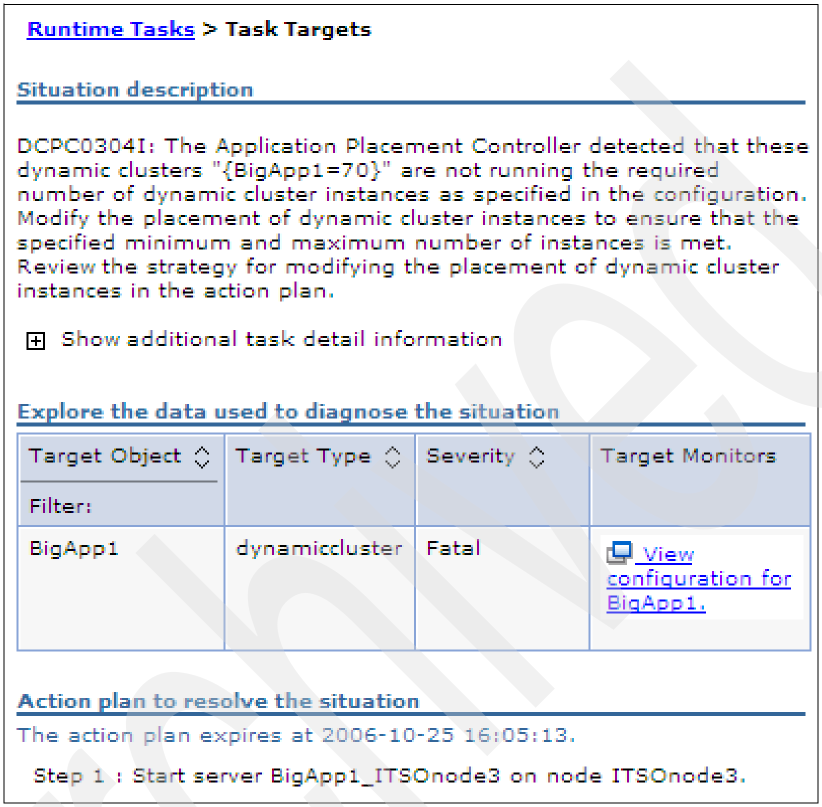


Figure 1-37 Runtime task advising to start a server

The task recommendation was accepted, a server was started, and the BigApp1 application became available.

Enable automatic mode

Having seen that WebSphere Extended Deployment did detect that there was an issue with the application availability of BigApp1 and recommended the correct action to take, we decided to change the dynamic cluster to work in automatic mode. In this mode WebSphere Extended Deployment automatically takes action in response to changes in workload.

Repeat procedure for BigApp2

We repeated the same process for the BigApp2 application so that it would run in a dynamic cluster. This time, the dynamic cluster mode was initially set to automatic mode.

1.3.15 The small applications

As the workload for the small applications was small compared to that of the big applications, we decided to leave the small applications as is for now, rather then splitting them out into separate dynamic clusters.

1.3.16 Configuration with dynamic operations implemented

Figure 1-38 shows the WebSphere environment after the dynamic operations were implemented. Note that the deployment manager is not shown.

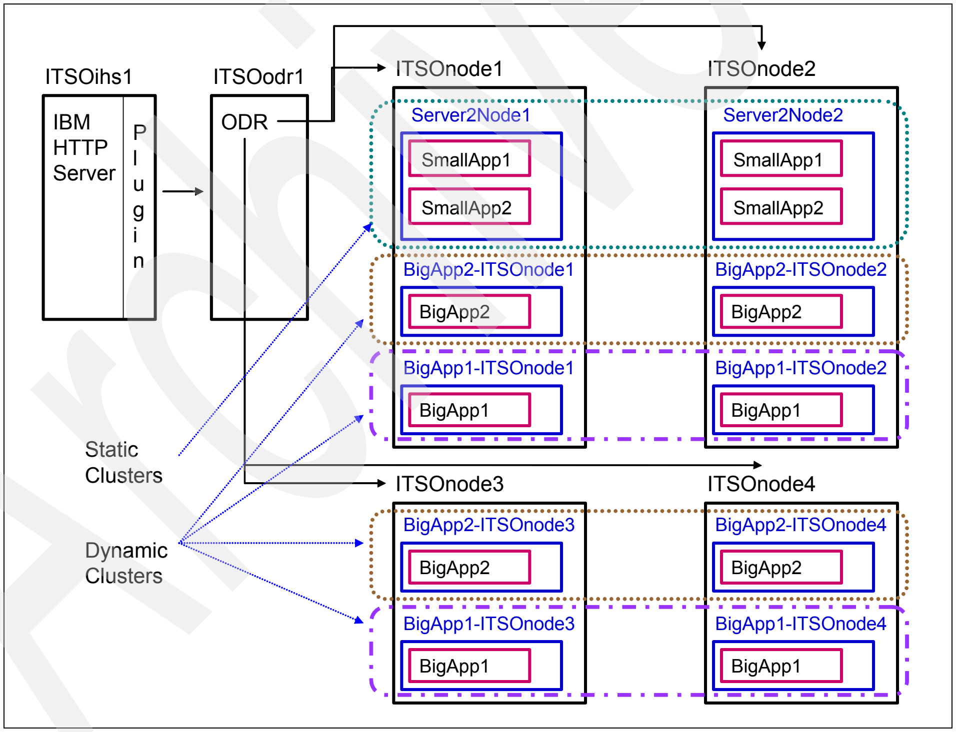


Figure 1-38 Configuration after dynamic operations implemented

1.3.17 Handling an unexpected business requirement

With dynamic operations implemented, the WebSphere technical team was able to use response time graphs to show the business manager that the response times for the business applications were meeting the agreed service-level goals. The BigApp1 application was now getting better throughput and response time since it was now running, when necessary, on all four nodes. Complaints about poor response time stopped.

The business manager then happened to mention that she was planning to give a demonstration of a specific part of the BigApp2 application to senior management the next day. She was concerned that the demonstration might not go smoothly since that was the same day as end-of-month processing, which produced a high workload.

We decided to use dynamic operations to prioritize the handling of her requests to help maximize the success of her demonstration.

Plan approach

The BigApp2 application was currently mapped to the Silver service policy, which has a response goal of 50 msec. One option was to just associate the required subset of the BigApp2 application with the Gold service policy. However, it was not known if that part of the BigApp2 application could deliver response times that met the Gold service policy goal of 20 msec. Note that mapping requests to a service policy with a lower response time goal does not make the application run any faster inside the application server.

We decided to define a SilverPlus service policy, with the same goal as the existing Silver service policy, but with a higher importance than the Gold service policy. We used a new transaction class and work class so that the targeted requests were mapped to the SilverPlus service policy.

Following is the approach we decided on:

- Define a new service policy called SilverPlus.
- Define a new transaction class called BigApp2-TC2 and associate it with the SilverPlus service policy.
- Define a new work class for the specific part of BigApp2 that would be part of the demonstration.
- Associate the new work class with the new transaction class.

Defining the SilverPlus service policy

A new service policy called SilverPlus was defined with an importance setting of *Very High*. A new transaction class called BigApp2-TC2 was defined in the SilverPlus service policy. The results are shown in Figure 1-39.

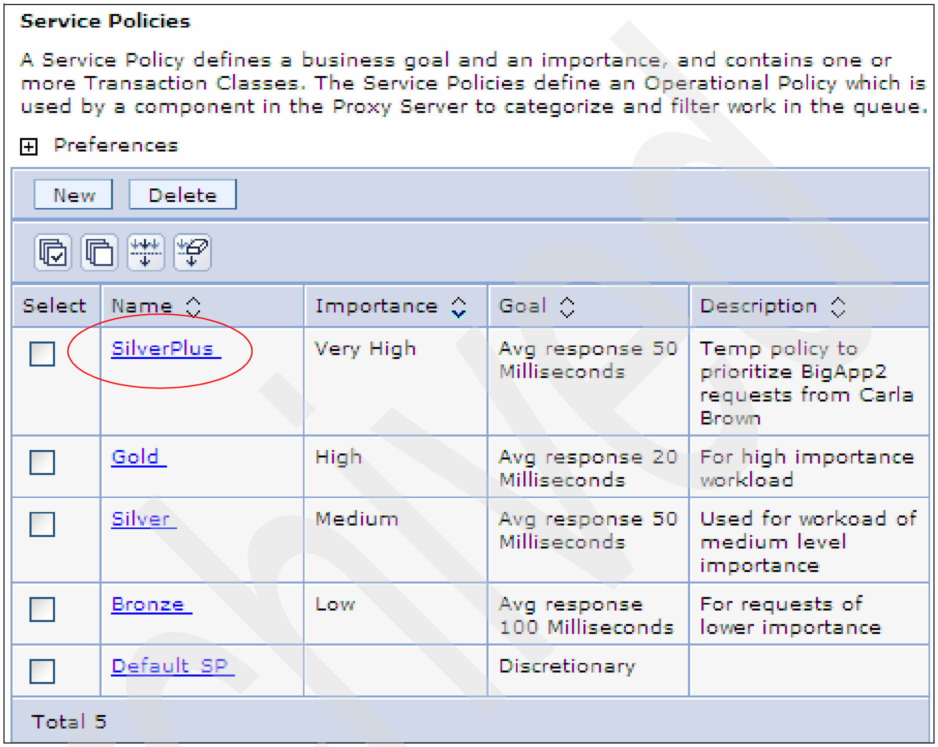


Figure 1-39 SilverPlus service policy defined

Define BigApp2-WC1 work class

A new work class called BigApp2-WC1 was defined. This work class only applied to a specific URL. Using the rule capability, a condition was added that this work class would only apply for requests sent from a specific TCP/IP address (the machine to be used in the demonstration).

The new work class was defined by going to the list of service policies for the BigApp2 application, expanding the Work Classes for HTTP Requests section and clicking **New**. Figure 1-40 on page 57 shows how the specific URL request was selected.

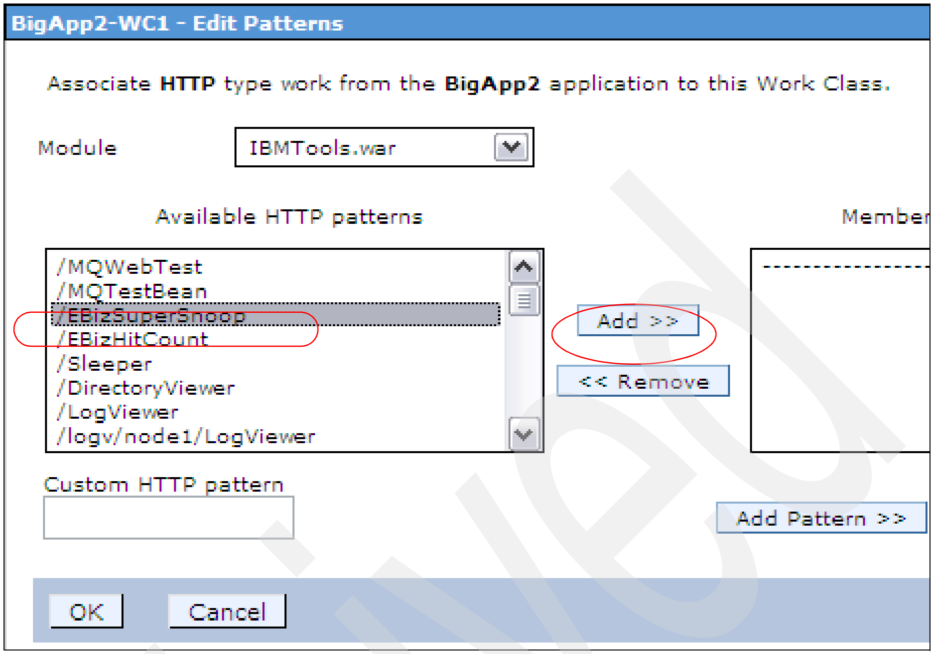


Figure 1-40 Selecting the specific HTTP pattern

A rule was added to the new work class by clicking the **Add Rule** button and then clicking the **Rule Builder** button, which produced the display shown in Figure 1-41.

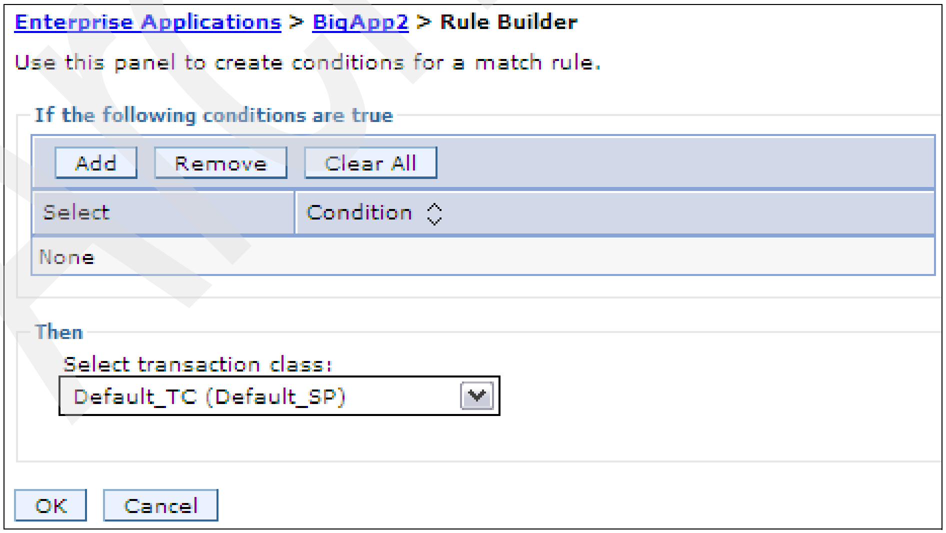


Figure 1-41 Initial rule builder panel

The **Add** button was clicked and the information shown in Figure 1-42 was used to create the condition for the rule.

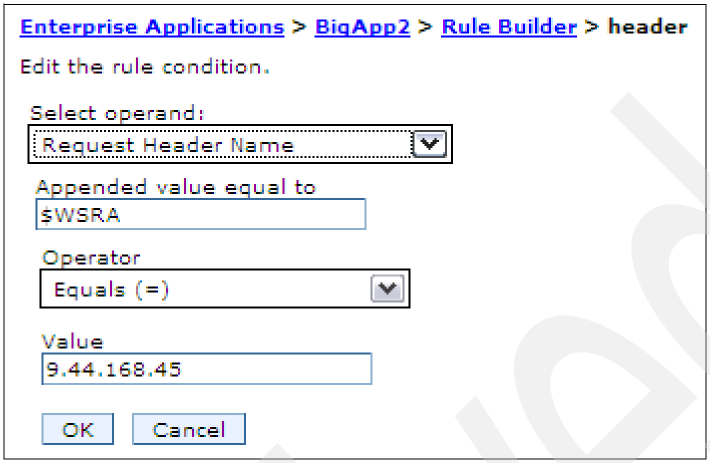


Figure 1-42 Defining the rule for a specific TCP/IP address

The condition specifies that the rule will match if it finds the $WSRA HTTP Header, which is added by the Web server plug-in, with a value of 9.44.168.45.

Clicking **OK** takes you back to the previous display, where the BigApp2-TC2 transaction class was selected (Figure 1-43).

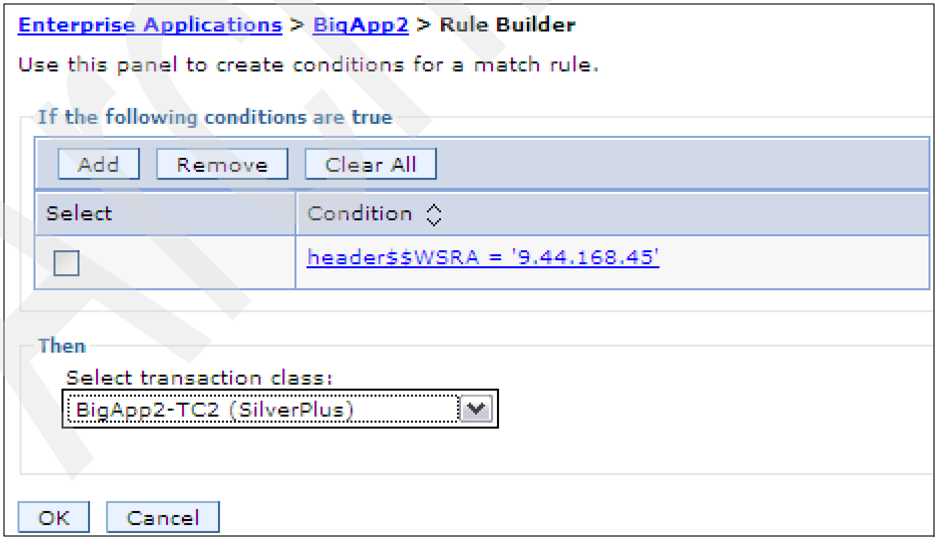


Figure 1-43 Setting the transaction class for the new work class

We clicked **OK** and saved the change.

A test was done prior to the meeting and the Runtime Topology view was used to verify that the requests were being managed under the new SilverPlus service policy as shown in Figure 1-44.

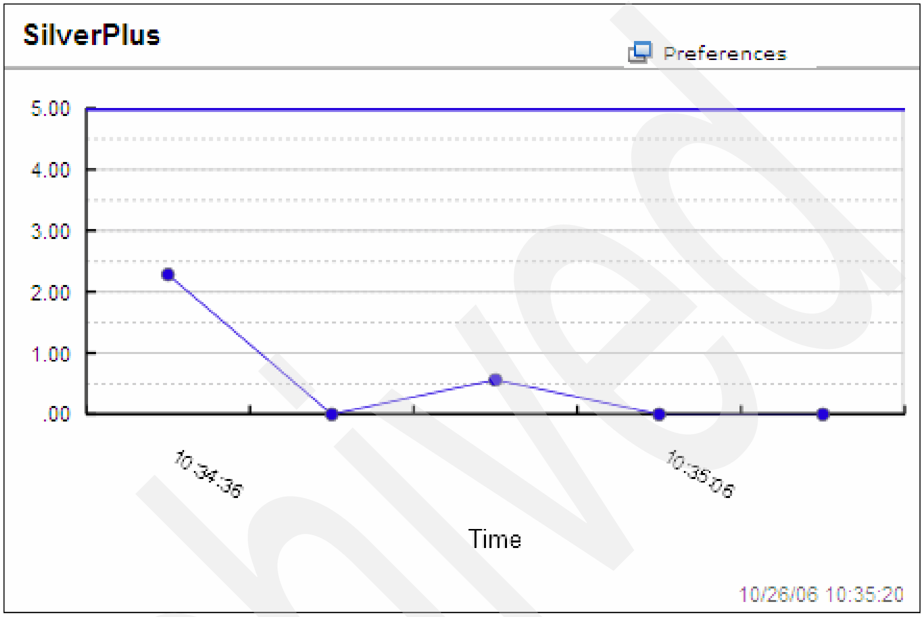


Figure 1-44 Showing activity in the SilverPlus service policy

1.3.18 Summary

The ITSO-Area-305 company has now implemented WebSphere Extended Deployment into an existing Network Deployment installation. The two key applications of the business are now able to use the available capacity of all nodes in the environment. The WebSphere technical team can use the Runtime Topology view to see how the applications are performing against the service levels that were defined. Additionally they can show this information to the business area manager in an easy-to-understand format. WebSphere Extended Deployment was also used to meet an unexpected business requirement by categorizing parts of an application into a separate work class and a new service policy.

2

Application hosting and chargeback

This chapter focuses on the centralized hosting of applications on behalf of multiple business units. These business units may be external or internal to your organization. In addition to agreeing to SLAs between yourself and these business units, you will typically want to recover the costs of your operation from them. We show how you can calculate recharges from the WebSphere Extended Deployment's visualization logs.

This chapter contains the following topics:

- Application hosting overview
- Planning and organizational aspects of application hosting
- Implementing chargeback
- Integration with IBM Tivoli Usage and Accounting Manager (ITUAM)

> **Note:** An Interim fix for WebSphere Extended Deployment V6.0.2 is in development that will add the ability to format time stamps through a custom property and will add two new log files, FineGrainedPowerConsumptionStatsCache and ServerPowerConsumptionStatsCache. This chapter was written using an early copy of this fix.

2.1 Application hosting overview

In this chapter we look at the following:

- Organizational considerations with regard to application hosting.
- How the visualization logs in Extended Deployment can be used as a basis for chargeback.
- Integration of these logs into IBM Tivoli Usage and Accounting Manager (ITUAM) as part of a comprehensive chargeback solution.

In Chapter 1, “Service level optimization” on page 3 you saw how WebSphere Extended Deployment could be used to optimize operational efficiency for a single business unit. Here we build on that capability and show how Extended Deployment can deliver a shared infrastructure to satisfy the needs of multiple business units. Political issues as well as technical ones are considered.

In this scenario a single application hosting service supports multiple independent applications. Each has a different virtual host and each has their own SLA for response time and availability. These applications could be functionally different, or they could be instances of the same application, but they service different groups of end users. These groups could be business units within your organization, or they could be external companies to whom you are providing a service.

Reasons to use WebSphere Extended Deployment for application hosting and chargeback:

- You need to provide application hosting to external businesses from a single IT infrastructure.
- You run a central IT function that provides applications to the businesses within your organization.
- You need to understand the costs associated with different applications and transactions.
- You need to charge for the services you provide based on usage.

2.2 Planning and organizational aspects of application hosting

In this section we describe how to plan an application hosting infrastructure to meet the varying, and possibly competing, needs of different business units.

We will look at planning for the following:

- A shared infrastructure that adequately isolates the workload of the different business units.
- A monitoring and chargeback structure.

2.2.1 Gathering information and negotiating

Before you start planning the technical implementation of your shared WebSphere infrastructure you need to do the following:

- Understand your business units and their applications
- Understand your infrastructure
- Gain acceptance of your shared infrastructures
- Batch work
- Consider final design

Understand your business units and their applications

Start by analyzing the business units you are servicing and their applications. Ensure that you understand their business priorities and how these relate to their applications. For each business unit, list all of its applications and document the business priorities associated with each. These could include the following:

- Minimize cost
- Ensure highest possible availability
- Rapidly roll out changes to applications according to business needs
- Deliver consistent response times

Typically, these vary from business unit to business unit and from application to application.

Understand your infrastructure

Document the scope of the existing systems to understand the current infrastructure and the extent to which it is capable of meeting business priorities.

If you are going to implement chargeback, you need to know how much cost you need to cover. This is likely to be the total cost of running your installation plus a contingency or required profit.

Gain acceptance for your shared infrastructure

If applications are running on dedicated hardware, there may be resistance to moving applications to a shared infrastructure. You can counter these objections by explaining the benefits to be obtained from a shared infrastructure based on Extended Deployment. These include the following:

- Reduced total cost of ownership. Not only is less hardware required but also fewer software licenses are required and administrative effort is reduced.
- Ability to set and meet SLAs.
- Capability to set up a recharging structure such that businesses pay only for resources used or the service delivered. Business units are therefore not required to pay for resources they do not need.
- Ability to accommodate large fluctuations in demand without having to over-provision. Extended Deployment can dynamically redirect available resources towards supporting the most important transactions.
- Support for rarely used applications. Extended Deployment can easily support applications that are run only occasionally, perhaps once per year.
- Ability to manage several versions of an application simultaneously and to non-disruptively roll out changes to these applications.

Try to understand the reasons for objections. Are your users concerned about loss of control? Plan to implement a level of usage reporting that gives your business units the information they need to have confidence in your planned infrastructure.

Agree on SLAs

Negotiate SLAs with your business units at the level of detail that they require and that you know that you can deliver for the appropriate combinations of user, application, and transaction.

> **Best Practice:** Avoid over-commitment. The dynamic operations capability of Extended Deployment cannot make your applications perform better than they can standalone.

Agree on chargeback

Consider whether chargeback is based on IT usage metrics or business value metrics:

- IT usage metrics reflect the cost of providing an IT infrastructure.
- Business value metrics are based on measures of what that IT infrastructure delivers to the business.

Examples of IT usage metrics are CPU and memory utilization. An example of a business value metric could be the numbers of balance inquiry transactions completed within the target response time.

Best practice: Measure and report both business value and IT usage metrics, but base chargeback on only one of them.

If you charge based on IT usage, you can use business value metrics to demonstrate the business value of IT expenditure. Conversely, if you adopt business value metrics as your basis for chargeback, you should still measure usage metrics so you can reconcile chargeback against IT costs.

Best practices:

- Keep your metrics simple. That way you and your business users can understand them.
- Favor business value metrics over IT usage metrics. Your end users will prefer them. Also it gives IT an incentive to become more efficient.
- Ensure that your charge out policy is compatible with the accounting standards and procedures for your business.
- Calculate your charge out rates based on expected usage patterns. (Build a spreadsheet model or similar.)
- Negotiate with your end users to agree the basis of re-charge rather than try to dictate. But try to keep it simple, and make sure you can implement anything you agree.
- Perform risk analysis based on possible usage patterns.
- Perform volume tests to stress test your system, and ensure that you get expected charge out values.
- Be on the lookout for unintended consequences. Implementing chargeback could influence user behavior. Users may find ways of reducing their costs while increasing yours!
- If you can, run for a trial period generating statistics and reports before using them as a way of automatically generating chargeback.

2.2.2 Designing the Extended Deployment configuration

Approaches for transactional and batch work are slightly different.

Transactional work

Use as few node groups as possible to maximize sharing and flexibility.

Dynamic clusters need to be sufficiently granular to accomplish the following:

- Allow Extended Deployment to optimize resources towards meeting SLAs
- Isolate business units from each other
- Provide logging data for chargeback

Best practice: Create a dynamic cluster for each business unit for each transactional application.

In some circumstances, where fine-grained SLAs are not important and the emphasis is on cost reduction through resource sharing, you could opt to run several applications for an individual business unit within one dynamic cluster.

Batch work

The WebSphere Extended Deployment Information Center recommends having each long running application in its own dynamic cluster. For application hosting this implies that a separate dynamic cluster is required for each business unit for each application. This could be excessive so unless you require very fine grained optimization of your batch work, it is generally more convenient to have a dynamic cluster for each business unit. Each dynamic cluster runs the long-running execution environment (LREE) and all the long-running applications for a business unit.

Best practice: For batch work create a dynamic cluster for each business unit.

Important: It is poor practice to use the same dynamic cluster for both batch and transactional work. To understand why read the following article:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/topic/com.ibm.websphere.xd.doc/info/scheduler/cxdbatchint.html

If you have relatively little batch work, it may be convenient to run it all within one dynamic cluster; however, it may be difficult to separate out usage statistics for your business units.

Final design consideration

It is likely that you will end up with many dynamic clusters. Some of these, particularly batch ones, are not in use at all times. You should consider setting minimum instances to zero in the cluster definition so that they do not consume resources when not in use. If you decide to use this technique for your transactional work, you should configure lazy application start. This is described in both the IBM Redbook *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153, and in the WebSphere Extended Deployment Information Center.

2.2.3 Planning for chargeback

Even if you are not required to chargeback to the business units, you should set up a system for monitoring usage. You will need it for capacity planning.

A shared virtualized environment makes the measurement and allocation of IT costs challenging. WebSphere Extended Deployment addresses this challenge by recording resource utilization and performance statistics in considerable detail. In particular, it uses advanced measurement techniques and algorithms to derive an accurate measure of CPU consumption. These statistics can be used to report usage in both IT and business value terms. They can be recorded for both transactional and batch work.

This data can be further processed by ITUAM to report resource consumption at company, department, or individual level and to bill users.

In this section we look at facilities in Extended Deployment for the creation of chargeback statistics.

Calculating compute power consumed

Extended Deployment computes a standardized power statistic for every node in the environment based on the number, speed, and architecture of the processors found on the host machine. This statistic, called the *node power*, is in MHtz. It can be multiplied by some time interval to determine for that interval how much work could be completed on a node - the *node work potential*.

Extended Deployment also exposes, per server, a CPU utilization metric based on operating system measurements. This statistic, called the *% CPU Utilization*, is then multiplied by the node work potential to determine how much work, over the interval, was actually completed by that server. This metric is called *work completed* and is in Mcycles.

A *transaction class module* (TCM) is an application module executing under a transaction class (middleware application, module, transaction class).

A *work factor* (Mcycles/req) is the amount of work consumed per request by a TCM. These are calculated by the work profiler.

The *work profiler* is a very sophisticated component of Extended Deployment. The following is a simplified account of how it calculates work factors.

- Each request that comes through the ODR gets classified to a specific TCM. Over the measurement time interval the number of requests serviced by the server for each TCM are counted.
- The work profiler then computes work factors for each TCM such that the sum over all TCMs that had requests serviced on that server of—work factor multiplied by the number of requests serviced equals the total work completed by the server during the interval.

The work factor can then be used to calculate various aggregations. Following are some examples:

- The work completed for all requests for a particular TCM is calculated by multiplying the work factor by the number of requests serviced.
- The work completed for an application can then be computed by summing the work completed for all TCMs associated with that application.

Visualization data logging

Through its visualization features, Extended Deployment can log historical performance metrics. This facility can be used in production to produce statistics for both transactional and batch work.

The logs are simple comma-separated-value (CSV) text files, and can be processed with scripting languages such as Perl, or they can be imported into a spreadsheet, data warehouse, or reporting tool. In particular, they can be imported into ITUAM for charge back accounting.

WebSphere Extended Deployment logs to the following files:

- BusinessGridStatsCache
- DeploymentTargetStatsHistoricCache
- NodeStatsHistoricCache
- ServerStatsCache
- TCModuleInstanceStatsCache
- TCModuleStatsCache
- TierStatsCache
- ServerPowerConsumptionStatsCache
- FineGrainedPowerConsumptionStatsCache

The contents of these files are described in detail in Appendix A, “Appendix - Visualization logs” on page 339, where we look at how you can use the data from these files.

BusinessGridStatsCache

This log provides information about batch work. Data recorded includes the following:

- Counts of jobs run, both successfully and unsuccessfully.
- Average execution times.

It creates a record for each application and deployment target combination. Although it is populated with data from both transactional and batch work, it is really only of value for batch work.

DeploymentTargetStatsHistoricCache

This file contains limited information. It is documented in "DeploymentTargetStatsHistoricCache" on page 342. It is not discussed further in this chapter.

NodeStatsHistoricCache

This log contains historic information from the node statistics cache. Data recorded includes the following:

- CPU utilization
- Memory usage

It records data for both transactional and batch processing. At the specified time interval it creates a record for the deployment manager and every node agent in the cell.

ServerStatsCache

This log file contains data from the server statistics cache. Data recorded includes the following:

- CPU utilization
- Memory usage
- Number of requests
- Average database response time
- Database throughput

It records data for both transactional and batch processing. At the specified time interval it creates a record for every server in the cell including stopped servers and Web servers. (The deployment manager and node agents are reported in the NodeStatsHistoricCache log).

TCModuleStatsCache

This file contains detailed performance information gathered at the level of TCM (transaction class application module) for clusters. Data recorded includes the following:

- Number of dispatched requests
- Number of requests dropped
- Response time
- Number of requests serviced
- Number of requests with response times above their service class threshold
- Work factor (see "Calculating compute power consumed" on page 67)

This log is produced only for transactional work.

TCModuleInstanceStatsCache

This file contains detailed performance information gathered at the level of TCM for servers. Data recorded includes the following:

- Number of dispatched requests
- Response time
- Number of requests serviced
- Number of requests with response times above their service class threshold
- Work factor (see "Calculating compute power consumed" on page 67)

This log is produced only for transactional work.

TierStatsCache

During our testing no data was ever observed in the log for this cache.

FineGrainedPowerConsumptionStatsCache

This log file contains fine grained power and work consumption data. Data recorded includes the following:

- Work factor
- Number of requests serviced
- Work completed
- Power consumed

This log is produced only for transactional work. A record is written for every TCM/server instance. This gives a record for every middleware application, module, transaction class, and server instance that has had work routed through an ODR. There are additional fields that hold relationship information such as the cluster to which the server belongs, the node group with which the cluster is associated, and the service policy with which the transaction class is associated.

ServerPowerConsumptionStatsCache

This file records power and work consumption for each server. It is a consolidation of FineGrainedPowerConsumptionStatsCache at the server level. Data recorded includes the following:

- CPU utilization
- Work completed
- Power consumed

It records data for both transactional and batch processing.

Visualization metrics summary table

Table 2-1 on page 72 provides a cross reference between metrics and log files.

Table 2-1 Cross reference of metrics to logs

Processing	Metrics	Logs
transactional	CPU utilization	► NodeStatsHistoricCache ► ServerStatsCache ► ServerPowerConsumptionStatsCache
	Memory usage	► NodeStatsHistoricCache ► ServerStatsCache
	Number of requests	► ServerStatsCache ► TCModuleStatsCache ► TCModuleInstanceStatsCache ► FineGrainedPowerConsumptionStatsCache
	Database response time and throughput	► ServerStatsCache
	Detailed request data	► TCModuleStatsCache ► TCModuleInstanceStatsCache
	Work factor	► TCModuleStatsCache ► TCModuleInstanceStatsCache ► FineGrainedPowerConsumptionStatsCache
	Number of requests exceeding response time threshold for service class	► TCModuleStatsCache ► TCModuleInstanceStatsCache
	Work completed and power consumed	► FineGrainedPowerConsumptionStatsCache ► ServerPowerConsumptionStatsCache
Batch	Count of jobs run and execution times	► BusinessGridStatsCache
	CPU utilization	► NodeStatsHistoricCache ► ServerStatsCache ► ServerPowerConsumptionStatsCache
	Memory usage	► NodeStatsHistoricCache ► ServerStatsCache

Processing and reporting

How you process these logs depends on your particular circumstances. Options that you should consider include the following:

- Import the logs into a relational database such as DB2® and process using SQL.
- Consolidate the logs using a language such as Perl and then import them to a spreadsheet or to a data warehouse.
- Import the logs to a specialized usage and accounting application such as ITUAM.

2.3 Implementing chargeback

Here we examine in detail the use of the visualization logs for chargeback.

2.3.1 Visualization logging

Visualization logs are created on the deployment manager in the visualization directory. This is typically the following:

```
<WAS_install_root>\profiles\Dmgr01\logs\visualization
```

Data logging is conveniently initiated from the administrative console by navigating to the visualization data service settings at **System Administration** → **Visualization Data Service**, as shown in Figure 2-1 on page 74.

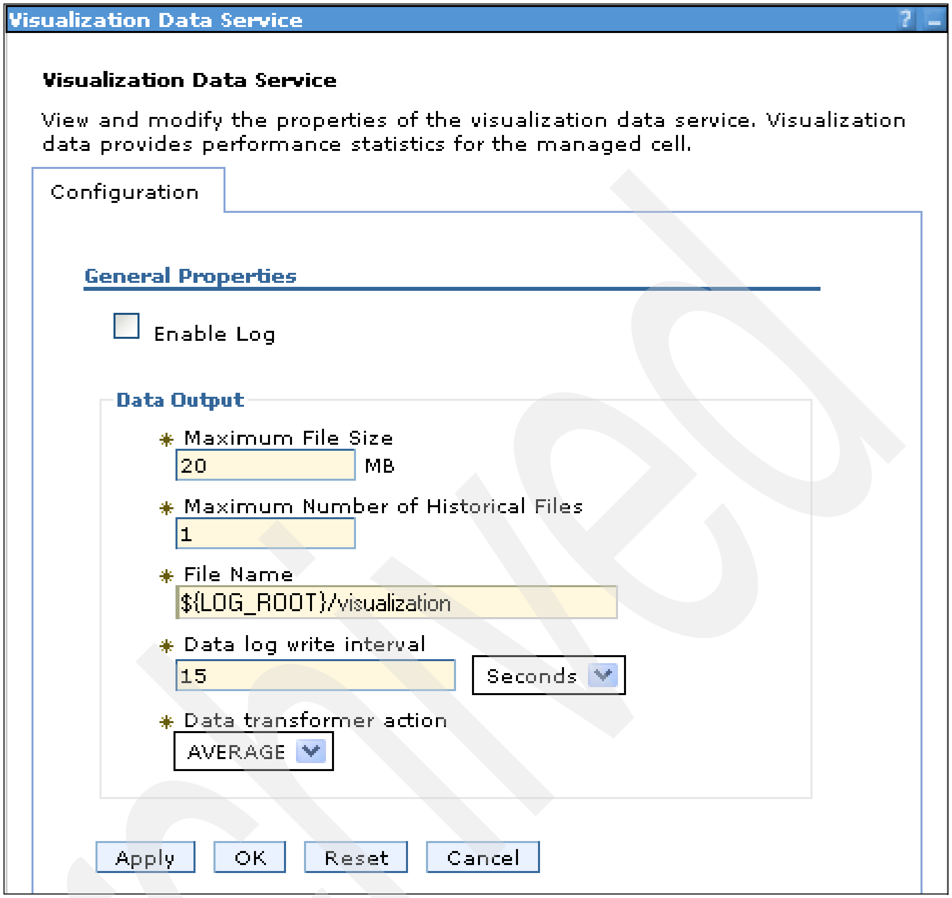


Figure 2-1 Visualization data service

Depending on how you intend to manage the log files, you probably want to change the Maximum File Size and Maximum Number of Historical Files parameters. After a log file reaches the maximum file size setting, its name is extended with a time stamp and a new file is started. You can also change the frequency of capture and thereby the amount of data logged.

When logging is enabled, WebSphere Extended Deployment always logs to the following files:

- BusinessGridStatsCache (However, this file only contains useful data if the long-running runtime environment is in use.)
- NodeStatsHistoricCache
- ServerStatsCache

- DeploymentTargetStatsHistoricCache
- ServerPowerConsumptionStatsCache

When transactional work is processed, data is written, in addition, to these files:

- TCModuleInstanceStatsCache
- TCModuleStatsCache
- FineGrainedPowerConsumptionStatsCache

Note: Because these are CSV files, they can be conveniently viewed with Excel®. Simply change the file extension to .csv and double-click.

2.3.2 Formatting the time stamp

The time stamp used in these files is the time in milliseconds since January 1, 1970, 00:00:00 GMT. If you would like something more readable, you can cause the time stamp to be formatted by means of a custom property for the cell. This is a name value pair:

name:xd.visengine.timestampformat
value:date and time format

The SimpleDateFormat class is used to format timestamps. Documentation on date time patterns can be obtained from its javadoc:

http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html

Essentially you can use formats like the following:

```
MM.dd.yyy
hh:mm:ss:SSS
```

If you want to separate the time stamp into two or more fields you can do so with a comma:

```
MM.dd.yyy, hh:mm:ss:SSS
yyyy.MMMMM.dd, hh:mm:ss
```

You can create the custom property from the administrative console by going to **System Administration** → **Cell** → **Custom Property.**

The end result is similar to Figure 2-2 on page 76.

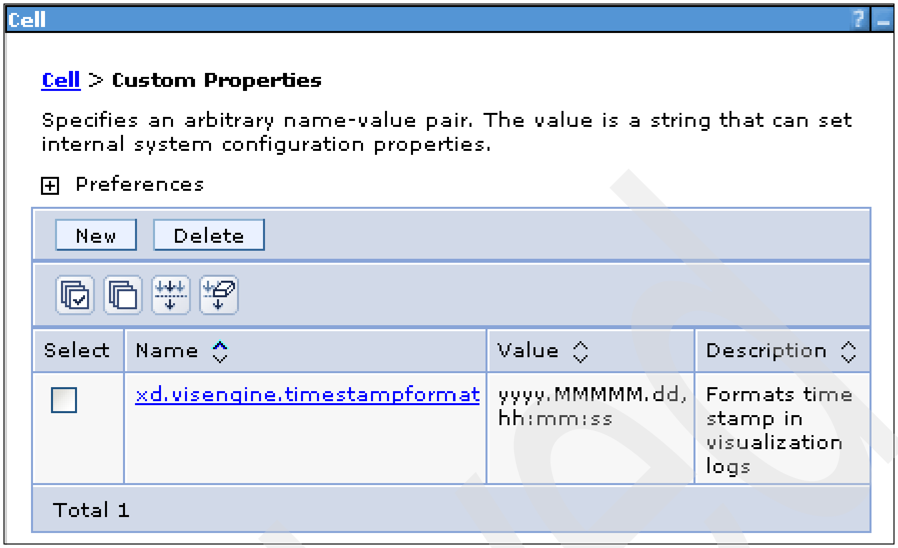


Figure 2-2 Custom property for time stamp format

> **Note:** If the time stamp is changed while data is being logged, the change takes place immediately. No new header is written until a new log file is created.

> **Important:** If you are going to process your files through ITUAM (see 2.4, "Integration with ITUAM" on page 87) you need to restrict your date formats to two-digit days, two-digit or three-character months, and 2 or four-digit years. For time, use 2 digits for hours, minutes, and seconds.

2.3.3 Log files from transaction processing

In this section, we look at log files created by Extended Deployment as a result of transaction processing, and how these files are used for obtaining data for chargeback.

For usage-based chargeback, the most useful log is likely to be the FineGrainedPowerConsumptionStatsCache, while for business value it is TCModuleStatsCache.

For completeness, all the relevant files are described.

FineGrainedPowerConsumptionStatsCache

This log file contains fine-grained power and work consumption data. A record is written for every TCM/server instance. Figure 2-3 shows a sample record from FineGrainedPowerConsumptionStatsCache after import into Excel and transformation to column format.

timeStamp	2006.November.09
timeStamp_2	01:56:40
tcmodname	BigApp2-TC1:!:BigApp2:!:IBMTools.war
gwid	:BigApp2:ITSOCell01_ITSOodr1Node01_odr1
cell	ITSOCell01
appname	BigApp2
j2eemodname	IBMTools.war
servicepolicy	Silver
transactionclass	BigApp2-TC1
server	BigApp2_ITSOnode4
node	ITSOnode4
odr	ITSOCell01_ITSOodr1Node01_odr1
cluster	BigApp2
nodegroup	ITSO-Area-305
begintm	2006.November.09
begintm_2	01:56:25
endtm	2006.November.09
endtm_2	01:56:40
workfactor	526.5955
numserviced	6
workcompleted	3159.5732
powerconsumed	210.63821
nodepower	3059
nodeworkpotential	45885
cellpower	19437
cellworkpotential	291555

Figure 2-3 Sample record from FineGrainedPowerConsumptionStatsCache

Notice how the values for time stamp, begintm and endtm, were split into date and time fields and formatted according to our custom property.

Refer to “Calculating compute power consumed” on page 67, and “FineGrainedPowerConsumptionStatsCache” on page 349 for information about the various fields in this record.

Using this record you can determine the following:

- Location:

  The IBMTools.war module of application BigApp2 was executing under transaction class BigApp2-TC1 on server BigApp2_ITSONode4 in dynamic cluster BigApp2.

- Work completed:

  The workfactor was 526.5955 Mcycles/request. Six requests were serviced (numserviced) so the following applies:

  ```
  workcompleted=6 x workfactor
  ```

- Power consumed node work potential:

  The nodepower was 3059 MHtz (3059 Mcycles/sec). The write interval was 15 sec so the following applies:

  ```
  powerconsumed=workcompleted/15
  nodeworkpotential=15 x nodepower
  ```

- CPU utilization

  The CPU utilization by this TCM on this node can be calculated as either:

  ```
  workcompleted / nodeworkpotential
  ```

  or

  ```
  powerconsumed / nodepower
  ```

  Similarly, you can obtain the utilization over the cell.

It can be clearly seen that this file can be used to calculate CPU consumed by an application for a business unit over a period of time. It can therefore be used as a basis for chargeback based on CPU.

ServerPowerConsumptionStatsCache

This file is a consolidation of the FineGrainedPowerConsumptionStatsCache at the server level with some additional server data. Figure 2-4 on page 79 contains a sample record from the ServerPowerConsumptionStatsCache log.

timeStamp	2006.11.16
timeStamp_2	04:11:01
cell	ITSOCell01
name	BigApp2_ITSOnode4
node	ITSOnode4
cluster	BigApp2
nodegroup	ITSO-Area-305
begintm	2006.11.16
begintm_2	04:10:50
endtm	2006.11.16
endtm_2	04:11:05
cpu	0.3125
workcompleted	143.39062
powerconsumed	9.559375
nodepower	3059
nodeworkpotential	45885
cellpower	19437
cellworkpotential	291555

Figure 2-4 Sample record from the ServerPowerConsumptionStatsCache

This file records work and power consumed at a server.

Although this record was collected some days after the one for the FineGrainedPowerConsumptionStatsCache, the values recorded for nodepower: nodeworkpotential, cellpower, and cellworkpotential are identical to the ones recorded earlier.

The CPU field contains the average % CPU consumed by this server over the time interval. The other two values are calculated as the following:

```
workcompleted = (cpu/100) x nodepower x time interval in seconds
powerconsumed = nodepower x cpu / 100.
```

This data can be aggregated to dynamic cluster level and used for chargeback when the detail contained in the FineGrainedPowerConsumptionStatsCache is not required. This has the advantage that less data needs to be processed.

> **Note:** Due to the asynchronous nature of data capture, endtm does not always line up with time stamp.

NodeStatsHistoricCache and the ServerStatsCache

These two files contain useful memory usage information. They can also be used to calculate CPU utilization.

> **Note:** With the availability of the FineGrainedPowerConsumptionStatsCache and the ServerPowerConsumptionStatsCache, the use of these files and technique described here have become obsolete for the calculation of CPU utilization. It is included because it could be adapted to generate memory utilization data.

The Perl script in "Perl script to calculate CPU utilization" on page 352 calculates CPU utilization for each cluster in a cell relative to the total utilization of the cell using the following method.

The NodeStatsHistoricCache and ServerStatsCache records are matched on timeStamp and node. You can then calculate the amount of resource consumed over time by the applications running in a cluster as follows:

1. Over all nodes, add nodeSpeed*CPU to obtain a measure of the total compute power used.
2. From the SeverStatsCache, records parse the name (using the underscore between server name and cluster name) to obtain the cluster name.
3. For each server in a cluster, multiply CPU utilization by node speed to get CPU used, and add these to get the compute power consumed by the cluster.
4. Divide the compute power consumed by the cluster by the total compute power used to obtain the CPU utilization by a cluster.

To illustrate this technique we took sample log files from our system, processed them through the Perl scripts, and input them into Excel.

> **Note:** The Perl script depends on the time stamp being a single field, so this will not work if you format the time stamp as two fields, as shown in 2.3.2, "Formatting the time stamp" on page 75.

A portion of the ServersStatsCache log, after time stamp conversion, is shown in Figure 2-5 on page 81. Similarly, the NodeStatsHistoricCache log is shown in Figure 2-6 on page 81. After processing, the combined logs become the DCStatsCache log shown in Figure 2-7 on page 82.

ServerStatsCache.log.csv

A	B	C	D	E	F
timeStamp	name	node	version	weight	cpu
Thu Nov 2 16:45:59 2006	BUE_BigApp1_ITSOnode3	ITSOnode3	XD 6.0.2.0	0	
Thu Nov 2 16:45:59 2006	BUE_BigApp1_ITSOnode1	ITSOnode1	XD 6.0.2.0	0	
Thu Nov 2 16:45:59 2006	BUE_BigApp2_ITSOnode2	ITSOnode2	XD 6.0.2.0	0	
Thu Nov 2 16:45:59 2006	BUE_BigApp2_ITSOnode4	ITSOnode4	XD 6.0.2.0	0	
Thu Nov 2 16:45:59 2006	DBTest1-Cluster_ITSOnode2	ITSOnode2	XD 6.0.2.0	20	0.3
Thu Nov 2 16:45:59 2006	webserver1	ITSOihs1-node		0	
Thu Nov 2 16:45:59 2006	DBTest1-Cluster_ITSOnode4	ITSOnode4	XD 6.0.2.0	0	
Thu Nov 2 16:45:59 2006	BigApp1_ITSOnode3	ITSOnode3	XD 6.0.2.0	7	
Thu Nov 2 16:45:59 2006	BigApp1_ITSOnode1	ITSOnode1	XD 6.0.2.0	0	
Thu Nov 2 16:45:59 2006	BigApp2_ITSOnode2	ITSOnode2	XD 6.0.2.0	0	
Thu Nov 2 16:45:59 2006	BigApp2_ITSOnode4	ITSOnode4	XD 6.0.2.0	20	
Thu Nov 2 16:45:59 2006	area302-a	ITSOnode1	XD 6.0.2.0	1	9.8958
Thu Nov 2 16:45:59 2006	ITSO_health_ITSOnode4	ITSOnode4	XD 6.0.2.0	0	

Figure 2-5 Server StatsCache.log

np	nodeName	nodeCPU	nodeFreeMemory	usedMemory	versi	nod
2 16:45:58 2006	ITSOodr1Node01				XD 6	
2 16:45:58 2006	ITSOihs1-node					
2 16:45:58 2006	ITSOnode4	19	379292	166027	XD 6	
2 16:45:58 2006	ITSOnode3	100	285400	160779	XD 6	
2 16:45:58 2006	ITSOnode2	29	386468	153774	XD 6	
2 16:45:58 2006	ITSOnode1	2	198956	183733	XD 6	
2 16:45:58 2006	ITSOdmgrCellManager01					
2 16:46:15 2006	ITSOodr1Node01	15	472356	88817	XD 6	
2 16:46:15 2006	ITSOihs1-node					
2 16:46:15 2006	ITSOnode4	13	382136	164903	XD 6	
2 16:46:15 2006	ITSOnode3	100	277944	152175	XD 6	
2 16:46:15 2006	ITSOnode2	9	386368	154097	XD 6	
2 16:46:15 2006	ITSOnode1	1	199444	173038	XD 6	
2 16:46:15 2006	ITSOdmgrCellManager01					
2 16:46:30 2006	ITSOodr1Node01	3	472436	105038	XD 6	

Figure 2-6 NodeStatsHistoricCache

timestamp	name	numinstance	utilization
Thu Nov 2 16:46:16 2006	BUW	1	0.523243645
Thu Nov 2 16:46:16 2006	ITSO	1	43.9312938
Thu Nov 2 16:46:30 2006	BigApp1	2	0.936588154
Thu Nov 2 16:46:30 2006	DBTest1-Cluster	1	0.546785627
Thu Nov 2 16:46:30 2006	BUE	0	0
Thu Nov 2 16:46:30 2006	BigApp2	2	0.31255107
Thu Nov 2 16:46:30 2006	BUW	1	0.208150557
Thu Nov 2 16:46:30 2006	ITSO	1	46.35038439
Thu Nov 2 16:46:45 2006	BigApp1	2	0.104095455
Thu Nov 2 16:46:45 2006	DBTest1-Cluster	1	0.130187054
Thu Nov 2 16:46:45 2006	BUE	0	0
Thu Nov 2 16:46:45 2006	BigApp2	2	0.573010296
Thu Nov 2 16:46:45 2006	BUW	1	0.208150557
Thu Nov 2 16:46:45 2006	ITSO	1	45.77376818
Thu Nov 2 16:47:01 2006	BigApp1	2	0.468537924
Thu Nov 2 16:47:01 2006	DBTest1-Cluster	1	0.052074822

Figure 2-7 DCStatsCache

TCModuleStatsCache and the TCModuleInstanceStatsCache

The TCModuleStatsCache and TCModuleInstanceStatsCache contain detailed performance information gathered at the level of transaction class application module for deployment targets (typically dynamic clusters) and for servers, respectively.

In Figure 2-8 on page 83 you can see a sample record taken from a TCModuleStatsCache log and rendered in Excel. Notice that it was recorded at approximately the same time as the FineGrainedPowerConsumptionStatsCache shown in Figure 2-3 on page 77,

timeStamp_2	01:56:45
tcmodname	BigApp2-TC1:!:BigApp2:!:IBMTools.war
gwid	:BigApp2:ITSOCell01_ITSOodr1Node01_odr1
dtname	BigApp2
j2eemodname	IBMTools.war
appname	BigApp2
tcname	BigApp2-TC1
scname	Silver
nodegroup	
cell	ITSOCell01
proxy	ITSOCell01_ITSOodr1Node01_odr1
arrivals	7
executingInt	15
lengthInt	0
currentLen	0
departs	7
dropped	0
waittm	0
resptm	15
servicetm	15
serviced	7
begintm	2006.November.09
begintm_2	01:56:26
endtm	2006.November.09
endtm_2	01:56:46
qlen	0
abvgoal	0
workFactors	526.5955

Figure 2-8 Sample record from TCModuleStatsCache

Figure 2-9 on page 84 shows some of the metrics for the same record as it was written out every 15 seconds. You can see that all requests (arrivals) were serviced. By dividing total response time by the number of requests processed, you can obtain an average response time over each period. The abvgoal value of zero shows that all transactions had a response time within the service class threshold.

You can use this log to record numbers of transactions processed, and you can record the number that failed to complete and the number that exceeded their service class threshold.

arrivals	7	64	59	35
executingInt	15	108	32	47
lengthInt	0	0	0	0
currentLen	0	0	0	0
departs	7	64	59	35
dropped	0	0	0	0
waittm	0	0	0	0
resptm	15	108	32	47
servicetm	15	108	32	47
serviced	7	64	59	35

Figure 2-9 TCModuleStatsCache time sequence

Figure 2-10 on page 85 shows a sample record from the TCModuleInstanceStatsCache. These are recorded for each TCM on a server. This may be useful for capacity work, but in general are not very useful for chargeback.

dtname	BigApp2
scname	Silver
appname	BigApp2
tcname	BigApp2-TC1
server	BigApp2_ITSOnode4
node	ITSOnode4
nodegroup	
cell	ITSOCell01
proxy	ITSOCell01_ITSOodr1Node01_odr1
arrivals	6
currentLen	0
lengthInt	0
executingInt	31
departs	6
dropped	0
waittm	0
resptm	31
servicetm	31
serviced	6
begintm	2006.November.09
begintm_2	01:56:25
endtm	2006.November.09
endtm_2	01:56:40
qlen	0
abvgoal	0
workFactors	526.5955

Figure 2-10 Sample record from TCModuleInstanceStatsCache

2.3.4 Log files from long-running (batch) applications

Data from long-running applications is collected in the following:

- BusinessGridStatsCache
- NodeStatsHistoricCache
- ServerStatsCache
- ServerPowerConsumptionStatsCache

If you have followed best practice, then your batch workload will run in a separate dynamic cluster for each business unit, and you can calculate CPU usage from the ServerPowerConsumptionStatsCache. If you wish to report on memory usage then use the ServerStatsCache.

The BusinessGridStatsCache can be used to provide business value metrics. However, the NodeStatsHistoricCache is of limited use.

It is anticipated that WebSphere Extended Deployment V6.1 will generate per job usage data for each cell, node, server, user, job name, and job ID. The data will be stored in a relational database and can be off loaded to TUAM import format.

BusinessGridStatsCache

You can extract performance statistics from the BusinessGridStatsCache. Figure 2-11 shows a sample of BusinessGridStatsCache data that was imported into Excel.

timeStamp	Mon Nov 6 17:33:41 2006	Mon Nov 6 17:33:41 2006
node	ITSOlree1node	ITSOlree2node
server	LREE_DC_ITSOlree1node	LREE_DC_ITSOlree2node
tcname	Default_TC	Default_TC
appname	SimpleCIEar	SimpleCIEar
j2eemodname	SimpleCIEJB.jar	SimpleCIEJB.jar
version	XD 6.0.2.0	XD 6.0.2.0
dtname	LREE_DC	LREE_DC
scname	Default_SP	Default_SP
nodegroup	LREE_NG	LREE_NG
cell	ITSO	ITSO
updateTime	1.16283E+12	1.16283E+12
num_requested	9	8
num_completed	6	5
exec_time	40122	30050
max_concurrency	0	0
num_queued	6	5

Figure 2-11 Sample from BusinessGridStatsCache

You can see that the sample application, SimpleCI, was running on two nodes in the LREE_DC dynamic cluster. On one node, nine requests were made and six completed, while on the other, five out of eight were completed. Average execution time is also shown.

At the time that these records were recorded, time stamp formatting through a custom property (2.3.2, “Formatting the time stamp” on page 75) was not implemented, and so updateTime is in exponential form. Timestamp was formatted using a Perl script.

ServerPowerConsumptionStatsCache

Figure 2-12 contains a sample record from ServerPowerConsumptionStatsCache. By aggregating these across a dynamic cluster belonging to a business unit, the CPU usage, when processing batch work, can be derived.

timeStamp	2006.November.09
timeStamp_2	04:48:39
cell	ITSO
name	LREE_DC_ITSOlree1node
node	ITSOlree1node
cluster	LREE_DC
nodegroup	LREE_NG
begintm	2006.November.09
begintm_2	04:56:41
endtm	2006.November.09
endtm_2	04:56:56
cpu	0.3125
workcompleted	567741.4
powerconsumed	9.35
nodepower	2992
nodeworkpotential	1.82E+08
cellpower	15361
cellworkpotential	9.33E+08

Figure 2-12 Sample record from the ServerPowerConsumptionStatsCache

2.4 Integration with ITUAM

In this section we introduce the IBM Tivoli Usage and Accounting Manager (ITUAM) and describe how you can use it to generate reports from your visualization log files. For our example we took the FineGrainedPowerConsumptionStatsCache log, and calculated chargeback based on power consumed for an application.

After introducing ITUAM we describe the following:

- Setting up ITUAM including installation and configuration and the creation of an Account Code structure and a Rate Code.
- Processing of the FineGrainedPowerConsumptionStatsCache log.
- Generation of a report.

2.4.1 Introduction to ITUAM

ITUAM is a sophisticated, fully functional application for measuring and accounting for IT usage and implementing chargeback. It collects usage and accounting data from multiple sources and converts it to a common format for costing and reporting. It is uniquely capable of reporting on virtualized multi-platform environments.

Through the use of ITUAM's specialized data collectors and by other means, ITUAM accumulates data from a wide variety of sources for storage in its database. From this data, ITUAM can automatically generate invoices and usage reports at a variety of levels.

As shown in Figure 2-13, ITUAM consists of a number of components.

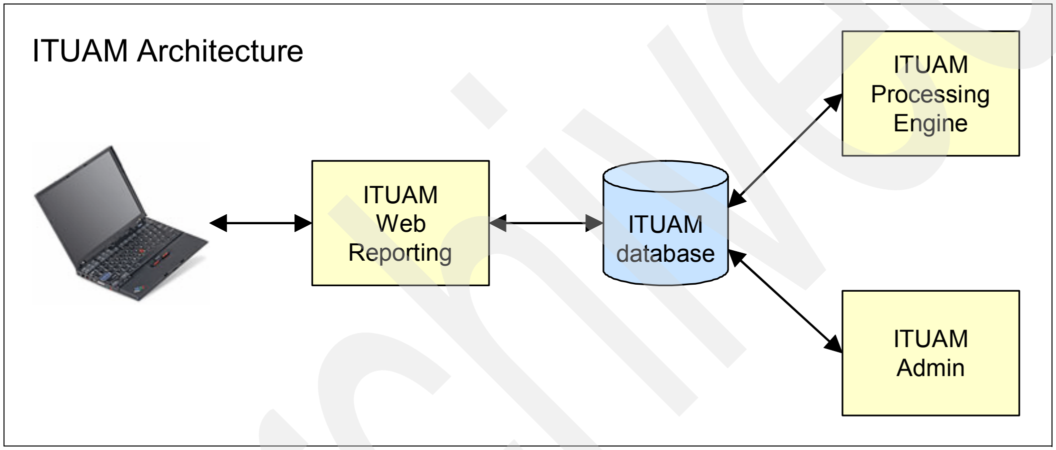


Figure 2-13 ITUAM architecture

1. The ITUAM database stores usage and accounting data plus ITUAM administrative data. It can reside on DB2 (distributed or z/OS), Oracle®, or Microsoft® database systems.
2. The ITUAM Administrator is a Windows-based application with a graphical user interface.
3. The ITUAM processing engine (which has been implemented in both Java and COM) gathers data through various collector modules and updates the database. The collectors and other processes are typically run on a scheduled basis.
4. ITUAM Web Reporting is the user interface to the system. It provides controlled access to a wide range of reports. It is based on Microsoft Internet Information Services and either of two reporting tools:

- Crystal Reports
- Microsoft SQL Server Reporting Services

Full documentation on ITUAM can be obtained from the following Web sites:

- IBM Tivoli Usage and Accounting Manager

  http://www-306.ibm.com/software/tivoli/products/usage-accounting/

- Tivoli Software Information Center

  http://publib.boulder.ibm.com/tividd/td/IBMTivoliUsageandAccountingManager6.1.html

At the time of writing ITUAM is at version 6.1.

2.4.2 Setting up ITUAM

Comprehensive description of ITUAM setup is outside the scope of this book. What we describe here is the minimum necessary to get a report from an Extended Deployment visualization log. Specifically, we process a sample FineGrainedPowerConsumptionStatsCache log.

Installation

We set up our ITUAM test system on a single Windows workstation. We installed the following:

1. Microsoft Internet Information Services (IIS) from Windows components
2. DB2 UDB Express v8.2
3. Crystal Enterprise 10
4. ITUAM base - Setup-ITUAM-6-1.exe
5. ITUAM collector base pack - setup1-6-1.exe

Configuration

We followed the instructions in the following:

- *Tivoli Usage and Accounting Manager Administrator's Guide*

  http://publib.boulder.ibm.com/tividd/td/IBMTivoliUsageandAccountingManager6.1.html

In summary, we did the following:

1. Created a database using the DB2 Control Center.
2. From the ITUAM administrator console, we performed the following:
   a. Created a data source to connect to DB2 (Figure 2-14 on page 90).

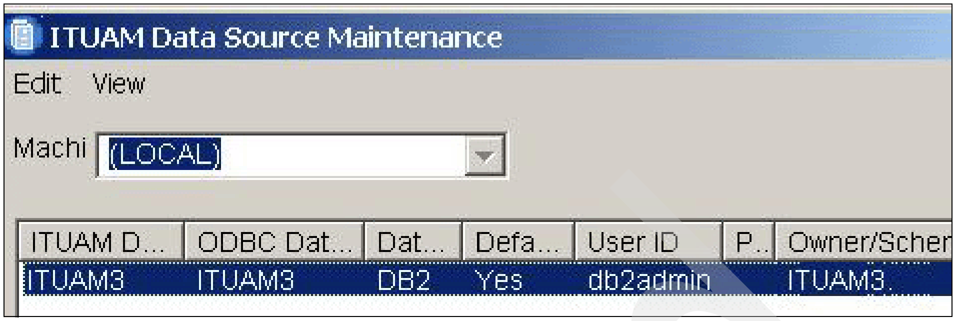


Figure 2-14 ITUAM Data Source Maintenance

b. Initialized the database.

3. Set up the IIS server and tested ITUAM Web Reporting by running the configuration report:

   http://localhost → **logon** → **Reports** → **Run Reports** → **Other** → **Configuration** (Figure 2-15 on page 91).

> **Note:** We had to put the user ASPNET into the DB2ADMNS group.

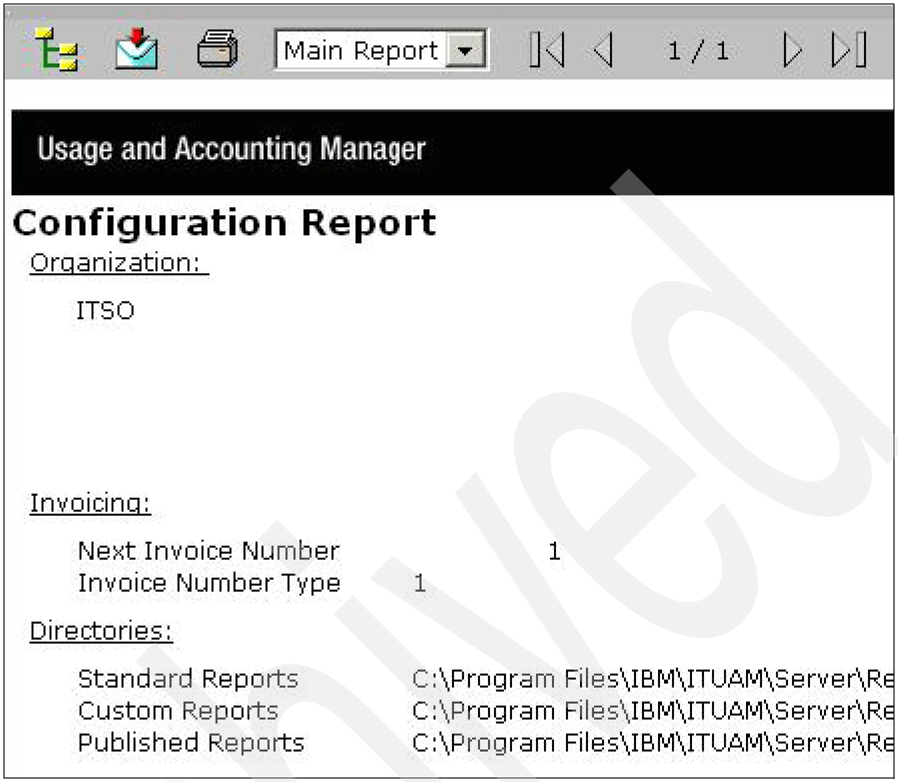


Figure 2-15 ITUAM Web Reporting - Configuration Report

Attention: Currently, only Windows Internet Explorer® is supported by ITUAM Web Reporting.

Further setup

The following further configuration steps were carried out using the ITUAM administrator GUI. (**Start** → **All Programs** → **ITUAM** → **ITUAM Administrator**)

Account code structure

We created a simple account code structure based on appname (Figure 2-16 on page 92).

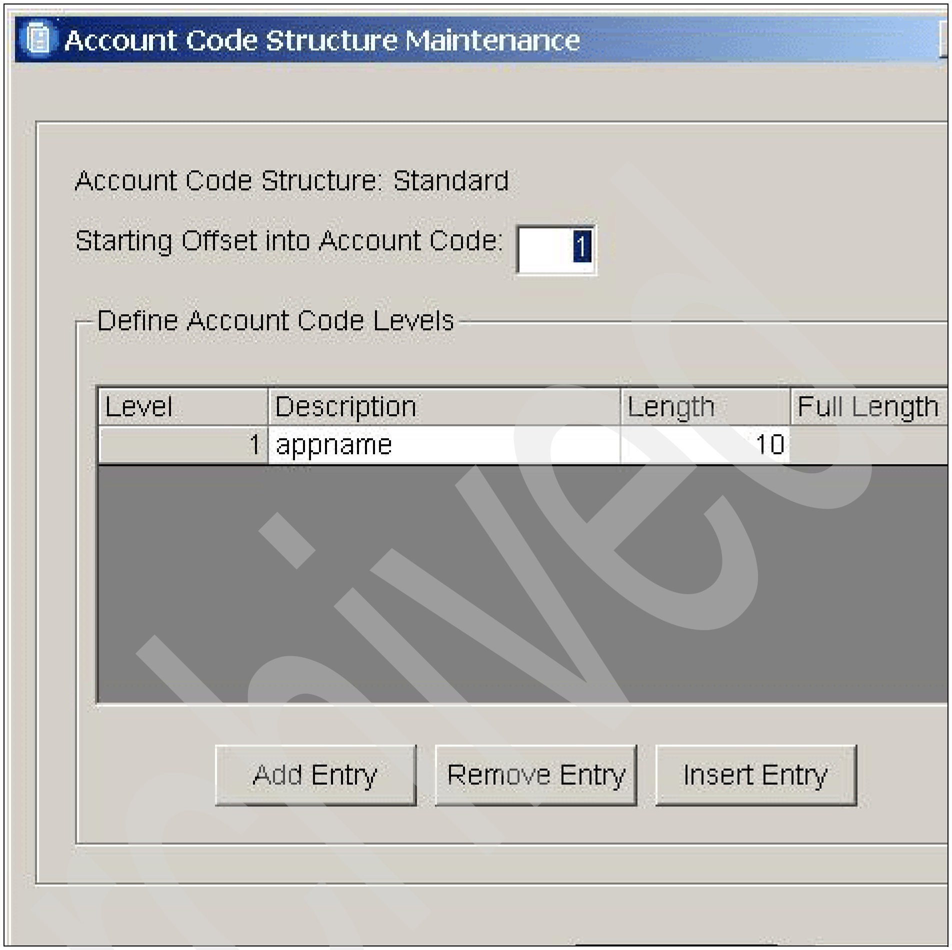


Figure 2-16 Account Code Structure Maintenance

Rate code

We defined a single-rate code to calculate a notional monetary value from powerconsumed in our Extended Deployment log file. This is done through the Rate Codes Maintenance, Figure 2-17 on page 93. (**Chargeback Administration** → **Chargeback Table Maintenance** → **Rate Codes**).

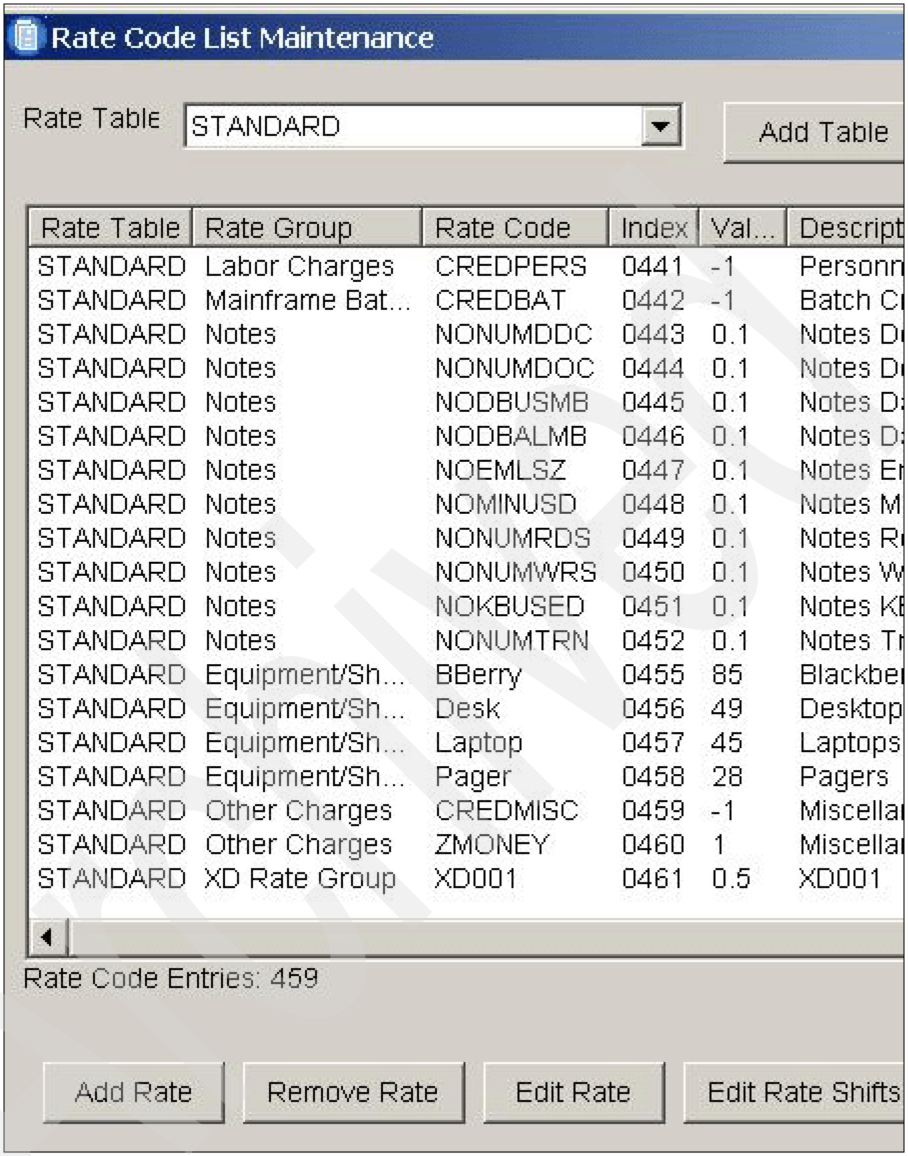


Figure 2-17 Rate Code List Maintenance

2.4.3 Processing the FineGrainedPowerConsumptionStatsCache log

This section describes the manual steps required to input and process an Extended Deployment log. For production running, you would set up ITUAM to process this log automatically.

Configuring and using a data collector

In order to get our data into ITUAM, we needed to use a data collector. We used the ITUAM Universal Data Collector. This is described in the following:

- IBM Tivoli Usage and Accounting Manager Data Collectors for Microsoft Windows User's Guide

  `http://publib.boulder.ibm.com/tividd/td/IBMTivoliUsageandAccountingManager6.1.html`

> **Note:** The Integrator component in TUAM 6.1.1 includes support for processing both FineGrainedPowerConsumptionStatsCache and ServerPowerConsumptionStatsCache log files.

Essentially a data collector takes data from a source and converts it to a standard ITUAM format file referred to as a CSR (Common Source Records) file. To do this, a conversion definition file must be created for our FineGrainedPowerConsumptionStatsCache log. The conversion builder is accessed from the ITUAM Administrator as shown in Figure 2-18.

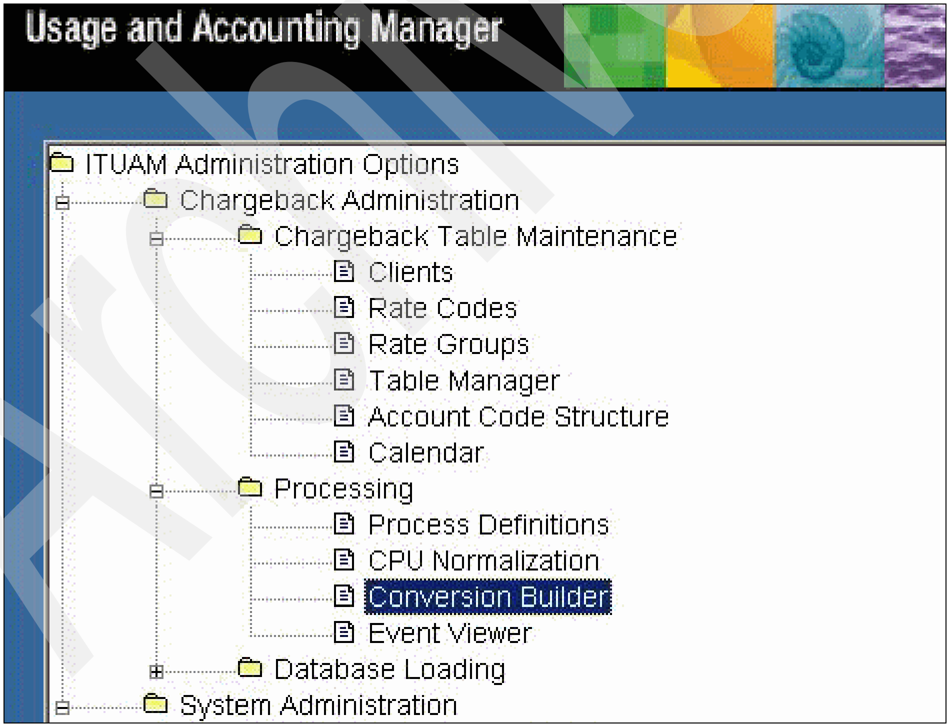


Figure 2-18 ITUAM Administrator - conversion builder

In the conversion builder you map the contents of the log file to the CSR file. Figure 2-19 shows how the format of the input file is specified.

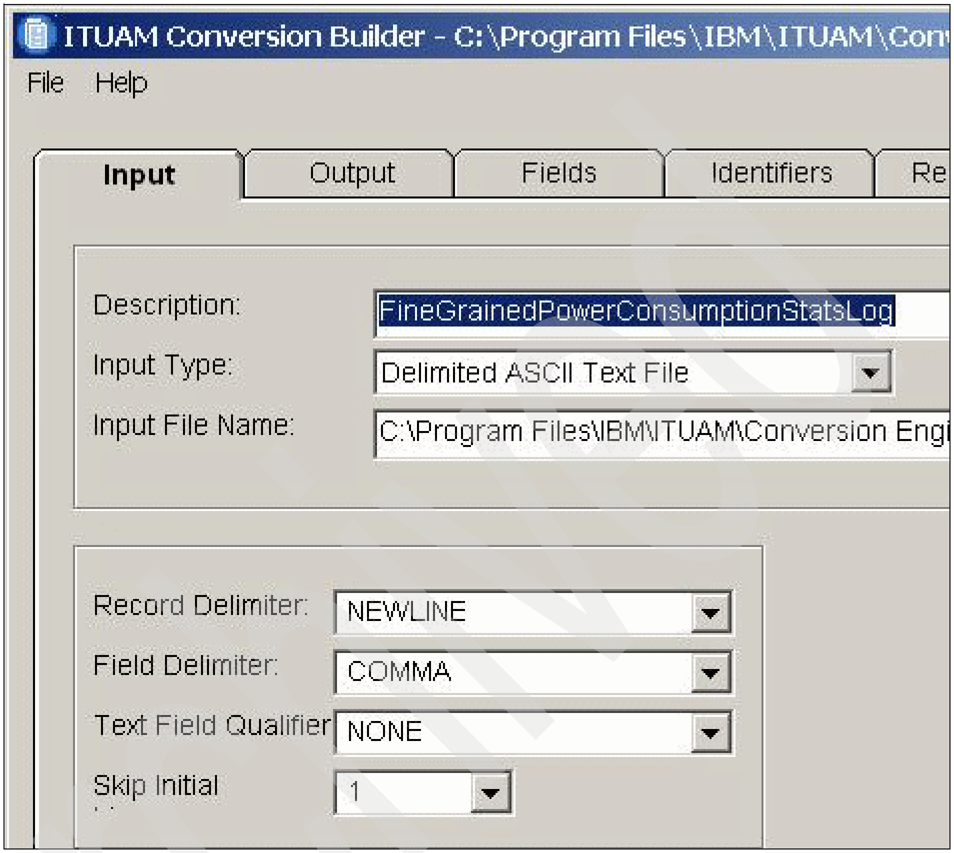


Figure 2-19 ITUAM Conversion Builder

The fields in our log file are easily defined in the conversion builder (Figure 2-20 on page 96), but the range of date and time formats that this data collector can accept is limited. So we had to redefine our time stamp format in Extended Deployment as described previously in 2.3.2, “Formatting the time stamp” on page 75. The format we defined is shown in Figure 2-21 on page 96.

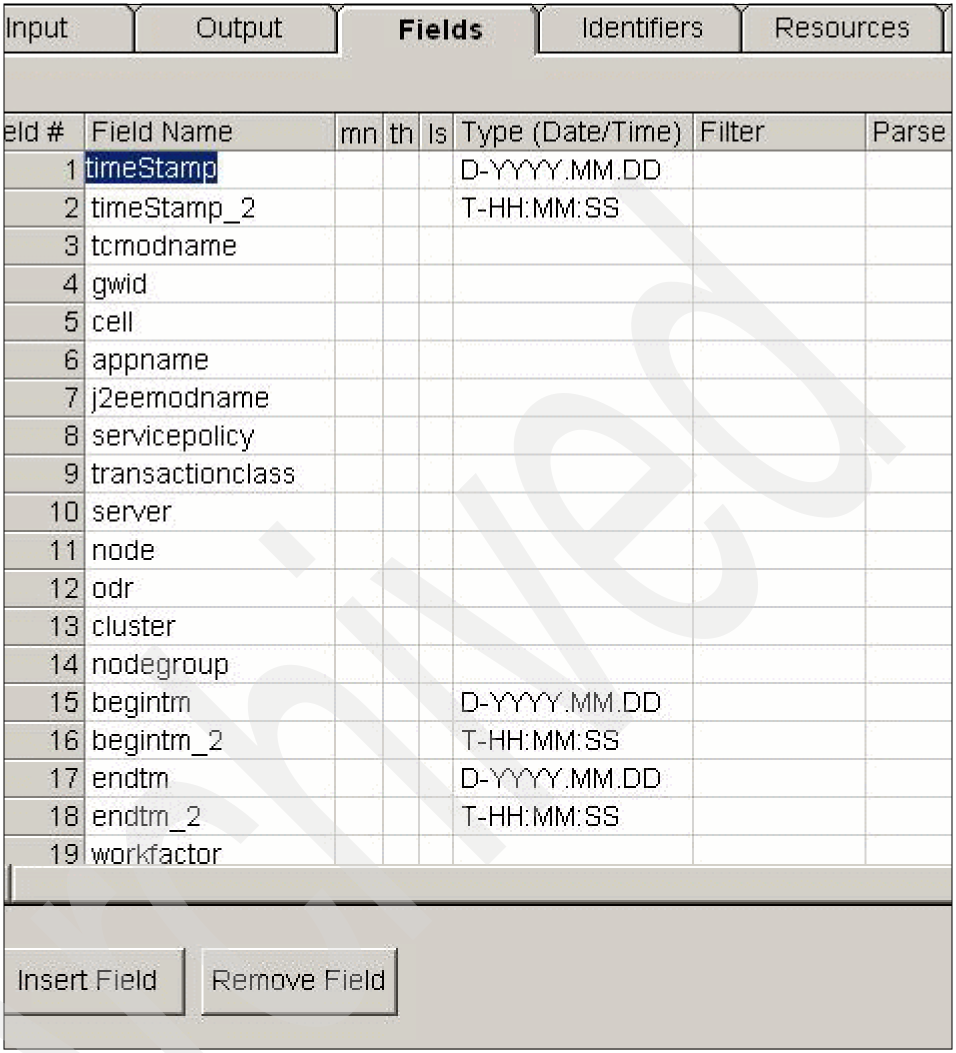


Figure 2-20 ITUAM Conversion Builder - field definitions

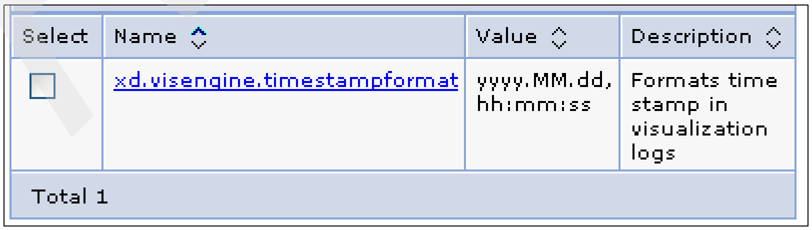


Figure 2-21 Formatting the time stamp for ITUAM

Identifier fields are defined to be used as literals or lookup keys for account code conversion in ITUAM. In our case we chose tcmodname, appname, and cluster (Figure 2-22). Although for this, only appname was used (see “Further setup” on page 91).

.

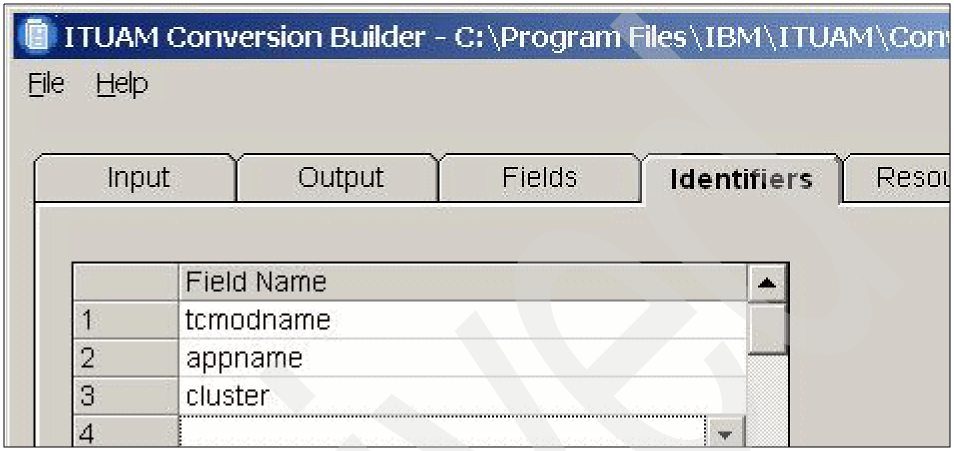


Figure 2-22 ITUAM Conversion Builder - identifiers

The resources fields determine fields that represent resource usage. We chose powerconsumed (Figure 2-23).

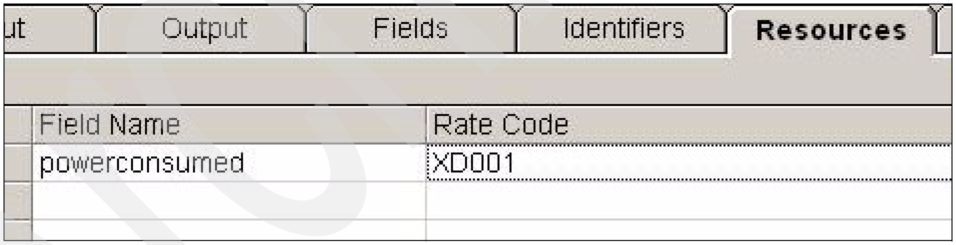


Figure 2-23 ITUAM Conversion Builder - resources

We then had our data converter. This was run successfully and directly from the conversion builder. Figure 2-24 on page 98 shows a sample record from the CSR file.

Header	BigApp2-TC1:!:BigApp2:!:IBMTools.war
Usage Start Date	20061123
Usage End Date	20061123
Usage Start Time	13:20:53
Usage End Time	13:20:53
Shift Code	
Number of Identifiers	5
Identifier name 1	tcmodname
Identifier value 1	BigApp2-TC1:!:BigApp2:!:IBMTools.war
Identifier name 2	appname
Identifier value 2	BigApp2
Identifier name 3	cluster
Identifier value 3	BigApp2
Identifier name 4	SourceName
Identifier value 4	C:\...\FineGrainedPowerConsumptionStatsCache.log
Identifier name 5	SourceLine
Identifier value 5	3
Number of Resources	1
Rate Code 1	XD001
Resource value 1	102.65992

Figure 2-24   CSR Record Example

The CSR file layout is well described in the *ITUAM Administrator's Guide, Appendix B File Layouts*.

Processing the CSR file

Next we needed to update the ITUAM database with the contents of the CSR file. We created a process definition, XDLog1, with default values (Figure 2-25 on page 99) and ran the process.

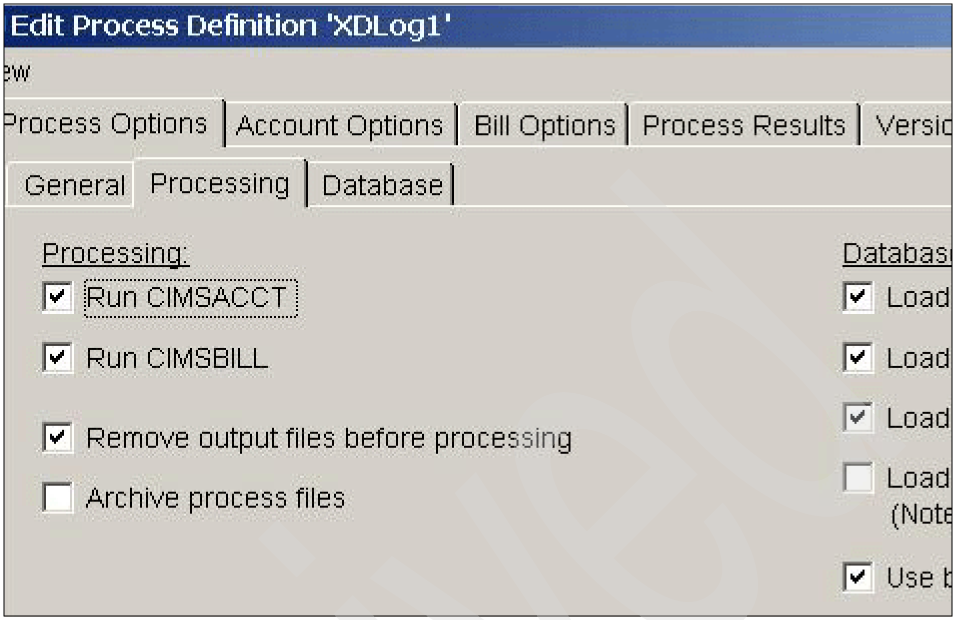


Figure 2-25 Edit Process Definition

2.4.4 Generating an ITUAM report

ITUAM provides a wide range of reports to end users through its web interface. Our purpose here is simply to show data from Extended Deployment log files reported by ITUAM.

To run the report, we logged on to ITUAM Web Reporting, selected **Reports**, and **Run Total Invoice**. (Figure 2-26 on page 100.)

Figure 2-26 Run Total Invoice

Figure 2-27 shows the report.

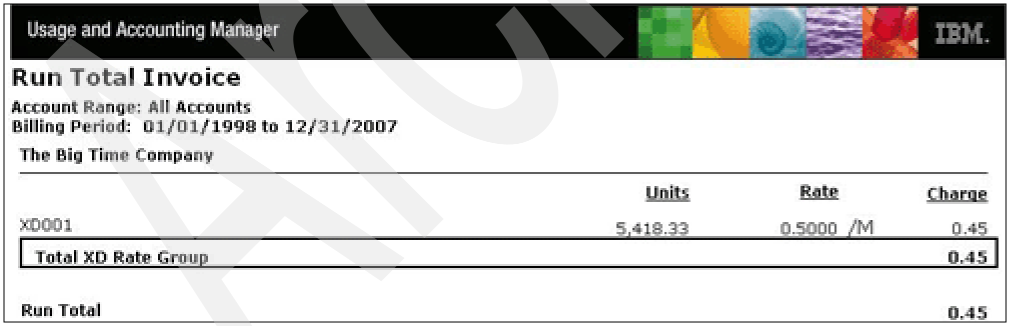


Figure 2-27 Run Total Invoice report

3

Performance monitoring and health management

This chapter describes the features available in WebSphere Extended Deployment to monitor applications and systems for performance and health management. It includes the following topics:

- Monitoring for performance and system health
- Monitoring tools for the runtime environment
- Performing health management
- Health management example
- Monitoring with ITCAM for WebSphere

3.1 Monitoring for performance and system health

Application performance monitoring and health management is an important part of any application environment. As application load and conditions change throughout the day, the system must perform equally well. The autonomic monitoring and health management features of WebSphere Extended Deployment can help you maintain a consistent level of service and detect and address problems quickly before they become apparent to users.

WebSphere Extended Deployment provides advanced support for performance monitoring and health management with the following features:

- Dynamic operations

  In Chapter 1, "Service level optimization" on page 3, we discussed how dynamic operations can help organizations define and meet service level agreements through two key capabilities: the virtualization of WebSphere environments and a goals-directed infrastructure. Applications are monitored against customer-defined goals, and actions are automatically taken to ensure a well-performing system. As resources are needed for spikes in workload demand, application resources are allocated where they are needed most.

- Autonomic health management

  WebSphere Extended Deployment provides a health monitoring and management subsystem that continuously monitors the operation of servers to detect functional degradation that is related to user application malfunctions. Health management provides a policy-driven approach to monitoring the application server environment and taking action when certain criteria are discovered.

- Application deployment management

  The Application Edition Manager is designed to provide interruption-free application deployment in production environments. Using the Application Edition Manager, you can ensure that the users of your application experience minimal loss of service when you install an application update in your environment.

  The Application Edition Manager provides an application versioning model that supports multiple versions of the same application deployed in a cell. This feature gives you the ability to either roll out an application update or revert to a previous level.

- Long-running application management

  Long-running applications typically require more resources and different types of support than the standard lightweight, transactional applications.

WebSphere Extended Deployment introduces a facility, referred to as *business grid*, for supporting these long-running applications. Business grid provides the capability to deploy different types of applications to different nodes within a cell, and to balance the work based on policy information. For more information, please see Part 2, “Long running application extenders” on page 137.

This chapter focuses on autonomic health management for problem detection and avoidance and strategies to reduce or eliminate outages during application deployment. In addition, it will illustrate the use of IBM Tivoli Composite Application Manager for WebSphere to complement the health management features.

3.1.1 Monitor and manage across the application life cycle

Figure 3-1 illustrates a high level view of the application life cycle and how it is addressed in a WebSphere Extended Deployment environment. The focus of this chapter is on the production and monitoring phases.

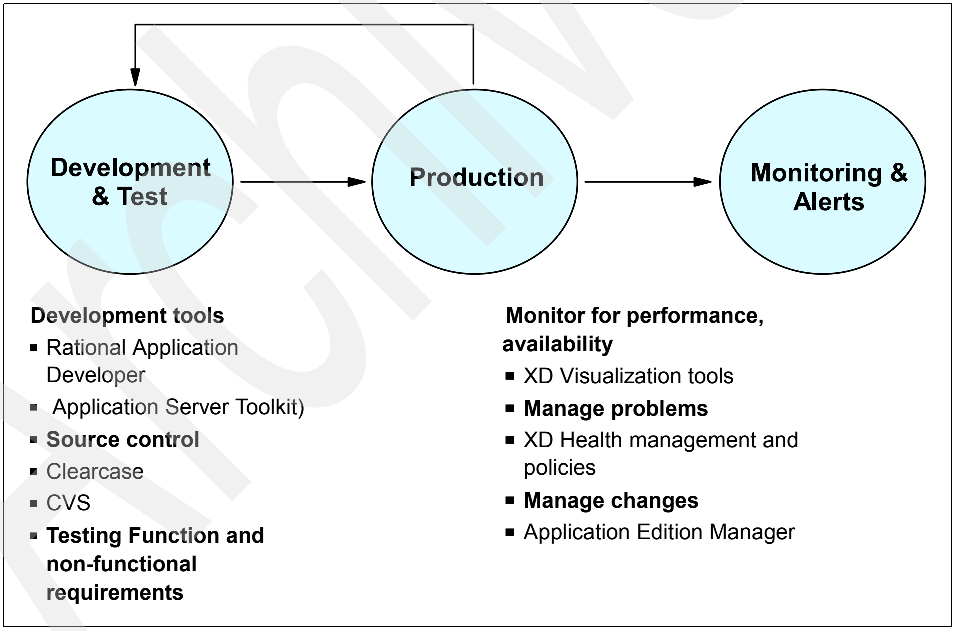


Figure 3-1 Application life cycle in a WebSphere Extended Deployment environment

Select the right development tools and test environment

The development stage encompasses both application development and sufficient testing to ensure an application is ready for production. Development tool selection should be based on the types of applications you develop, ease-of-use, the test environments needed, packaging and deployment capabilities, and source control capabilities. Two development tools particularly suited for WebSphere Application Server are Rational® Application Developer 6.0 and the Application Server Toolkit shipped with WebSphere Application Server V6.1.

In addition to a proper development environment, you should have a comprehensive and robust test environment to ensure that applications are production-ready and will have minimal impact on the existing environment. Testing should be sufficient to ensure that the functional requirements (such as application business logic, user interface and so on) as well as non-functional requirements (such as performance or capacity requirements) are being met. It should also help you understand the application execution characteristics and how it performs under load. Each change to an application or system should trigger the test cycle.

There is nothing unique here with regard to WebSphere Extended Deployment. Proper development tools and test environments are critical to the success of any production environment.

For further discussion on these topics, see *WebSphere Application Server V6.1: Planning and Design*, SG24-7305.

Monitor applications and systems in production

Goals should be set for production environments. These goals can be in the form of individual requirements such as application availability or response time, or they can be defined as a part of a larger work, such as a service level agreement. These goals may vary depending on the type of application (transactional, batch, or compute-intensive) and their importance to the business.

WebSphere Extended Deployment provides visualization features that help you understand how a system is performing. These features allow you to take a quick snapshot of the system and application state, or they can be used to generate alerts that can initiate actions.

If you need extended monitoring and deep-dive troubleshooting, ITCAM for WebSphere can be used. Note that the current version of WebSphere Extended Deployment does not support SNMP alerts to the enterprise wide monitoring products.

Perform health management to identify and manage problems

Health management is used when a problem is suspected and needs to be identified and managed. With autonomic health management, WebSphere Extended Deployment uses a policy-driven approach to monitor applications and to take action when certain conditions are met, for example when response time criteria is exceeded. Once WebSphere Extended Deployment detects a problem, it can take pre-configured actions designed to bypass the problem or send an alert to the operator.

Manage changes to the applications

There are occasions when applications need to be redeployed, whether to implement a fix or as a regular update from the development team. The Application Edition Manager of WebSphere Extended Deployment can be used to redeploy applications with minimal disruption to users.

3.2 Monitoring tools for the runtime environment

WebSphere Extended Deployment has extensive monitoring features that provide the basis for its autonomic features, including health management. These features work behind the scenes to keep track of application and system performance. Information gathered by the monitoring features is also exposed to the WebSphere administrator through the visualization tools, providing useful views of the runtime environment that allow you to chart application performance against business goals and providing an interface to manage the complex applications.

Event notification capabilities alert operators about decisions made by autonomic managers. Notifications can represent either planned or unplanned events. Planned events are those (expected) events for which the Extended Deployment runtime has an action plan. If Extended Deployment is operating in automatic mode, the action plan runs, and you can view a notification of the action that was taken. In supervise mode, you can view and approve the action plan. If an unplanned event occurs (an event that is not assigned to an action plan), you can be notified with a warning that lets you know something unexpected happened. Notifications can be viewed through the administrative console. E-mail notifications can also be made.

3.2.1 Visualization tools

The administrative console in WebSphere Extended Deployment can be used to view the current runtime environment, including utilization and performance data. This is particularly helpful in a virtualized environment where the runtime resources can be transient. These views are useful for operators and administrators that need to understand the current state of the system.

The following is a brief overview of the visualization tools.

Runtime Topology view and dynamic charting

The Runtime Topology view (**Runtime Operations** → **Runtime Topology**) allows you to view or to change the runtime environment. You can adjust configuration options to keep informed of the activities occurring in the environment, and to make changes based on the information yielded in the topology view and dynamic charting.

Three perspectives are available to display runtime information: node group, application, and service policy. These perspectives determine which topology and charts display in the page. Within the perspectives, you can choose to view all the information or information for a specific instance.

Dynamic charting of objects in the Runtime Topology view provides interactive graphing of runtime data. Statistics such as availability, response time, traffic, and throughput are available. The view refreshes on the same configurable interval as the Runtime Topology view. A wide range of options from which you can create various charts is provided.

Runtime Tasks view

The Runtime Tasks view (**System Administration** → **Task Management** → **Runtime Tasks)** allows the administrator to view and act on actions generated by autonomic managers. For example, when the health controller finds that a health policy criteria was matched and an action is required, that action is recorded in the tasks view. If the reaction mode is supervise, the operator can accept or deny the action from this view. If the reaction mode is automatic, the operator can see what action was taken.

Visualization data service

The visualization data service logs historic data from the runtime topology in text files for reuse with other charting programs. This service can be configured through the administrative console (**System administration** → **Visualization Data Service**).

3.3 Performing health management

With health management, you define health policies designed to identify potential problems, the action to take when the event occurs, and how the actions will be taken.

When to use health management:

The monitoring subsystem introduces a small performance impact, so you would normally only enable health management in systems where you are having or anticipate having problems. In the event that you do wish to enable the health management features in a stable environment, it is recommended that you maximize the data collection interval and avoid having actions taken automatically.

Examples of when health management is appropriate include:

- Your system is having issues, for example poor response time, that need to be monitored and resolved. Customized health policies can help you monitor for specific conditions that indicate a problem is occurring, and take action to bypass the condition.
- You have installed a new system and would like to monitor for potential problems. Using the default health policies and taking supervised actions or operator alerts can be helpful in identifying problems that occur in new or upgraded installations.

3.3.1 Health management overview

The health management subsystem (Figure 3-2 on page 108) consists of two main elements: health policies and the health controller. Health policies define conditions that can indicate a problem, where to monitor for this problem, the action to take, and whether the action is done automatically or by an operator. The health controller monitors for these conditions and performs the actions defined by the policies.

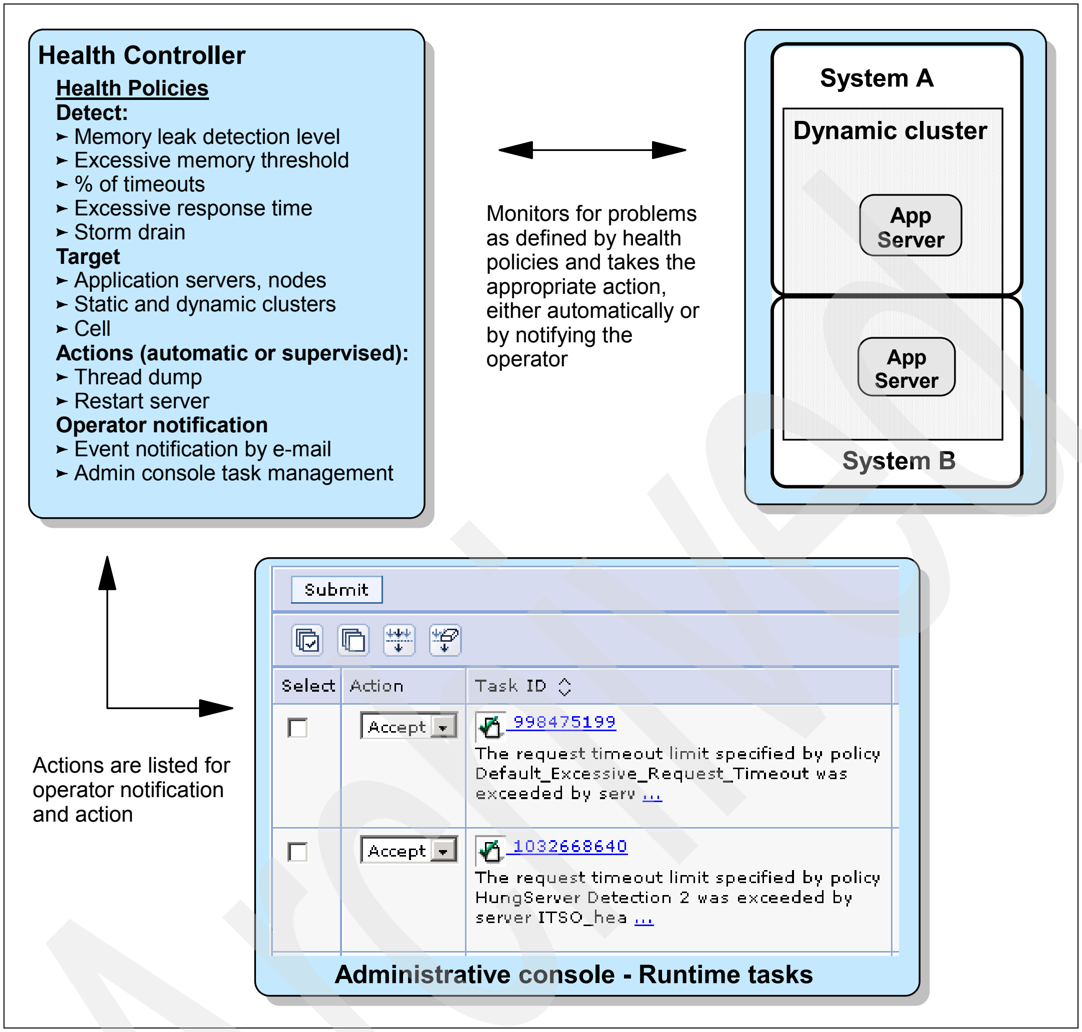


Figure 3-2 Health management overview

3.3.2 Creating health policies

Health policies are available for a specific set of conditions. You can use the cell-level default policies shipped with WebSphere Extended Deployment, modify the criteria for the default policy, or create your own.

The conditions a policy can be measured against include the following:

- Excessive memory consumption, which can indicate a memory leak. The default policy sets this at 95% of the JVM™ heap size for 15 minutes.

- Excessive response time, which can indicate a hung server. The default policy sets this at 120 seconds.
- Excessive request timeout, which can indicate a hung server. The default policy sets this at 5% of the requests timing out.
- The volume of work performed by a server.
- Storm drain detection, which relies on change point detection on a given time series data.
- The age of the server.

Health policy targets can be any or all servers in a cell. A server can be monitored by multiple health policies.

There are two types of action that can be chosen when the criteria for a policy are matched to current conditions.

- Take a JVM heap dump. This is only available for the IBM JDK™ and only for request timeout and memory leak conditions.
- Restart server. Note that recycling a server is not really a fix, but a temporary means to keep things going until the problem gets fixed.

You can elect one of two modes for the actions:

- Automatic - the health management system executes a predefined action when it detects a health policy violation.
- Supervise - the health management system creates a runtime task proposing one or more reactions. The system administrator then can approve or deny the proposed actions.

Note: It is best to run in supervise mode for a while before switching to the automatic mode.

Best practices for creating health policies

Health management features should be planned and used carefully to avoid accidentally introducing negative impacts to the system. The following are some basic suggestions for planning your health management policies:

- Understand the environment, including its capacity, usage, and loads. These numbers will help you plan your policies.
- When defining new health policies, consider the effect it will have on your production system. For example, if you have only two JVMs supporting a high- usage application, you do not want to configure a health policy that could recycle one of them during peak times. This could cause the remaining

JVM to crack under the heavy load. Configure your health policy to exclude those.

- If you do see a need to recycle, you can plan a manual recycle before peak time.
- For any new policy, set the reaction mode to supervise. Once you are confident about the effect, you can choose the automatic mode.
- WebSphere Extended Deployment provides a set of default health policies that run in supervise mode. You can use these policies on all the new installations to monitor for any possible issues. The first few days are very critical for all the new applications.
- When you choose to take a thread dump, it is advisable to restart the server also.

3.3.3 Configuring the health controller

The default settings of the health controller in WebSphere Extended Deployment are a good start. However to fully utilize the system, there are settings that can be adjusted. As with any tuning activity, it is important that you understand the basic needs of your system, the peaks and lows for system usage, and the capacity of the hardware infrastructure.

> **Note:** As a best practice, apply your changes to the Runtime tab and test the changes before committing them. Once you are comfortable with the new settings, save to configuration.

The health controller settings can be configured by going to **Operational policies** → **Autonomic managers** → **Health controller**

- **Enable health monitoring**

  Enables or disables the operation of the health controller. When enabled, the health controller continuously monitors the health policies in the system. You can disable the health controller without removing the health policies from the system.

- **Control cycle length**

  This setting specifies the time between consecutive health checks. The value is specified in minutes and ranges from 1 to 60 minutes. Longer control cycles reduce the health monitoring load but could be less efficient depending on the condition for which you are monitoring.

- **Maximum consecutive restarts**

  This setting specifies the number of attempts made to revive an application server after a restart decision is made. If this number is exceeded, the restarts are disabled for the server.

- **Minimum restart interval**

  Controls the minimum amount of time that must elapse between consecutive restarts of an application server instance. If a health condition for an application server is breached during that time, the restart is set to a pending state. When the minimum restart interval passes, the server restart occurs. The value can range from 15 minutes to 365 days, inclusive. A value of 0 disables the minimum restart value.

- **Restart timeout**

  The restart timeout specifies how long to wait for a server to stop before explicitly checking its state and attempting startup. The value for this should take into account the time taken by your application to stop and start.

- **Prohibited restart times**

  This setting can be used to specify the time and day of the week during which a restart of an application server instance is prohibited. You can also specify multiple time blocks, if needed.

  > **Note:** There are times when you would not want to restart the server automatically, for example, during troubleshooting. Another example would be during peak times when a recycle of the JVM would put extra load on the other servers in the cluster, thus creating a domino effect.

If you experience problems using the health management features, the WebSphere Extended Deployment Information Center has an article with suggestions:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp?topic=/com.ibm.websphere.xd.doc/info/odoe_task/rodhealthfail.html

The health management log messages are stored in the node agent log for the node where the health management controller is running. To find these logs, you first need to identify which node is hosting the health controller. You can identify the node hosting the Health Management Controller by navigating to **Runtime Operations** → **Runtime Topology** in the administrative console. Look for the red cross icon on the Runtime Topology panel. In Figure 3-3 on page 112, it is running on ITSOodr1Node01.

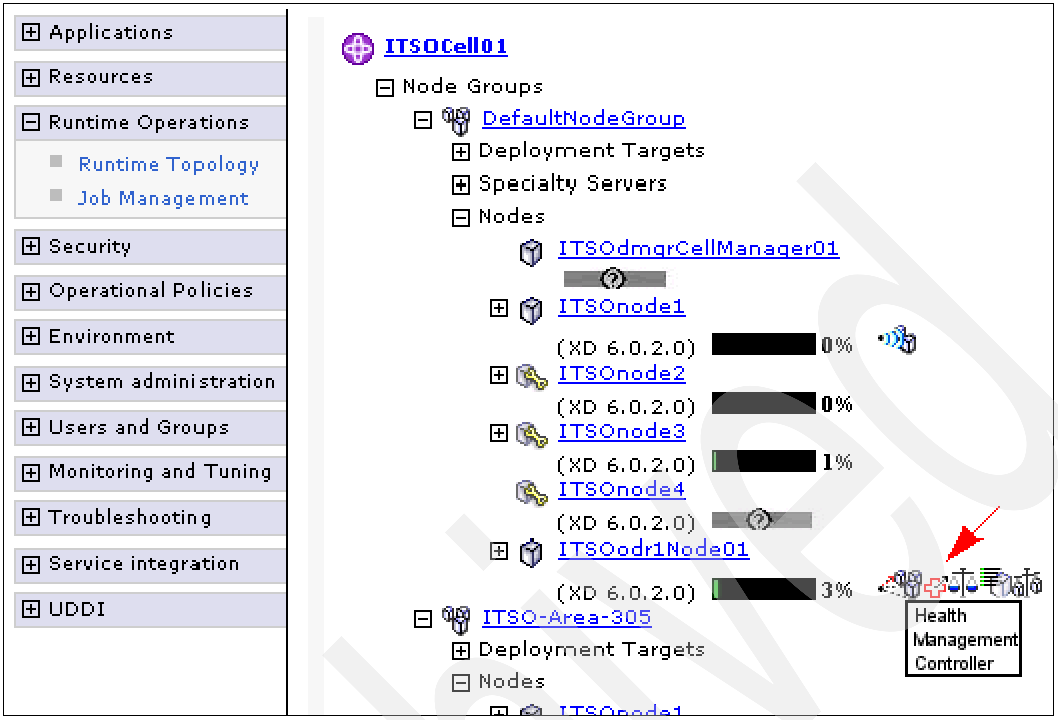


Figure 3-3 Health management controller runtime location

3.3.4 Monitoring health management

To see actions taken by the health controller, navigate to **System Administration** → **Task Management** → **Runtime Tasks** (Figure 3-4 on page 113).

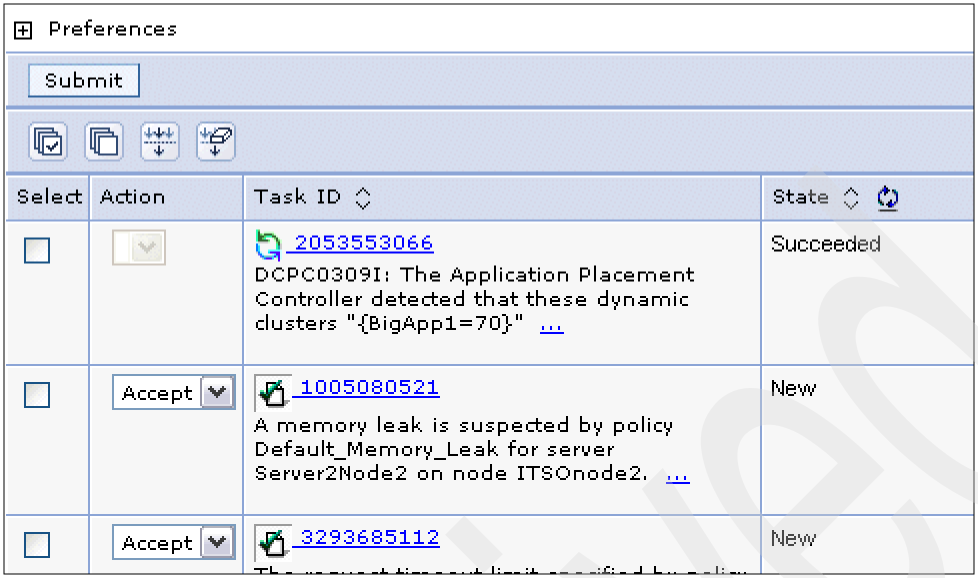


Figure 3-4 Runtime tasks

This view shows you the conditions that were detected and the actions taken as a result of your health policies. If there are any actions waiting in supervise mode you will see those here and can act on them.

Using event notifications

The event notification feature in WebSphere Extended Deployment eliminates the need for an operator to manually monitor the administrative console for tasks generated by the health controller and application placement controller. When notifications are enabled and a task is generated, an e-mail notification is sent to each of the e-mail addresses specified.

To configure and enable the notification feature go to **System Administration** → **Task Management** → **Notifications**.

You will need the SMTP server host name and port, a user ID and password for authentication with the SMTP server, and the e-mail address(es) to which to send the notification.

3.4 Health management example

To illustrate the use of the health management features, we will examine a fictional example.

A customer is running WebSphere Extended Deployment in production to support various applications. One application is having problems that include poor response time and a hung JVM. This application is a mission-critical, high-volume transaction application. There are two immediate challenges:

- Keep the application running and responsive during the peak times.
- Find the root cause of the problem and fix it.

3.4.1 Define a health policy

The first task is to define a health policy. We have the option of using default health policies; however, because we understand the current problem, we can create a custom policy to help with problem determination. In this scenario, the issue is slow or no response from the application. The policy type that most suits the problem is the policy for timed out requests (Figure 3-6 on page 115). Please note that multiple policies can be used for a single server.

To create a new health policy, go to **Operational Policies** → **Health Policies** → **New**. The wizard allows you to select the health condition and to set the parameters. In this example, we select the excessive request timeout condition (Figure 3-5).

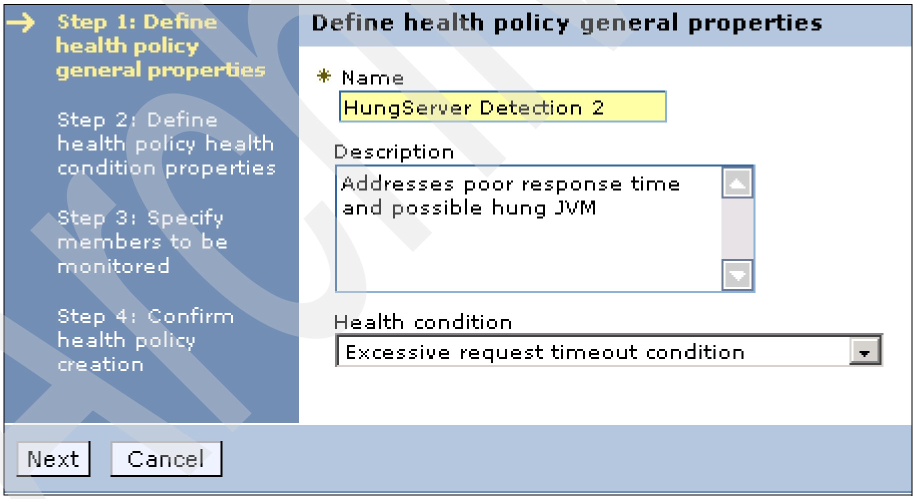


Figure 3-5 Define a new health policy

The policy condition properties we selected (Figure 3-6 on page 115) indicate that action should be taken when more than 30% of the requests start to time out. When this condition occurs, the health controller creates a task to take a memory dump to use for diagnostic purposes and then restarts the server. The reaction mode is set to supervise. This is a cautious approach that allows the operator to decide if the action is appropriate. Notifications are enabled and configured for e-mail to be sent to the operator.

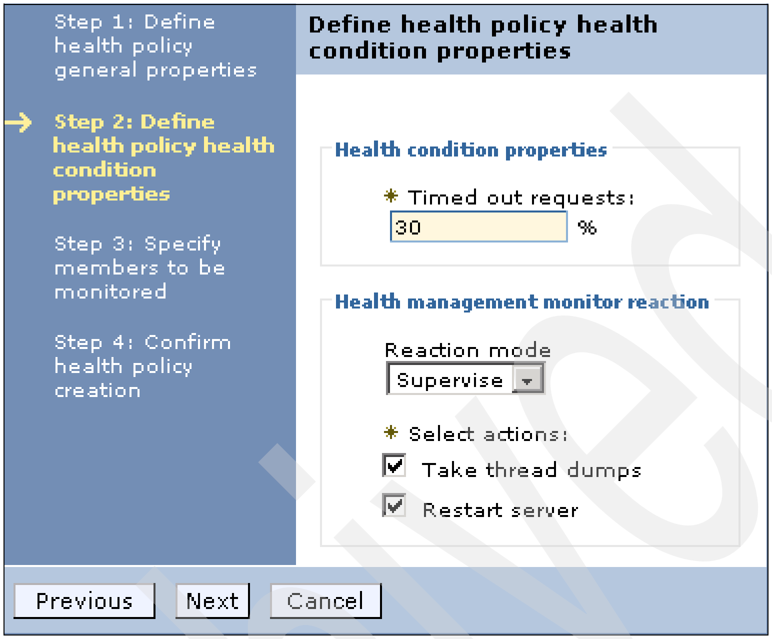


Figure 3-6 Health condition properties

The policy is applied to the two servers in the node group where the application runs.

The new policy can be seen in the administrative console (Figure 3-7).

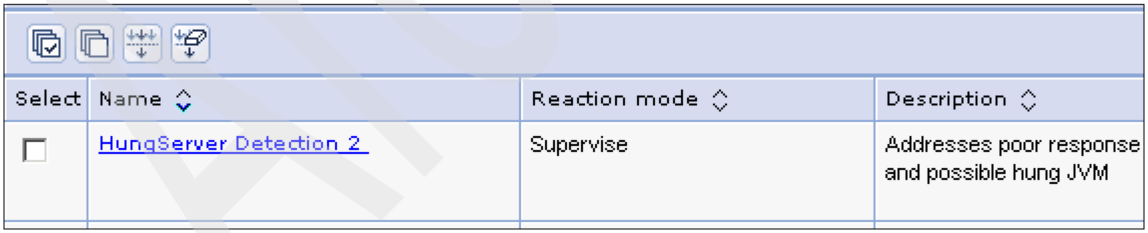


Figure 3-7 New health policy

3.4.2 Taking a thread dump

Once the policy is applied, load is generated for the application to recreate the problem. When the required timeout conditions are met, the action is invoked and an e-mail notification is sent to the operator. The operator checks the runtime tasks, where he can see the action waiting for approval. He decides to let the action take place (accept) and submits it.

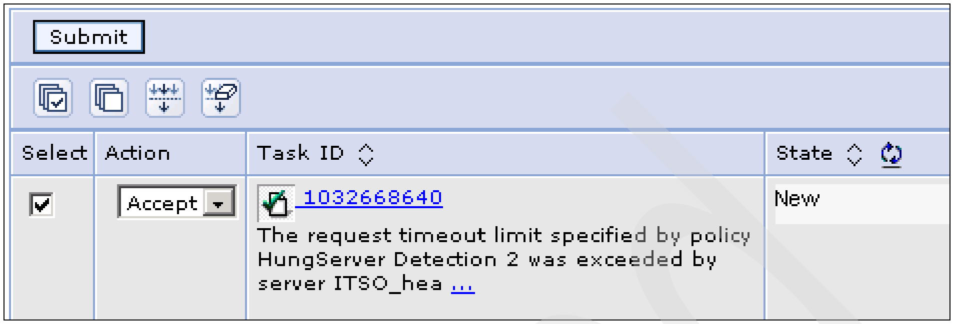


Figure 3-8 Runtime task

The thread dump is taken and the server is recycled. Once the action is complete, you will see that in the Runtime Task view. The operator sends the thread dump to the application developers for debugging.

Meanwhile we decided to use the automatic mode to enable automatic server restarts to keep the application running until a final resolution is achieved. The action was modified to restart the server only (no thread dump) and the reaction mode was changed to automatic.

3.4.3 Deploying the update

The application team finds a problem in the application and sends a new update to operations for deployment. The Application Edition Manager is used to install the new updated application.

The health policy is changed again so that the action includes taking a thread dump as well as restarting the server, and the reaction mode is set back to supervise. This is kept in place for several days to make sure that the problem is resolved.

3.5 Monitoring with ITCAM for WebSphere

Note: This is a high level overview of using IBM Tivoli Composite Application Manager or ITCAM for WebSphere to complement the health management features in a WebSphere Extended Deployment environment. For an in-depth look at ITCAM for WebSphere, see *IBM Tivoli Composite Application Manager V6.0 Family: Installation, Configuration, and Basic Usage*, SG24-7151.

WebSphere Extended Deployment delivers strategic value to customers by enhancing the existing capabilities of WebSphere Application Server. WebSphere Extended Deployment adds the ability to dynamically respond to variable demand for resources and ensure that goals-directed application performance is maintained. WebSphere Extended Deployment can monitor and respond to operational issues and hence improves the application performance and availability.

ITCAM for WebSphere has advanced capabilities for WebSphere application monitoring. This can help isolate and analyze the cause of problems detected by WebSphere Extended Deployment. ITCAM for WebSphere provides the deep-dive view into the state of discreet transaction events. It can be used by support teams to identify root causes. ITCAM for WebSphere also enhances the WebSphere Extended Deployment monitoring by integrating with other Tivoli infrastructure management solutions to provide an enterprise-wide view.

Both products, when used together, complement one other. By using both we can build solutions to increase productivity and customer satisfaction. WebSphere Extended Deployment focuses on keeping the environment healthy. ITCAM uncovers the source of application performance problems and lowers the amount of time and effort needed to resolve them.

Here is brief comparison among the monitoring capabilities of both the products.

Table 3-1 Feature comparison

Features & functions	WebSphere Extended Deployment	ITCAM for WebSphere
Administration		
Account Management	x	x
Server Management	x	x
Monitoring on-demand (L1 L2 L3)		x

Features & functions	**WebSphere Extended Deployment**	**ITCAM for WebSphere**
Managing Sever		x
Availability metrics		
Virtualized Environment (ODR, dynamic cluster etc.)	x	
Server		x
Web server	x	x
Custom groups		x
Enterprise	x	x
Server statistics overview		x
Recent activity display		x
Systems resource comparison		x
Problem Determination		
In-flight request search		x
Server activity display		x
Memory analysis		x
Heap analysis		x
Heap dump	x	x
Memory leak	x	x
JVM thread display		x
Trap & alert management	x	x
SNMP alert		x
Performance Analysis and Reporting		
Reports		x
Application reports	x	x
Server reports	x	x
Top reports and trends analysis	x	x

Features & functions	WebSphere Extended Deployment	ITCAM for WebSphere
View reports		x
Daily statistics		x
Total summary view	x	

3.5.1 Using ITCAM for WebSphere

ITCAM for WebSphere has two main components: a managing server and data collectors. The managing server collects data from collectors running on J2EE, CICS, and IMS™ servers. The managing server prepares the data for real-time, displays, and stores it into the data repository. The data collectors run inside the application servers collecting data to send to the managing server.

To integrate WebSphere Extended Deployment with an existing ITCAM for WebSphere installation, you need to do the following:

- Install the data collector on the WebSphere Extended Deployment application servers.
- Configure the new servers to ITCAM for WebSphere.
- Collect and analyze the data.

Special considerations for ITCAM V6.0 and WebSphere V6.1

Note: This assumes that you installed and configured the IBM Tivoli Composite Application Manager for WebSphere V6.0 environment, including the managing server.

For information about this, see the following:

- *IBM Tivoli Composite Application Manager V6.0 Family: Installation, Configuration, and Basic Usage*, SG24-7151
- Tivoli Composite Application Manager for WebSphere documentation in the Tivoli Product information center at:

  http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?toc=/com.ibm.itcamwas.doc/toc.xml

During this project we used ITCAM for WebSphere V6.0, which did not automatically support WebSphere Application Server V6.1 (the base for

WebSphere Extended Deployment in our lab systems). To add this support, two things were necessary:

1. During the data collector installation wizard, the WebSphere Application Server V6.1 installation was not recognized. We manually entered the install path (Figure 3-9):

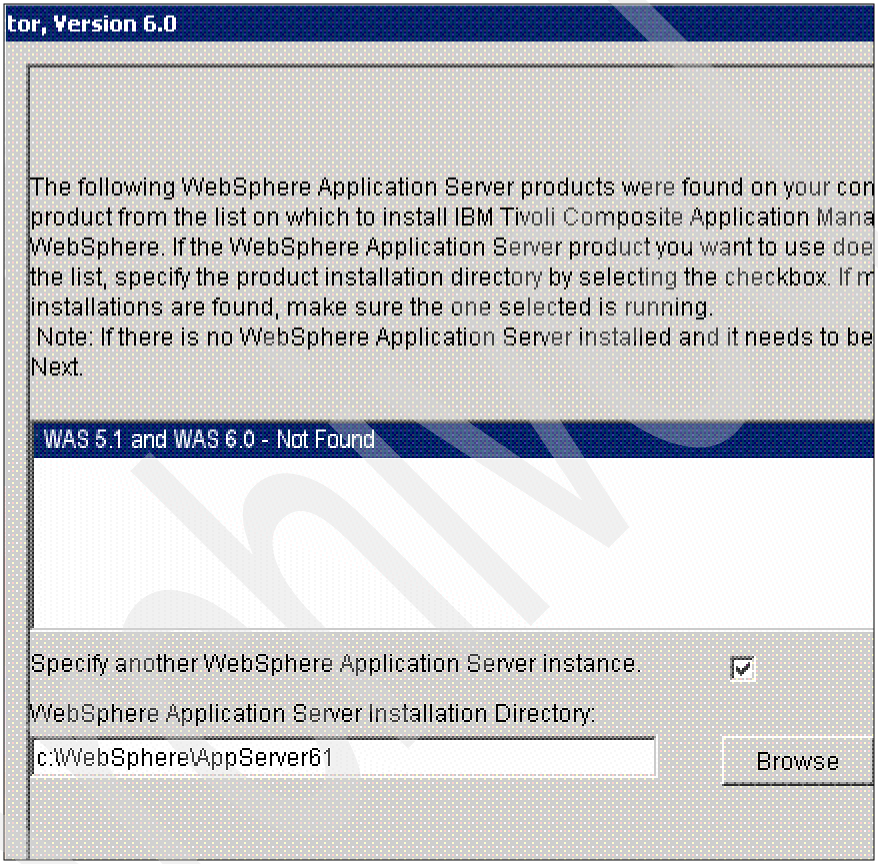


Figure 3-9 DC install wizard: WebSphere Application Server location

We ensured that the window that displayed a summary of information gathered by the installer was correct (Figure 3-10 on page 121).

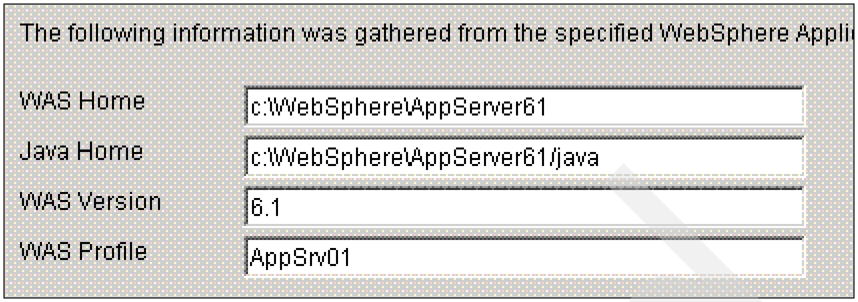


Figure 3-10 DC install wizard: WebSphere Application Server V6.1 location summary

2. After the installation was complete, we installed the following to the base data collector installation.

   – Fixpack4 - 6.0.0-TIV-ITCAMfWAS_MP-FP0004.tar

   – IF002 - 6.0.0.4-TIV-ITCAMfWAS_MP-IF0002.tar

   You can download the updates and updateinstaller from the ITCAM support site:

   http://www-1.ibm.com/support/docview.wss?rs=2344&context=SS3PGL&dc=D400&uid=swg24013978&loc=en_US&cs=utf-8&lang=en

   To install these updates, perform the following actions:

   a. Expand Fixpack 4 (6.0.0-TIV-ITCAMfWAS_MP-FP0004.tar) and the update installer (ITCAMfWAS_V6_UpdateInstaller.tar) to a temporary directory.

   b. Update the silentUpdate.properties file with required information. For WebSphere Application Server 6.1 node, you need to update the correct value of the was plug-in location:

      wasPlugins.location=/WebSphere/AppServer61/plugins.

   c. Run the **`silentUpdate`** command with **`-install`** option.

   d. Repeat step 1 and step 3 to install IF002.

   e. After the updates are installed, recycle the WebSphere processes on the node, including nodeagent.

Troubleshooting

If you are not able to start the application server, check the am.dll file in the lib directory of your data collector install location. If the file is not there, copy am_was_6_ibm_15_windows.dll to am.dll.

- Review the log files under log dir of the DC install location.

- Look for the plugin/itcam directory. If it is not there, it means the updater did not finish properly. You should uninstall the updates, check for the permissions to write in the plugin directory, and run the install again.
- For further verification, logon to the Application Monitor console.

3.5.2 Application Monitor console

The Application Monitor (AM) console is the browser-based interface to administer and use ITCAM. You can login to the console from a Web browser with the following URL:

```
http://<servername:port>/am/console
```

```
For example:
```

```
http://localhost:9080/am/home
```

When you log on to the AM console, the first thing you see is the availability display (Figure 3-11 on page 123).

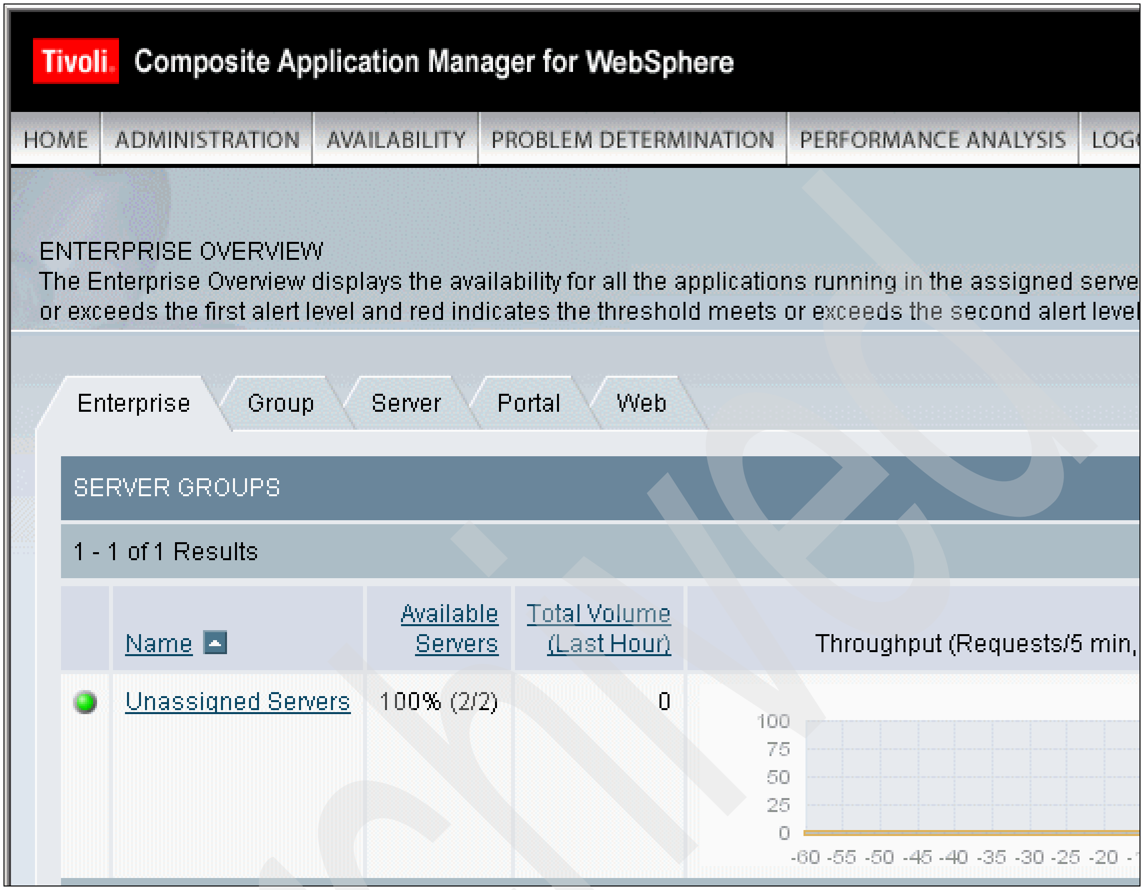


Figure 3-11 AM console

The availability display shows all data collectors that you have access to with the following level of detail:

- **Enterprise** shows the overview of all server groups to which you have access.
- **Group** shows all servers within a particular server group.
- **Server** shows the overview page of an application server.
- **Portal** shows a portal server overview.
- **Web** shows Web servers overview.

3.5.3 Adding a new server to ITCAM for WebSphere

During installation of the data collector, the managing server is identified and communication is established. However, the data collector must still be configured to the managing server. Use the following steps to configure new servers:

1. Logon to the AM console and select **Administration** → **Server Management** → **Data Collector Configuration**.
2. From the Menu on the left hand side pick **Unconfigured Data Collectors**. This displays a list of servers that have data collectors installed but have not been configured.
3. Select the server by checking the box in the table to the right. Select a configuration from the drop-down, in this case, the J2EE default configuration (as opposed to the CICS or IMS default configuration). Click **Apply** to configure the data collector.

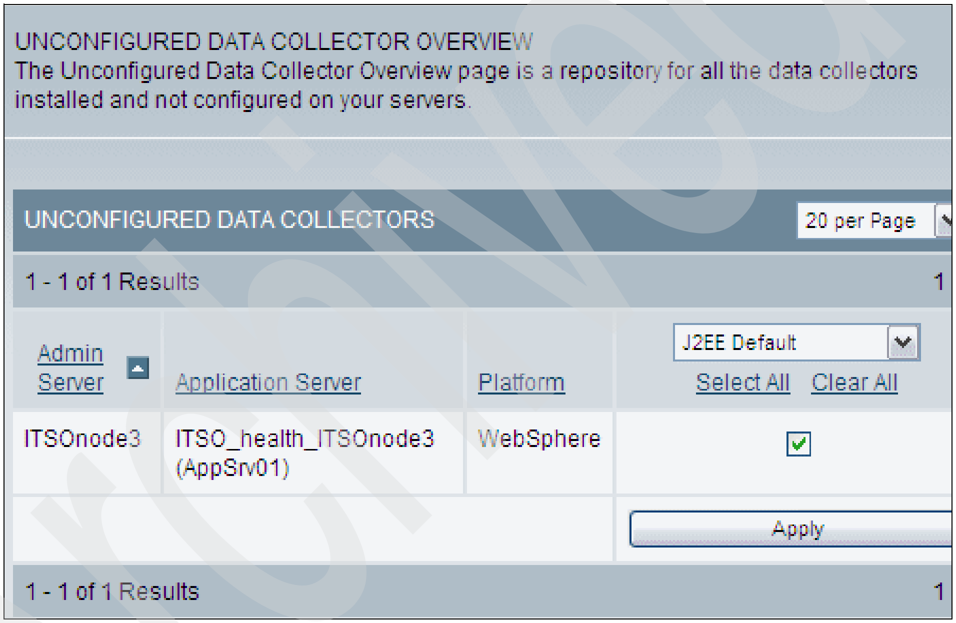


Figure 3-12 Data collector Configuration

4. Select **Configured Data Collectors** from the menu to see the server in the list of configured data collectors.

CONFIGURED DATA COLLECTORS

1 - 2 of 2 Results

Admin Server	Application Server	Configuration Name	Platform	Status	Se
ITSOnode3	ITSO_health_ITSOnode3 (AppSrv01)	J2EE Default	WebSphere	Disable	
kcgl6hcNode01	server1 (AppSrv01)	J2EE Default	WebSphere	Disable	

1 - 2 of 2 Results

Figure 3-13 Configured data collectors

5. To verify the communication between the data collector and the managing server, go to **Administration** → **Managing Server** → **Self diagnosis**. Look for PPECONTROLLER and PPPROBE, which show you the data collector server names and IPs.

PPECONTROLLER	e023e618-466b-db01-cb58-eae074f8a8a1.1000	NT2000 5.2 build 3790 Service Pack 1	9.42.171.52
PPECONTROLLER	01e73d82-fd64-db01-9320-3269181ab231.4052	NT2000 5.1	9.42.170.134
PPEPROBE	e123e618-466b-db01-cb58-eae074f8a8a1.1000	NT2000 5.2 build 3790 Service Pack 1	9.42.171.52
PPEPROBE	02e73d82-fd64-db01-9320-3269181ab231.4052	NT2000 5.1	9.42.170.134

Figure 3-14 Verify the data collector

3.5.4 Server overview

The AM console has a large variety of views that present information about systems monitored and data collected. The Server Overview view is just one of many. However, it is a useful tool in getting a high-level view of the monitored servers and their current state.

You can see the Server Overview page by navigating to **Availability** → **Systems Overview** → **Server.** Select a group and server to view. The view shows the latest data collected along with a small history.

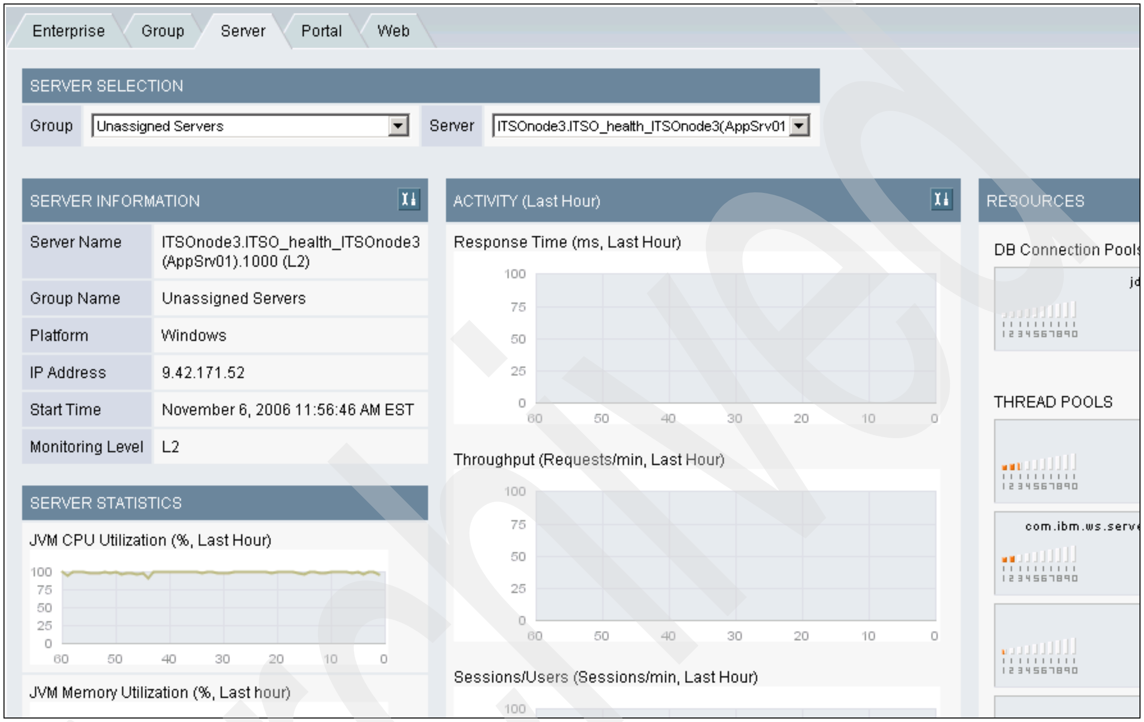


Figure 3-15 Server overview

There are four sections on the page: Server information, Activity, Resources, and Server statistics.

- Server information

  This area (Figure 3-16 on page 127) shows basic information about the server, including the IP address, server start time, and the monitoring level. Note that the default monitoring level for non-z/OS platforms is L2 (problem determination). For z/OS platforms, the default level is L1 (production mode). L2 may not be necessary unless you are addressing a problem.

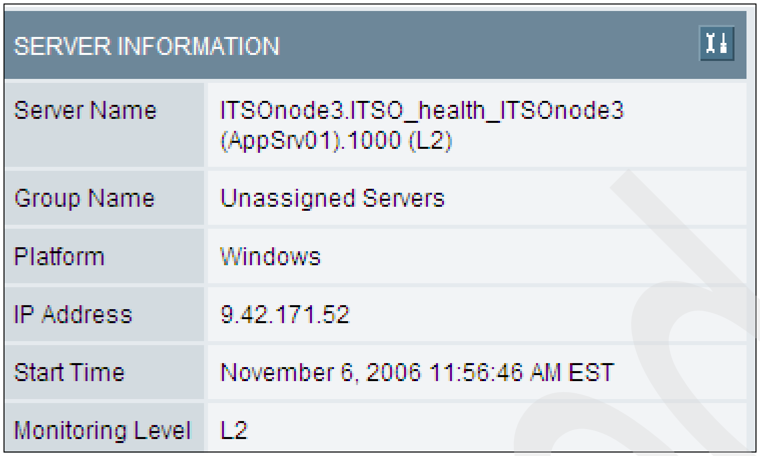


Figure 3-16 Server Information

- Activity

  This area (Figure 3-17) shows response time and throughput indicators.

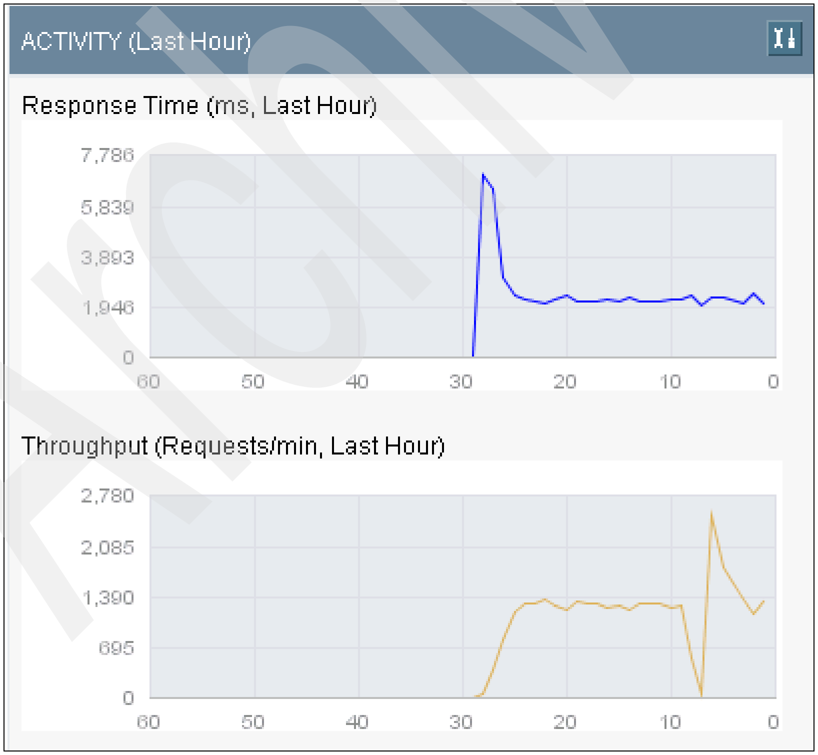


Figure 3-17 Activity

- Resources

  This area (Figure 3-18) shows information about resources in use by the application server.

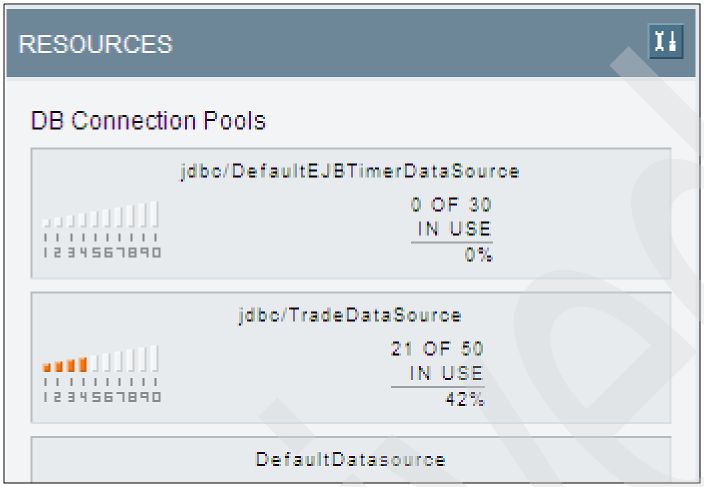


Figure 3-18 Resources

- Server statistics

  This area (Figure 3-19 on page 129) shows CPU and memory utilization information.

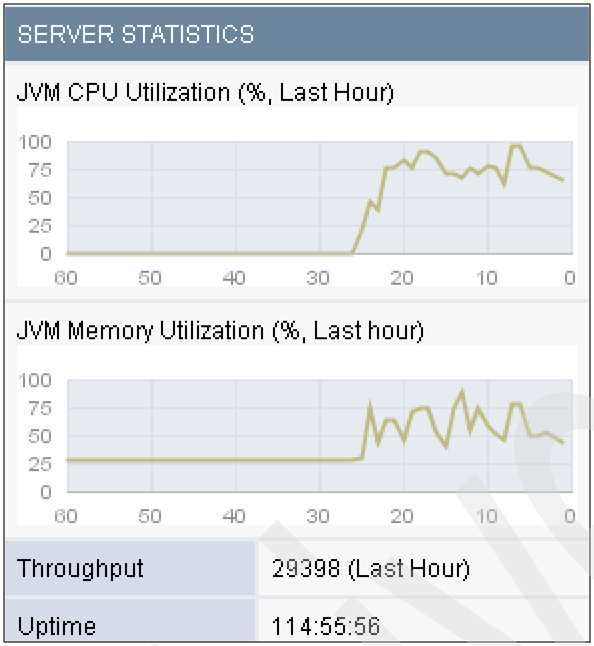


Figure 3-19 Server statistics

3.5.5 Troubleshooting scenario with ITCAM

Let us assume that after following the process outlined in section 3.4, “Health management example” on page 113 we still have poor response time and hung JVM conditions. The thread dump was not sufficient to debug the problem. But now we installed and configured ITCAM for WebSphere and installed the data collector on the application servers.

We will continue our problem determination as follows:

1. The first step is to increase the monitoring level in ITCAM for WebSphere to L2. The monitoring level determines the amount of data to collect. Setting it to L2 refines the granularity of the transaction capture so that J2EE component calls (JMS, JDBC™ etc.) are included.

   We also increase the sampling rate from 1 to 10% to increase the percentage of transaction instances saved to the database. This additional data may be needed to find the failed transaction.

   You can change these settings by going to **Administration** → **Managing Server** → **System Properties**.

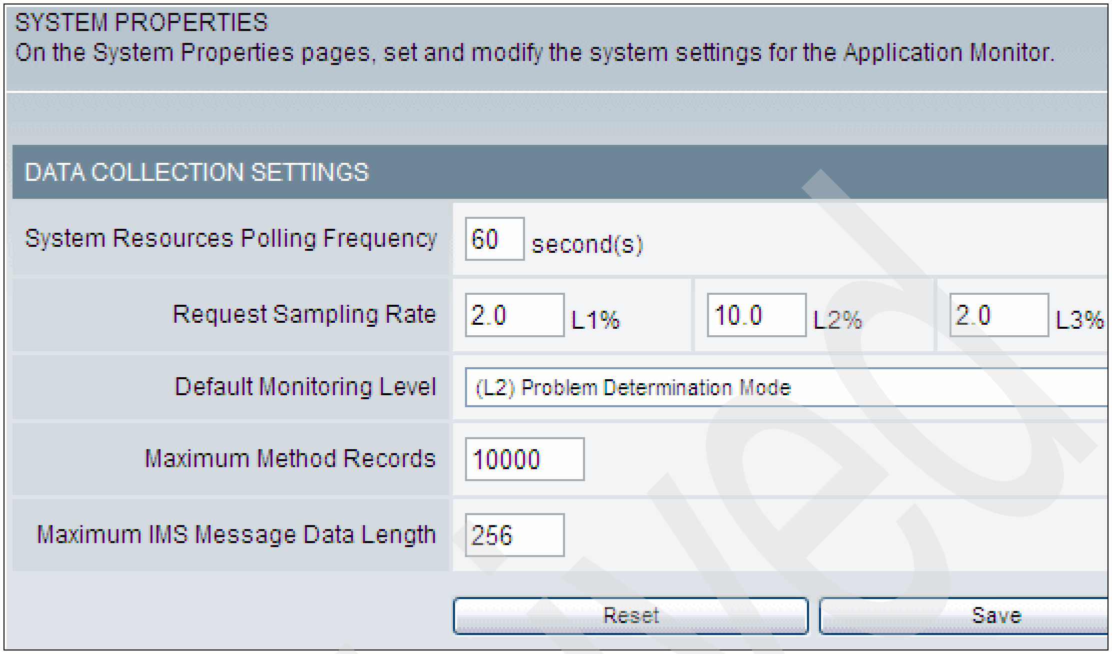


Figure 3-20 System properties

2. The administrator spot checks the Server Overview page for possible problems during the day but cannot sit in front of the console all day. Support has decided to let the agent collect the data during the hours when the problem is most likely to occur. They use the performance analysis features to analyze the data.

   After they are sure the problem occurred, they create a report that summarizes the data. The report is created by going to **Performance Analysis** → **Create Application Reports** → **Request/Transactions** in the AM console.

   The report can be generated according to your specifications. They choose the response time metric and specify that all request types be included. They specify the time range for the data to use. They also specify the server where they saw the problem occurring. The reporting facility also provides advanced filtering and report comparison options.

3. In the resulting report (Figure 3-21 on page 131), it is clearly visible that some of the requests are taking more time then others.

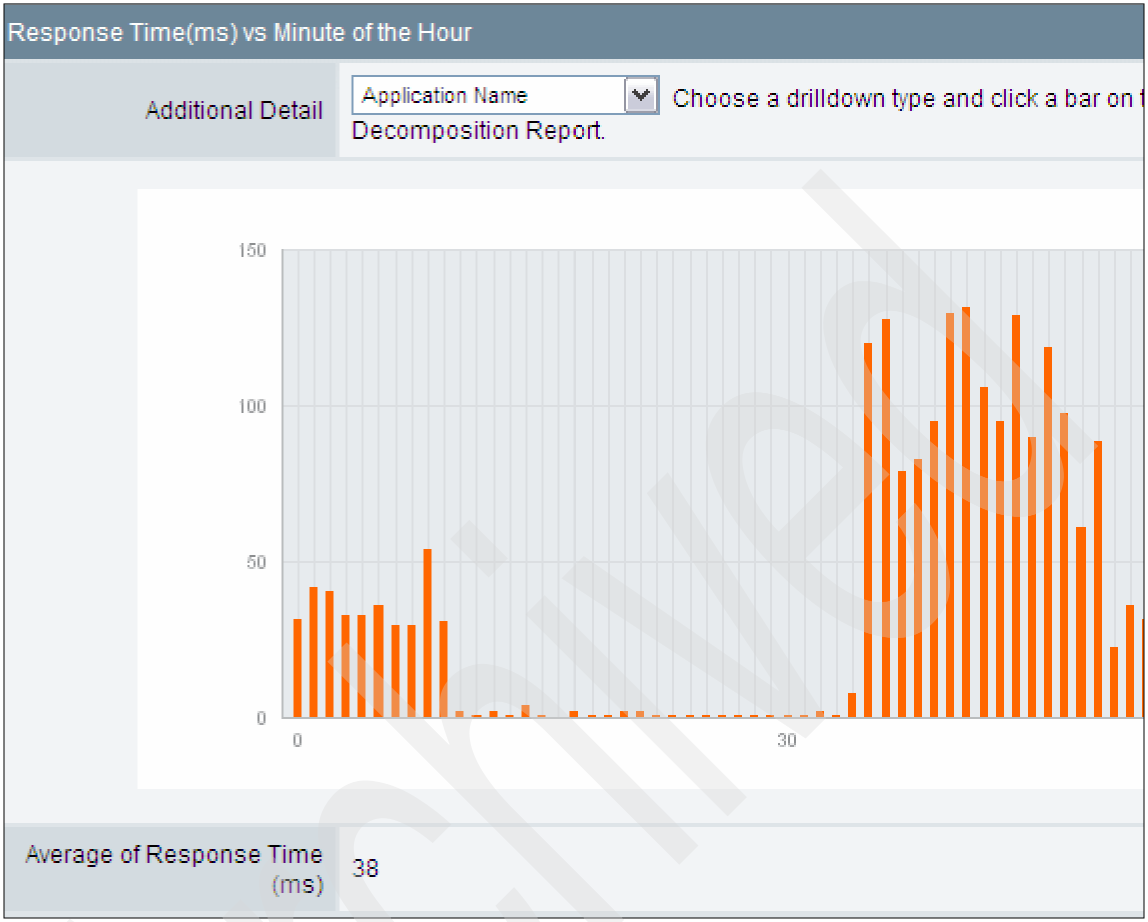
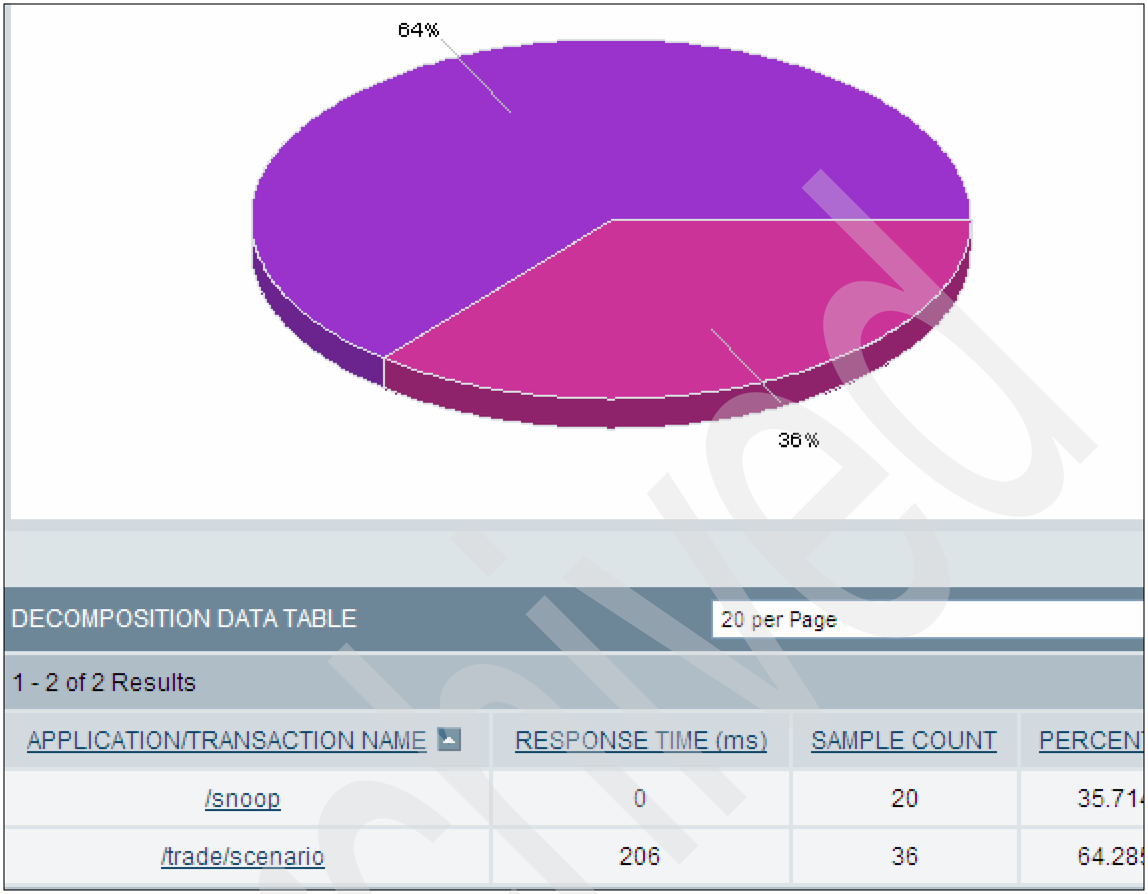


Figure 3-21 Response Time (ms) Vs Minute of the hour

4. Click on a bar in the graph that represents a long response time to view the decomposition report for that instance (Figure 3-22 on page 132).



APPLICATION/TRANSACTION NAME	RESPONSE TIME (ms)	SAMPLE COUNT	PERCEN
/snoop	0	20	35.71
/trade/scenario	206	36	64.28

Figure 3-22 Decomposition chart

The chart shows the response time for the transactions relative to the others. This report makes it apparent that the /trade/scenario transaction is taking a long time.

5. Click the link for /trade/scenario to see a detailed report of those transactions.

DETAIL REPORTS DATA TABLE

1 - 20 of 1000 Results

REQUEST/TRANSACTION NAME	REQUEST/TRANSACTION TYPE	RESPONSE TIME (ms)	CPU TIME (ms)	SERVER N
/trade/scenario	Servlet	49341	60.086	kcgl6hcNode0 (AppSrv01
/trade/scenario	Servlet	38125	40.058	kcgl6hcNode0 (AppSrv01
/trade/scenario	Servlet	13209	30.043	kcgl6hcNode0 (AppSrv01
/trade/scenario	Servlet	12889	0.000	kcgl6hcNode0 (AppSrv01

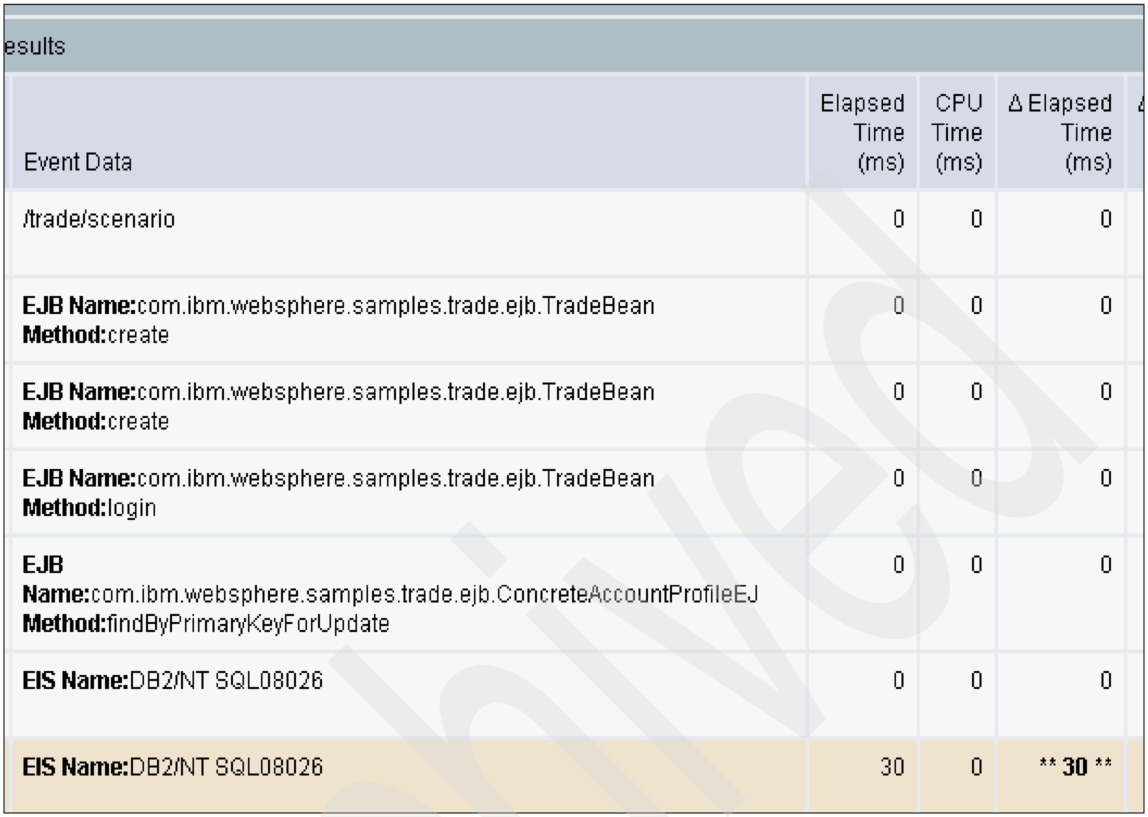
Figure 3-23   Detail transaction report

6. To see the details of a specific transaction, click the link for that transaction. In this case, select one with the highest response time. The resulting report shows the ten slowest components, as seen in Figure 3-24 on page 134.

10 SLOWEST COMPONENTS			
Rank	Depth	Event Type	Event Data
1	0	Servlet	/trade/scenario
2	1	EJB	**EJB Name:**com.ibm.websphere.samples.trade.ejb.TradeBean **Method:**getHoldings
3	4	JCA	**EIS Name:**DB2/NT SQL08026
4	3	EJB	**EJB Name:**com.ibm.websphere.samples.trade.ejb.ConcreteQuoteEJB_7 **Method:**PMInternalFinder_Local
5	2	EJB	**EJB Name:**com.ibm.websphere.samples.trade.ejb.ConcreteHoldingEJB **Method:**getDataBean
6	1	EJB	**EJB Name:**com.ibm.websphere.samples.trade.ejb.TradeBean **Method:**login
7	1	EJB	**EJB Name:**com.ibm.websphere.samples.trade.ejb.TradeBean **Method:**getAccountData

Figure 3-24 Slowest Component

In this same report, you can switch to a Flow view that might be helpful in understanding the sequence of events and where things went wrong (Figure 3-25 on page 135).



esults

Event Data	Elapsed Time (ms)	CPU Time (ms)	Δ Elapsed Time (ms)
/trade/scenario	0	0	0
EJB Name:com.ibm.websphere.samples.trade.ejb.TradeBean **Method:**create	0	0	0
EJB Name:com.ibm.websphere.samples.trade.ejb.TradeBean **Method:**create	0	0	0
EJB Name:com.ibm.websphere.samples.trade.ejb.TradeBean **Method:**login	0	0	0
EJB Name:com.ibm.websphere.samples.trade.ejb.ConcreteAccountProfileEJ **Method:**findByPrimaryKeyForUpdate	0	0	0
EIS Name:DB2/NT SQL08026	0	0	0
EIS Name:DB2/NT SQL08026	30	0	** 30 **

Figure 3-25 Flow view

7. Once the problem is narrowed down to a component, the support team shares the information with the application developer.
8. The developer makes the fix and after testing the code sends the updated EAR to be deployed in production. The Application Edition Manager is used to install the new application in the production environment.
9. Once the system is stable, the support team resets the system back to L1 and 1% sampling to conserve platform resources and to avoid possible performance issues. They also disable health management.

Part 2

Long running application extenders

The chapters in this section discuss how to use WebSphere Extended Deployment to effectively run batch and compute intensive applications in a J2EE environment. It contains the following chapters:

4

Introduction to long-running applications

This chapter introduces you to the WebSphere Extended Deployment long-running environment and long-running applications. The contents of this chapter include the following:

- What are long-running applications?
- Why use long-running applications?
- Long-running environment concepts
- Managing long-running jobs
- High availability in a long-running environment
- Setting operational policies for long-running applications

4.1 What are long-running applications?

Long-running workloads or applications typically require more resources and different types of support from a runtime environment than the standard lightweight, transactional work that is typical of today's J2EE applications. WebSphere Extended Deployment V6.0 introduces a facility to the J2EE environment for supporting these long-running applications. The term *business grid* is often used to describe this environment and the overall support required for long-running applications. The business grid provides the capability to deploy different types of applications to different nodes within a WebSphere cell and to balance the work based on policy information.

The submission of a long-running workload, also known as *job*, is asynchronous from the workload being executed. The separation of the submission and execution also allows for the submission of work from outside the WebSphere environment. Once long-running work begins, state information needs to be persisted to a highly-available data store. Administrators require the ability to monitor and manage long-running work. The environment needs to be able to schedule and prioritize the work based on service policy information set by the user.

A number of these functions can be provided within an existing WebSphere J2EE environment by utilizing message driven beans (MDBs). Some WebSphere customers have successfully exploited this. However, the business grid provides an enhanced environment for long-running applications, making it easier to support this type of work within WebSphere.

The business grid components support two types of long-running workloads or applications: *batch* and *compute-intensive*.

A typical batch application does large amounts of work based on repetitive tasks. A batch application needs to provide the logic for a single unit of work, and the container provides the support to run the job with transactions and the ability to checkpoint and restart the application as required. For example, a typical batch application would process a large number of records. Each record can represent a unit of work. The application provides the logic to process a single record. The environment manages the process of repeatedly invoking the application's task for processing each record until complete and performs the appropriate transaction when required.

Compute-intensive applications perform work that requires large amounts of system resources, in particular CPU and memory. In this case the application provides all the logic for performing the work including acquiring the resources. The business grid makes sure that the application is appropriately situated within the environment.

4.2 Why use long-running applications?

Reasons to use WebSphere Extended Deployment for long-running applications

- You want to reuse existing J2EE business logic in batch applications.
- You want to develop new applications that have a batch or compute-intensive characteristics and your company has a wealth of Java-skilled programmers.

Most of the batch workloads running today are written in COBOL and run on z/OS. These workloads contain complex business logic such as interest calculations and process very large amounts of data, such as credit card postings and insurance claims. Customers have very tight windows to do these operations.

Even with the emergence of Java as a popular and powerful programming language, Java is viewed as not fast enough for traditional batch workloads, and more importantly, Java does not have access to many of the z/OS resources used in a typical batch job. With today's fast maturing WebSphere Application Server container server services, Java fits very well into the online transaction processing (OLTP) paradigm.

Why would you want to implement a Java batch environment on WebSphere? To answer this we need to probe the following areas:

- Who will be developing modern batch applications?
- Which run-time environment is best positioned to host Java batch applications?
- What does WebSphere provide to make it an attractive environment to develop and deploy Java batch style applications?

4.2.1 Java and J2EE skills

In today's development environment Java skills are readily available, and in general it is comparatively less expensive to achieve an equivalent level of application development competency with Java as with older languages. The object oriented nature of the language enables rapid development cycles. There is a large portfolio of powerful Java development tools from ISVs. JVM performance is continually increasing. This marks Java as the language of choice for the development of new business applications or a target language for the re-engineering of existing applications.

Additionally, the J2EE standard as implemented by today's application server products has emerged as a very popular model and environment for developing and deploying modern, enterprise-class business applications, especially OLTP applications. As a result, the J2EE skill base is rapidly growing in the industry. This makes the Java language and J2EE application servers an interesting and obvious target to host other types of enterprise business workloads, such as batch and compute-intensive applications.

4.2.2 The J2EE run-time environment

Today you can deploy Java batch applications on z/OS in a traditional batch initiator environment. There are some disadvantages. A JVM is created for every batch job. JVM creation is the most expensive operation in the Java world. This increases the CPU usage, consumes additional resources, can decrease the throughput for a given batch window, and hence increase the overall cost of implementing a Java batch application as opposed to implementing the same application in a more traditional language. You can also deploy a Java batch style application in a Unix System Services shell. However there are no facilities to start, stop, and manage batch work. You have to design and develop all of it.

The J2EE environment as implemented by the WebSphere Application Server provides a natural home for deploying enterprise-class business applications written in Java. First, each application server process is a JVM and within that JVM WebSphere provides for the instantiation, management, and termination of the Java business applications. Second, WebSphere provides many of the overall system services, also known as container services, required by complex business applications. Some typical container services are security, transaction support, high availability features, Web services, session pooling, connection pooling, caching, and message buses, just to name a few. All this leads to drastically simplifying the Java application's design, accelerates development cycles, simplifies management, and increases the application's overall security, availability, performance, and stability.

4.2.3 Re-use of existing J2EE business logics

The object-oriented nature of the Java language enables existing business logic to be reused, leading to rapid development cycles. Even though batch applications are developed using the long running programming model of WebSphere Extended Deployment, core business logic can be shared with OLTP applications, helping to maintain existing applications.

4.2.4 WebSphere Extended Deployment services for batch

WebSphere Extended Deployment builds on top of the existing WebSphere J2EE programming model and container services by providing a job scheduler, an execution environment, and additional features specifically designed to provide for the execution and management of long-running batch applications. Just as WebSphere Application Server Network Deployment provides a more natural environment for hosting Java enterprise applications, specifically OLTP applications, Extended Deployment provides a more robust environment for deploying long-running Java applications.

4.3 Long-running environment concepts

Figure 4-1 provides a high level overview of the main components involved in the long-running environment or business grid. Each of these components are explained in more detail in the following sections.

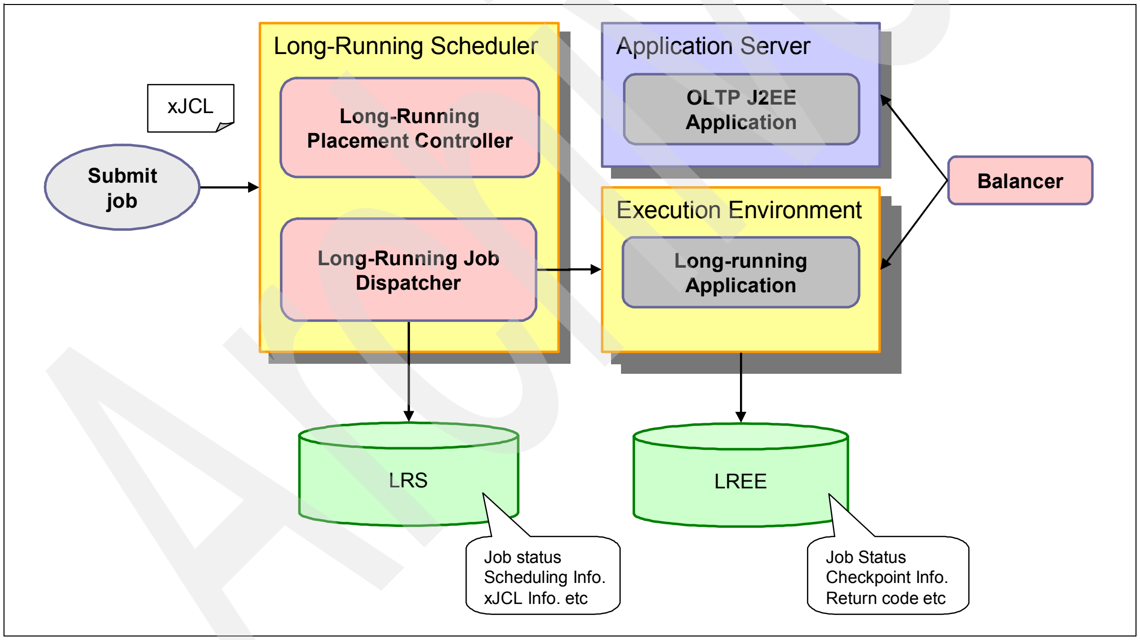


Figure 4-1 Main components in a long-running environment

4.3.1 Long-running execution environment

The long-running execution environment on WebSphere Extended Deployment provides the runtime resources required by the long-running applications.

Long-running applications are deployed into dynamic clusters that are configured to support long-running execution. The services to enable long-running applications are contained in a provided J2EE application called LREE.ear. This application is installed into a dynamic cluster.

A dynamic cluster configured to support long-running applications is referred to as the *long-running execution environment (LREE)*. A database gives the execution environment a transactional data store that tracks job status, does checkpoints, and recovers from failures.

Different long-running execution environment instances (for example, LREE configured dynamic clusters) can be configured in a cell to support different types or styles of long-running work. A particular instance of the execution environment can have its own dedicated database or it can share a database with one or more other instances of the execution environment. The process of configuring a long-running execution environment is described in detail in Chapter 5, "Configuring a long-running runtime environment" on page 163.

The LREE supports the two types of long-running workloads, batch and compute-intensive. To support the two types of long-running workloads, within the LREE.ear there are two types of long-running execution environments.

- The *compute-intensive execution environment* supports long-running applications that expect to consume large amounts of memory or CPU time. This execution environment provides a relatively simple programming model based on asynchronous beans.
- The *batch execution environment* supports long-running, batch-oriented applications. These applications are expected to repetitively perform record processing *units of work* similar to more traditional J2EE applications but are driven by batch inputs rather than by interactive users. This environment builds on familiar J2EE constructs, such as entity beans, to provide batch applications with a programming model that supports container-managed restartable processing and the ability to pause and cancel executing jobs.

From a configuration point-of-view, you can share the long-running execution environment since both execution environments are provided by the LREE application.

4.3.2 Long-running scheduler

The *long-running scheduler (LRS)* is responsible for managing the execution of long-running jobs. It accepts jobs when they are submitted, assigns job IDs, and persists their state in a database. It also selects where and when jobs should be run. The LRS has two key components: Job dispatcher and Endpoint selector.

- *Job dispatcher*: Job dispatcher accepts job submissions, assigns job IDs, persists jobs in job database, and sends jobs to execution environments.
- *Endpoint selector*: The endpoint selector uses service policies to select which jobs to run, where to run them, and when.

As with the dynamic operations function, the business grid has autonomic management functions to dynamically adapt the long-running environment to changing workloads.

The business grid provides the *Long-running placement controller* function. This autonomic controller is analogous to the application placement controller (APC) that starts and stops instances of transactional applications. This long-running placement controller runs within the LRS and starts and stops instances of long-running execution environments in response to the situations of the service policies.

The LRS services are provided by a J2EE application, LongRunningScheduler.ear. Only one instance of this application can run at a time.

4.3.3 Balancer

Due to the nature of the long running work, co-locating it on the same system with the transactional work may have a negative impact on the transactional work being performed. When the dynamic cluster for the transactional work and the dynamic cluster for the long running work are mapped to the same node group, the *balancer* makes decisions about when and on which node the transactional and long-running work should run, communicating with long running placement controller and application placement controller as shown in Figure 4-2 on page 146. These decisions are based on a number of factors including how well the service policy goals for the two types of work are being met. If there is more work than the system can handle, the balancer uses service policy importance settings to determine which service policies it tries the hardest to achieve.

System resources are assigned to long-running work or transaction work by node, not by application server instance. The balancer gives control of the node to the application placement controller or long running placement controller, which then determines how many instances should run. The balancer can switch a node between long-running and transactional work over time, but will never attempt to automatically start both types of work on a node concurrently.

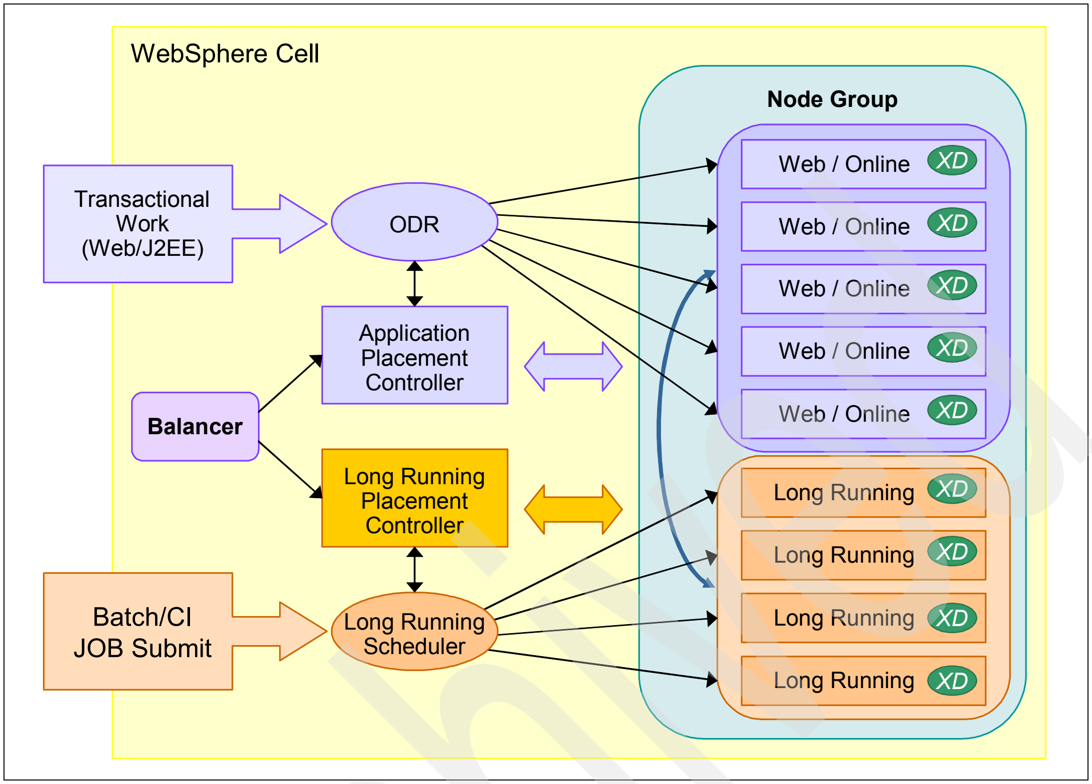


Figure 4-2 Balancer coordinates the transactional work and the long running work

The balancer and application placement controller run in an instance that the HA manager manages, while the long running placement controller runs within a long-running scheduler. The scope of the components are at the cell level. The runtime topology chart available in the administrative console shows the location of the HA managed components and the type of work the node is assigned to, as shown in Figure 4-3 on page 147.

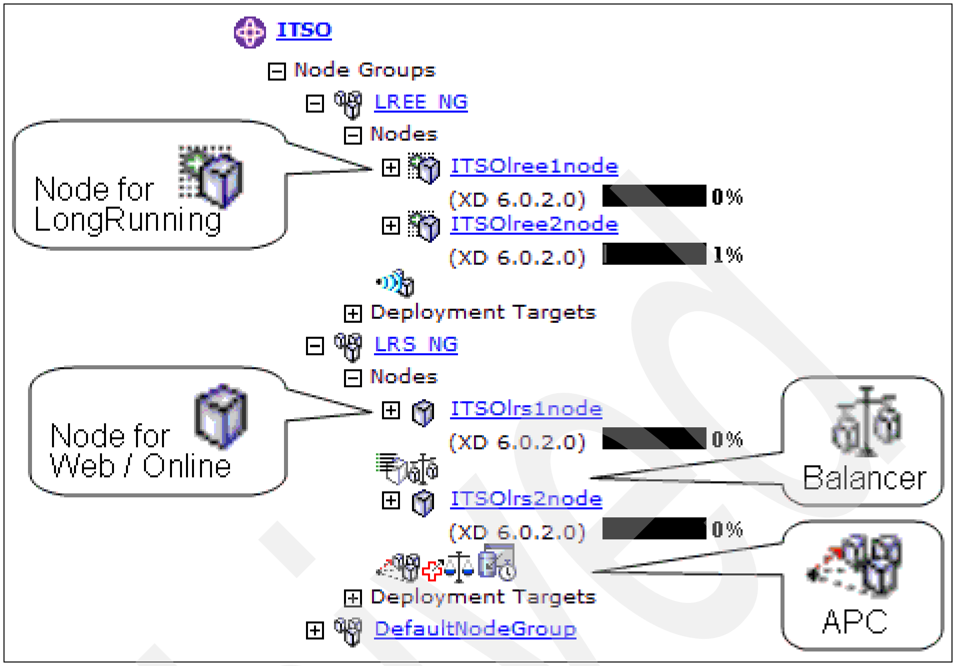


Figure 4-3 Runtime topology

4.4 Managing long-running jobs

This section discusses job management tasks including defining and submitting long-running jobs.

4.4.1 Job management interfaces

The long-running scheduler is responsible for job management tasks, such as submitting a job, checking the status of the job, cancelling the job, and so on. It manages the LRS database, passes the job ID, and selects where and when the job is executed.

The following interfaces to the LRS are available for submitting long-running workloads or jobs (Figure 4-4 on page 148).

- Command line (`lrcmd.bat/sh`)
- Administrative console
- EJB interfaces
- Web service interfaces

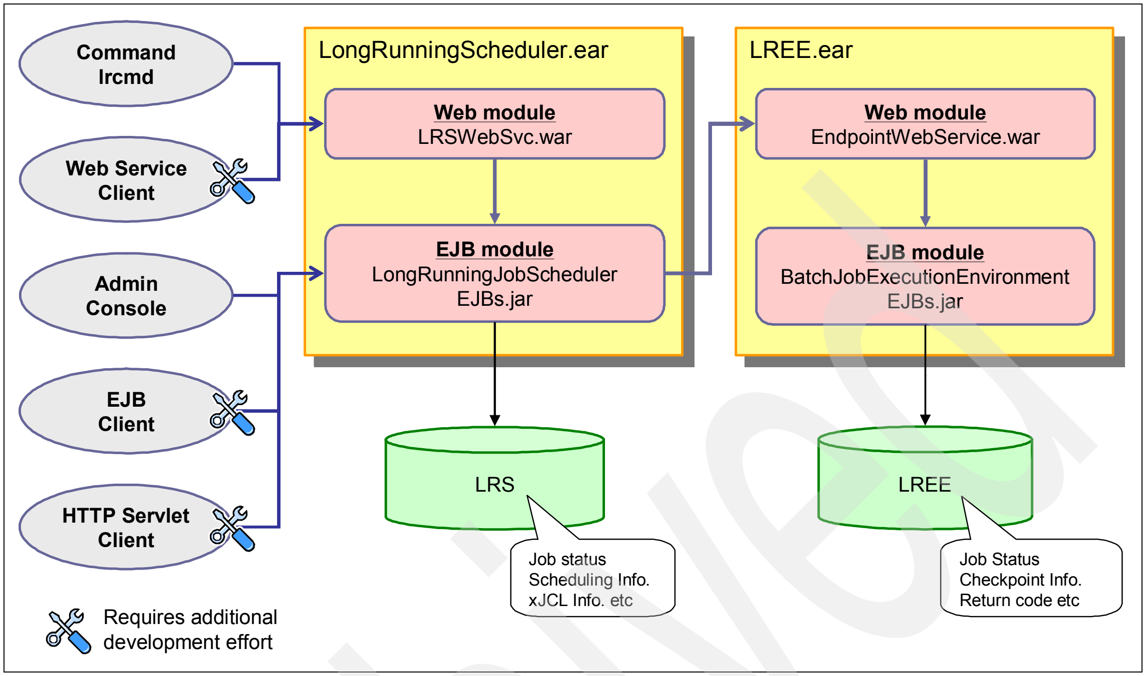


Figure 4-4 Long-running job management interfaces

Command line

A command line interface to the long-running scheduler is provided to submit and manipulate long running jobs. The command is called **`lrcmd.bat`** or **`lrcmd.sh`**, and resides in the bin directory of WebSphere Extended Deployment.The **`lrcmd`** command makes Web services calls to the long-running scheduler application, and can be run from any machine with Extended Deployment installed and network access to the long-running scheduler application servers.

Following is the sample usage of this command.

1. To submit a job:

   Specify `submit` for the cmd option. Specify the location of an xJCL that defines the job and the port and host name of the long-running scheduler.

   ```
   lrcmd.bat -cmd=submit -xJCL=<path_to_xJCL> -host=<host> -port=<port>
   ```

2. To check the status of the job:

   Specify `status` for the cmd option and the job ID of the job to be checked. If you do not specify the jobid, the statuses of all jobs persisted in LRS database are listed.

   ```
   lrcmd.bat -cmd=status -jobid=<jobid> -host=<host> -port=<port>
   ```

3. To show the explanation of a command:

   Specify `help` for the cmd option.

   `lrcmd.bat -cmd=help`

For detailed information about **`lrcmd`**, see the following Web page:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp?topic=/com.ibm.websphere.xd.doc/info/scheduler/cbgcommd.html

> **Note:** The **`lrcmd`** command line tool starts a separate JVM each time it is launched and hence typically consumes quite a bit of resources. You might want to explore more effective ways of submitting and checking the status of jobs, such as the EJB and Web service interfaces.

Administrative console

From the administrative console, you can check the status or cancel a job. You can find the job management page by navigating **Runtime Operations** → **Job Management** as shown in Figure 4-5.

Select	Action	Job ID	Submitter	Last Update	State	Node
☐	Remove	SimpleCIEar:34		Mon Nov 06 12:45:30 EST 2006	Ended	ITSOlree1r
☐	Remove	SimpleCIEar:35		Mon Nov 06 12:45:57 EST 2006	Ended	ITSOlree1r

Figure 4-5 Job Management in the Administrative console

In Extended Deployment V6.0, these job management functions are limited, for example, you cannot submit a job from the administrative console or get the return code of a job.

EJB interface

The long-running scheduler provides an EJB interface for J2EE applications to programmatically submit and manipulate long-running jobs. Suppose you need to submit and manage a large number of jobs in the production environment. The **`lrcmd`** command starts a JVM process each time you execute the command; therefore, consuming resources. An alternative is to develop a Web application using the JobScheduler remote interface for the long-running scheduler EJB to submit and manage your jobs.

Example 4-1 shows a sample servlet that submits a job using the EJB interface. In order to submit a job from this servlet, the xJCL needs to be accessible to the server running this Web application.

Example 4-1 Sample servlet code using EJB interface to submit a job

```
package com.ibm.itso.sample.batch.client;
import java.io.*;
import java.rmi.RemoteException;
import javax.naming.InitialContext;
import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.websphere.longrun.JobScheduler;
import com.ibm.websphere.longrun.JobSchedulerHome;
import com.ibm.websphere.longrun.SchedulerException;
import com.ibm.websphere.longrun.JCLException;
import com.ibm.websphere.longrun.JobSubmissionException;

public class JobSubmitServlet extends HttpServlet implements Servlet {
   private String jndiNameforScheduler =
"ejb/com/ibm/websphere/longrun/JobSchedulerHome";
   private JobScheduler js;

   public void init(ServletConfig arg0) throws ServletException {

      try{
         InitialContext ictx = new InitialContext();
         JobSchedulerHome jsHome =
(JobSchedulerHome)ictx.lookup("cell/clusters/LRS_DC/"+jndiNameforScheduler);
         js = jsHome.create();
      }catch(javax.naming.NamingException ne){
         System.out.println(" NamingException occurred. Failed to Lookup
JobSchedulerHome."+ne);
      }catch(javax.ejb.CreateException ce){
         System.out.println(" CreateException occurred. Failed to Create
JobScheduler. "+ce);
      }catch(RemoteException re){
         System.out.println(" RemoteException occured. Failed to Create JobScheduler.
"+re);
      }
   }

   protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
```

```
      //Get the xJCL file path from request parameter.
      File xJCLfile = new File(req.getParameter("xJCLpath"));

        //Read the xJCL file until the end
      FileReader fr = new FileReader(xJCLfile);
        BufferedReader br = new BufferedReader(fr);
        String xJCL="";
        String line;
        while ((line = br.readLine()) !=null ){
        xJCL = xJCL+line;
        }
        //Submit the job
        try{
      String jobid = js.submitJob(xJCL);
      System.out.println("Job is submitted. JobID is "+jobid);
        }catch(SchedulerException se){
        System.out.println(" SchedulerException occured."+se);
        }catch(JCLException je){
        System.out.println(" JCLException occurred. Failed to parse the xJCL file.
"+je);
        }catch(JobSubmissionException jse){
        System.out.println(" JobSubmissonException occured. Failed to Submit a Job.
"+jse);
        }
   }
}
```

Additional information about the other methods defined in the JobScheduler interface can be found in the WebSphere Extended Deployment Information Center at the following Web address:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/topic/com.ibm.websphere.xd.doc/info/reference/apidocs/com.ibm.websphere.longrun.JobScheduler.html

Web service interface

The LRS also supports programmatic access to its functions over a Web service interface for both J2EE and non-J2EE applications. This also enables jobs to be scheduled from outside a WebSphere environment. Internally the long-running scheduler Web service interface calls the EJB interface methods.

4.4.2 Defining the long-running jobs

A *job* is the central concept to all long-running applications and represents an individual unit of work to be executed. A job of batch applications can consist of one or more job steps while a job of compute-intensive applications can consist of only one job step.

xJCL

Every long-running job is defined using an XML-based file called *xJCL* (XML Job Control Language). The xJCL has constructs for expressing all of the information needed for both types of long-running workload (job) execution; although, some elements of xJCL are only applicable to either compute-intensive or batch type jobs.

> **Note:** The xJCL definition of a job is not part of the actual long-running application. It is constructed separately and submitted to the long-running scheduler for execution. The long-running scheduler uses information in the xJCL to determine where, when, and how the job should be run.

Before you submit jobs to the long-running scheduler, you need to prepare the xJCL files to define the necessary information for job execution. For more detailed information about xJCL, please refer to section 7.7, "Batch application syntax in xJCL" on page 279 and section 8.4, "Compute-intensive application syntax in xJCL" on page 298.

4.4.3 Long-running application flow

In this section we look at how the long-running environment components work together to run a long-running application. The flow is illustrated in Figure 4-6 on page 153.

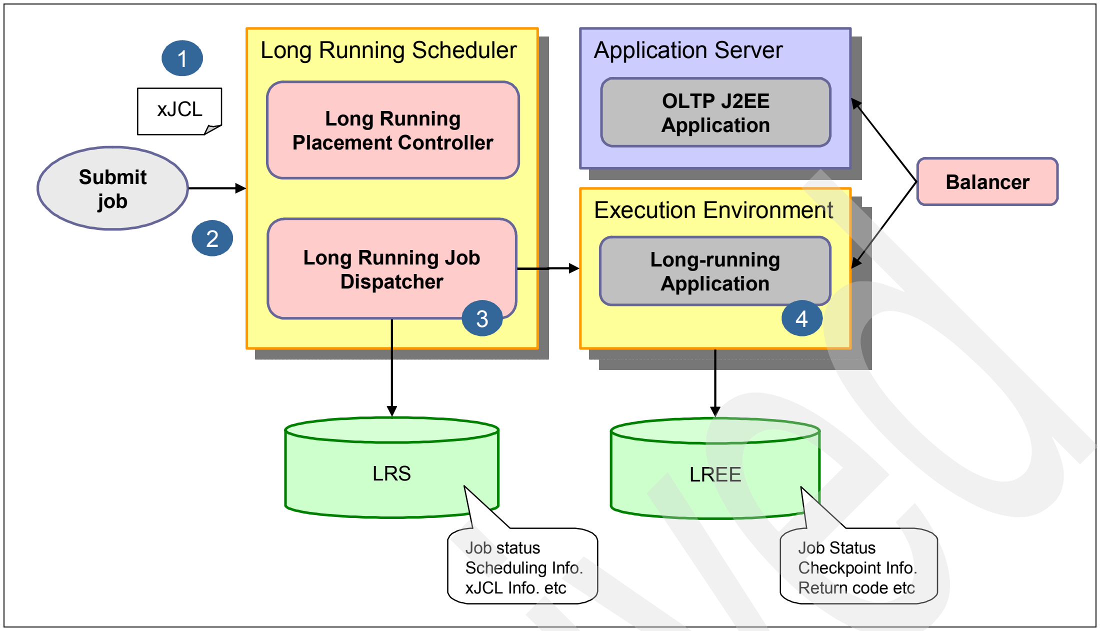


Figure 4-6 Long-running application flow

1. The xJCL describing the information needed to run the long-running job is prepared.
2. The xJCL is submitted to the LRS via the command line, EJB interface, Web service interface, or the administrator console.
3. The LRS accepts the job submission and dispatches the job to an LREE.
   a. If WebSphere security is active, the user is authorized against the long-running scheduler security roles.
   b. A job ID is assigned to the submitted xJCL request.
   c. The xJCL passed in the request is validated and stored in the LRS database.
   d. The job is classified based on several factors including service policies (priority level and max queuing time).
   e. The job is queued (in memory) and is dispatched for execution to an LREE via the Web service interface. The LRS endpoint selector uses several factors to determine which LREE to dispatch to it. LRS can start a new LREE instance if required (depending on the configuration, load, and so forth).
4. The LREE executes the job as described in the xJCL. While processing the job, the LREE persists and updates the information about the job, such as

state and checkpoint information, to the LREE database. The job state is also communicated by the LREE to the LRS for storage in the LRS database. When the job completes, the final status is sent to the LRS.

Note: Checkpoints are only used in batch applications.

4.4.4 Lifecycle of a job

While a job is submitted to the LRS and processed in an LREE, the job transits to various job states. When the job state is changed during the processing, the updated state is persisted in the LRS database. The current state of a batch job can be viewed from the administrative console job management panels or retrieved using the LRS command line, EJB, or Web service interfaces.

Lifecycle of a batch Job

Figure 4-7 on page 155 shows the relationship between the states of a batch job and the events that trigger the transition of the states.

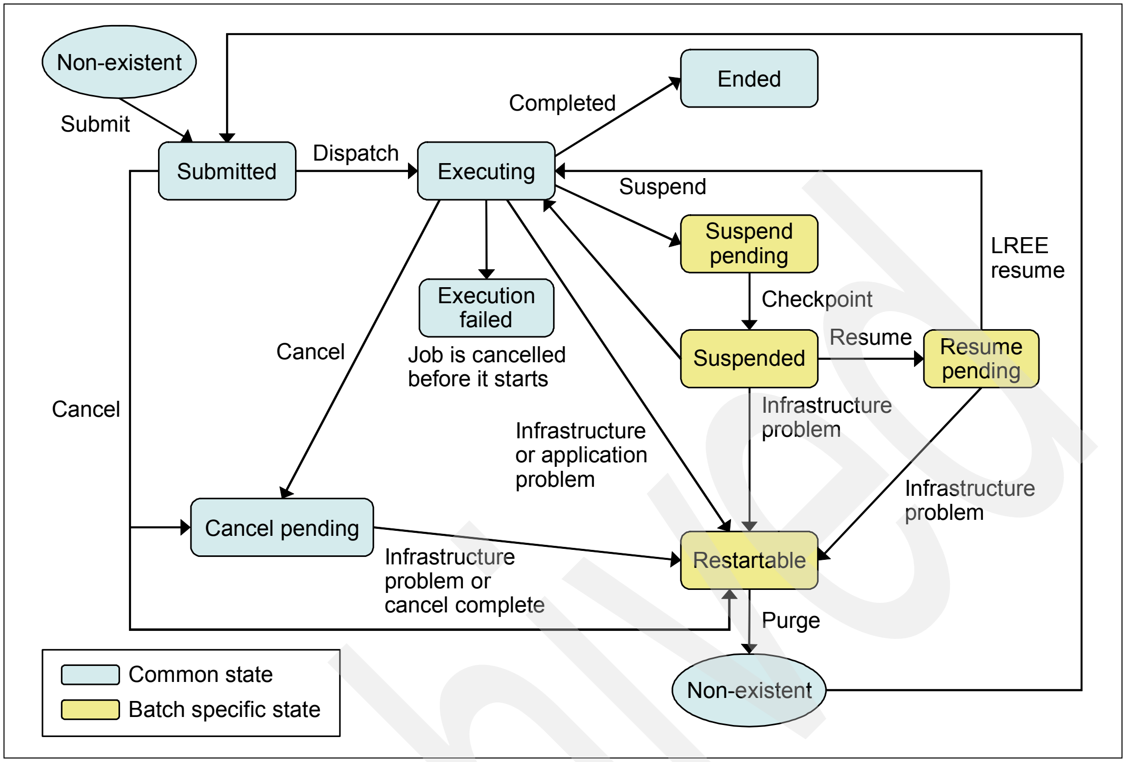


Figure 4-7 Lifecycle of a batch job

- When a job is submitted, the job state of the job is “Submitted”.
- When the job is dispatched to a LREE, the job state changes to “Executing”.
- If a failure occurs before a batch step initializes, the batch job goes into an “Execution failed” state and cannot be restarted.
- If the job fails or is cancelled after the initialization phase, the job state will be “Restartable”, which allows the job to be restarted.
- If the job steps complete without failure, the job state will be “Ended”.

Some states such as Restartable and Suspended are specific to batch jobs.

Life cycle of compute-intensive job

Figure 4-8 on page 156 shows the relationship between the states of a compute-intensive job and the events that trigger the transition of states.

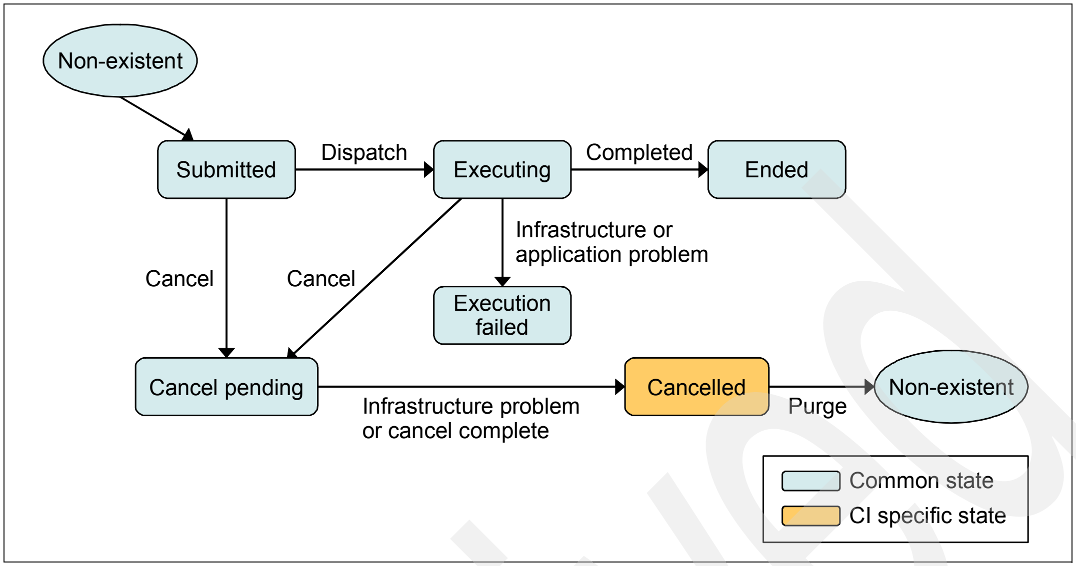


Figure 4-8 Life cycle of a compute-intensive job

- When a job is submitted, the job state of the job is “Submitted”.
- When the job is dispatched to a LREE, the job state changes to “Executing”.
- If a failure occurs, the job goes into “Execution failed” state.
- If the job completes without failure, the job state will be “Ended”.
- Since compute-intensive jobs cannot be restarted, if the job is cancelled the job state will be “Cancelled” instead of “Restartable”.

For more information, see the following Web page in the WebSphere Extended Deployment Information Center:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp?topic=/com.ibm.websphere.xd.doc/info/odoe_task/codhealthbgrid.html

4.5 High availability in a long-running environment

Figure 4-9 on page 157 shows an example of a highly-available configuration of a long-running environment.

The LRS is deployed in a dynamic cluster (LRS_DC in the figure) for high-availability reasons, though only one instance of the long-running scheduler can run at a time. To ensure that only a single LRS application server runs at a time, set both minimum and maximum instances in the dynamic cluster to 1.

To enhance availability, dispatchers and ODRs can be placed in front of the LRSs. This enables you to send a request without being aware of which LRS receives it. When you send requests to submit or control jobs from the Web service or command line interfaces, the dispatcher and ODR will route the request to the active LRS. When you send a request to submit or control a job from an EJB interface, the Object Request Broker (ORB) routes the request to the active LRS.

The LREEs are also deployed in a dynamic cluster (LREE_DC in the figure). The LRS dispatches jobs to the available LREE application server. The long running placement controller in the LRS can start additional LREE instances on demand if service policy goals are not satisfied.

You can set the maximum number of the jobs running concurrently on an LREE instance by setting the **`com.ibm.websphere.longrun.EndPointCapacity`** custom property for the dynamic cluster.

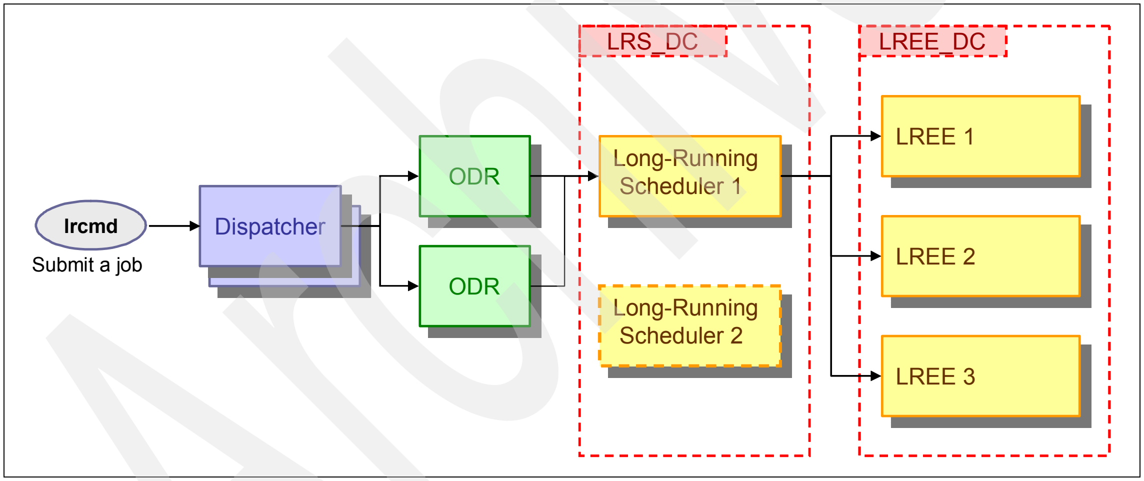


Figure 4-9 Configuration of long-running environment for high availability

4.6 Setting operational policies for long-running applications

Operational policies can be set for long running applications.

4.6.1 Configuring service policies for long-running applications

Service policies for long-running applications use the *Queue Time* goal. WebSphere Extended Deployment judges whether it meets the service level by how long jobs are queued in the long-running scheduler before being dispatched to the LREE. This type of service policy can be used for the following purposes.

- Start LREE instances on demand

  When jobs are queued up in the LRS waiting for an LREE to become available and the queue time of the waiting jobs is over the limit specified by the service class to which the jobs are related, the long running placement controller will start additional LREE instances to process the queued jobs and to meet the service level.

  The LREE dynamic cluster must be set to automatic or to supervise mode. Note that the LRS will not start any LREE instances over the Maximum Instance setting of the LREE dynamic cluster.

- Process important jobs by priority

  Suppose you have two long running applications. One is very important and must be processed in as short a time as possible but its jobs do not take a long time. The other is less important and its jobs do not need to process in as quick a time.

  You can create a service policy for each application. The service policy for the important application is set to have the “Highest” importance with a rather short time goal value, while the service policy for the second application is set to have “Low” importance and a long time goal value.

  Suppose the jobs of the low importance application are queued in the LRS when the job of high importance application is submitted. If the queue time of the high importance application is over the service level goal, the LRS prioritizes and dispatches it, keeping the low importance job waiting. Note that the higher importance job is not always dispatched first. The LRS calculates the balance of the jobs and works to meet the service classes as a whole.

Creating the service policy and transaction class

Use the following steps to create a service policy and transaction class.

1. In the administrative console, navigate to **Operational Policies** → **Service Policies**, and click **New**.
2. In step 1, specify a name for this service policy, for example LongRunning_Policy, and select **Queue Time** as a Goal Type, as shown in Figure 4-10 on page 159.

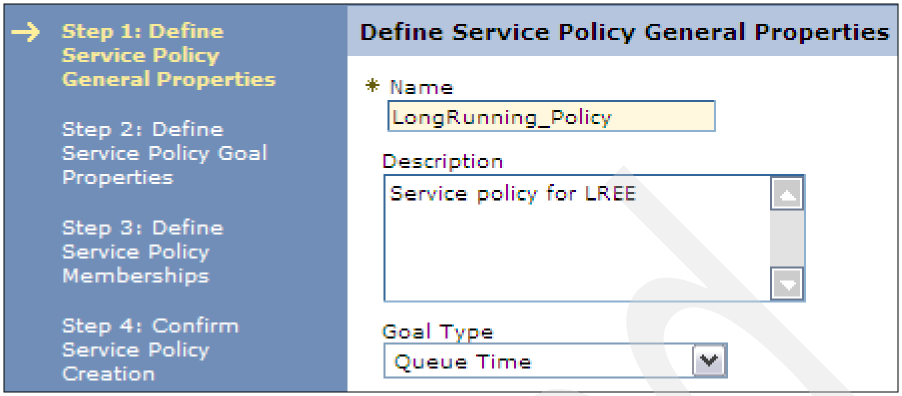


Figure 4-10 Define the service policy general properties

3. In step 2, specify the goal and importance of this service policy. Here we specify `1 Minutes` as the goal value and `Highest` as the importance as shown in Figure 4-11.

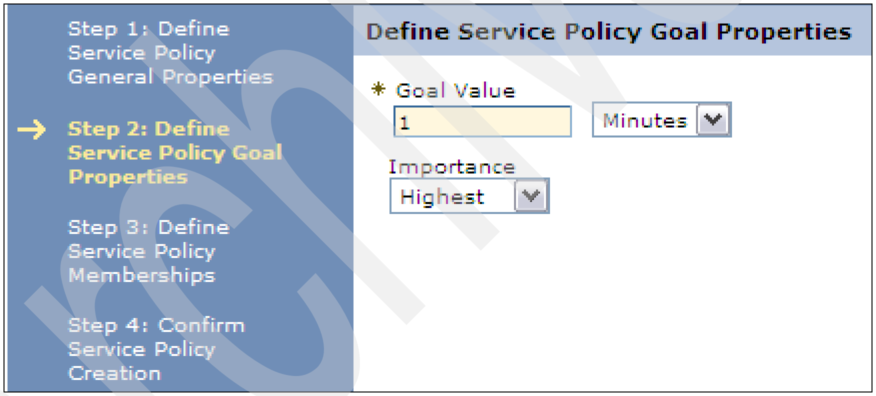


Figure 4-11 Define service policy goal properties

4. In step 3, click **New** to create a new transaction class.
   a. Specify a name for the transaction class, for example `LongRunning_TC`, and click **Next** to proceed.
   b. Check the summary page, and click **Finish**.
5. Confirm that the transaction is created, and click **Next**.
6. In step 4, check the summary, and click **Finish**.
7. Save your change to the master configuration.

Relate the service policy to the long running application

Use the following steps to relate the new policy to a long running application. In our example, it is related to the SimpleCI.ear sample application.

1. Navigate to **Applications** → **Enterprise Applications**.
2. Click the application to which you want the service policies to relate. In this case, SimpleCIEar.
3. Move to the **Service Policy** tab, and expand the **Work Classes for IIOP requests**.
   a. Under “If IIOP request matches”, you can see the controller bean of the SimpleCI.ear (com.ibm.ws.ci.CIControllerBean) is specified as the IIOP pattern.
   b. Under “If no classification rules apply, then classify to this transaction class”, select the transaction class created in the previous step as shown in Figure 4-12 on page 161.

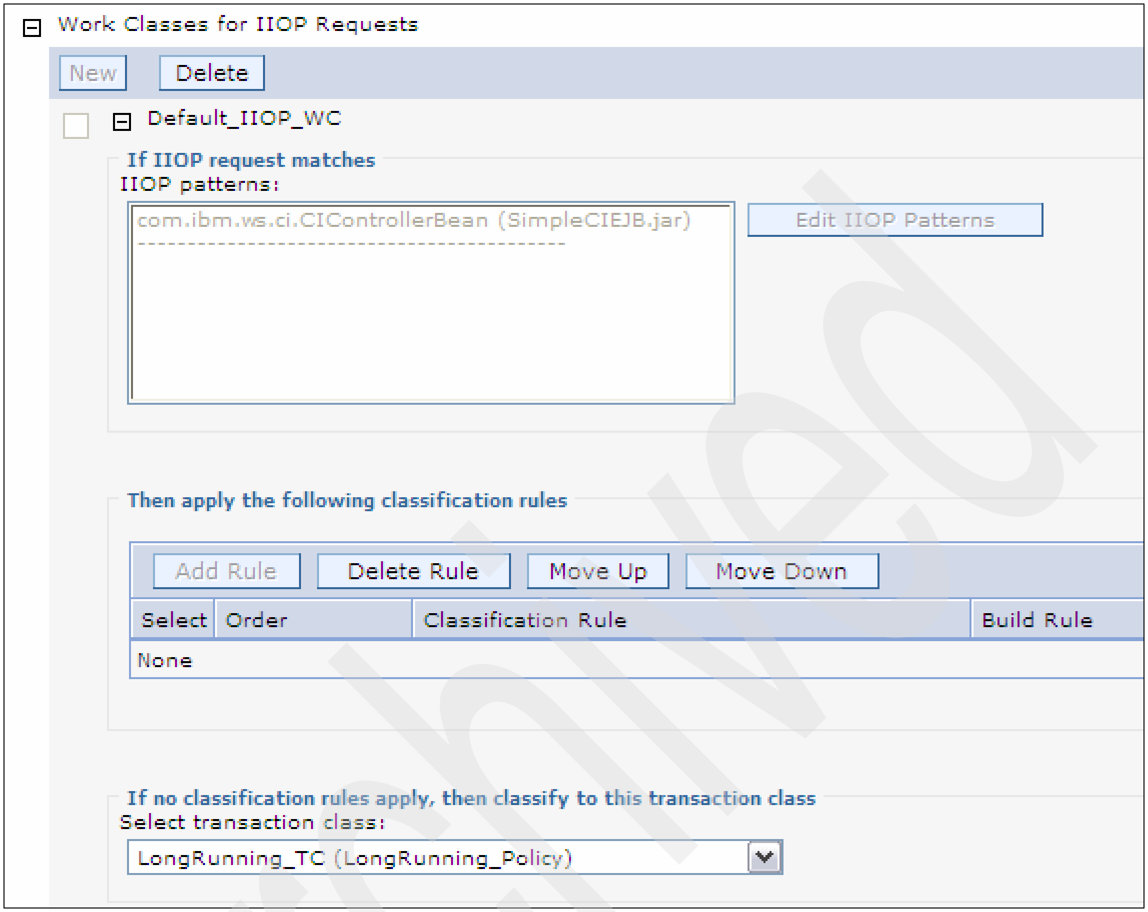


Figure 4-12 Relate the work class of the application to the transaction class

4.6.2 Configuring health policies for long-running applications

Health policies can be used to detect unhealthy servers and to take the action. While the steps to configure health policies are the same as those for the transactional applications, the type of the health policy applicable to long-running applications is limited. The type of the health policies used for long-running applications are age-based condition and memory condition. Some policies that use data from the ODR or Web container, such as excessive response time condition or workload condition, cannot be applied to long-running applications. The behavior to the age-based policy is also different from the one for the transactional application in that it waits for the long-running jobs to complete.

Age-based condition

When the age-based condition occurs and no jobs are running on the business grid server, the restart of the node is performed.

When the age-based condition occurs for a node in which jobs are running, the long-running scheduler stops dispatching new jobs to that server. The health controller continues to indicate the age-based condition periodically each time the controller cycle comes. After the last job on the server completes, the long-running scheduler allows the restart of the node the next time that the age-based condition is called by the health controller.

Memory condition: excessive memory usage, memory leak

Memory conditions are considered more severe than age conditions. The long-running scheduler allows the restart of the node promptly without waiting for the jobs to complete. Any batch jobs that are interrupted because of the restart are automatically restarted when the new server starts. Any active compute-intensive jobs fail.

5

Configuring a long-running runtime environment

This chapter describes how to configure the runtime environment for batch and compute-intensive applications.

> **Note:** This chapter assumes that you already installed WebSphere Extended Deployment on a WebSphere Application Server Network Deployment V6.1. If you are using V6.0.x as a base, these instructions still apply, though you may have some navigational differences.

The following topics are contained in this chapter:

- Sample topology overview
- Create the cell
- Configure the long-running scheduler
- Configure the long-running execution environment
- Verify the environment

5.1 Sample topology overview

Figure 5-1 shows the sample topology that illustrates how to configure a long-running runtime environment.

Note: This illustrates a simple topology. The topology of your runtime may vary, depending on business and application requirements. For example you might choose to run the LREE dynamic cluster on more than just two nodes.

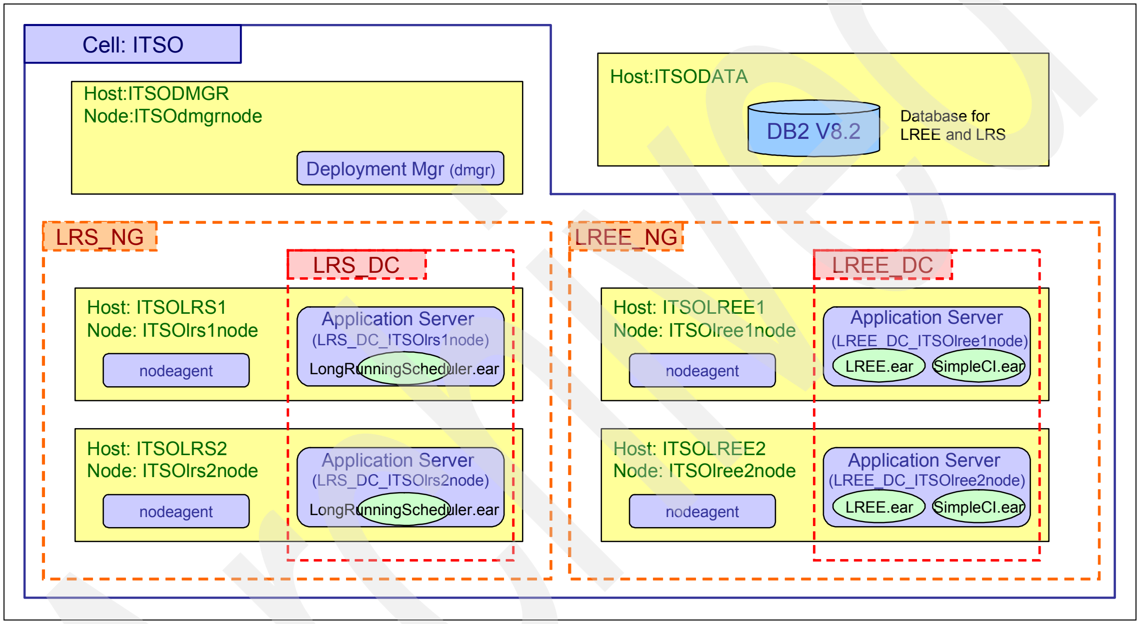


Figure 5-1 Topology for the long-running environment

The sample consists of six physical machines running Windows Server® 2003. All machines are connected to the same local area network. Five of the servers run WebSphere Extended Deployment. The sixth, ITSODATA, is used as a database server and runs DB2 Universal Database™ V8.2.

The WebSphere Application Server environment consists of one cell called ITSO. The deployment manager for the cell resides on the ITSODMGR host. The other four WebSphere machines each have one node that belongs to the ITSO cell.

Two node groups exist with a dynamic cluster for each, as shown in Table 5-1 on page 165. The LongRunningScheduler application and the LREE application were deployed, each in its own dynamic cluster. Databases defined on the database server, ITSODATA, are used by these applications to persist state information when running business grid applications.

Table 5-1 Node groups, dynamic clusters and applications in the sample topology

Node Group	Nodes	Dynamic cluster	Application (.ear)
LRS_NG	ITSOlrs1node ITSOlrs2node	LRS_DC	LongRunningScheduler
LREE_NG	ITSOlree1node ITSOlree2node	LREE_DC	LREE

The following sections go into more detail about how the sample runtime environment is configured.

5.2 Create the cell

This section describes how we set up the WebSphere cell for the sample scenario. This process assumes that the software listed below is installed on all five WebSphere Extended Deployment servers:

- WebSphere Application Server Network Deployment V6.1.0.1 or later
- WebSphere Extended Deployment V6.0.2 or later

For the sample topology, we used WebSphere Extended Deployment V6.0.2 and WebSphere Application Server Network Deployment V6.1.0.2.

Creating a deployment manager and node in WebSphere Extended Deployment is no different than creating them in WebSphere Application Server Network Deployment with the exception of the location of the wizard used to create the profiles. For that reason, we do not go into a step-by-step description of the process; instead, we simply summarize how this was done for the sample topology.

5.2.1 Create the deployment manager

The first step is to create a deployment manager profile on the ITSODMGR server.

Note: To create Extended Deployment-capable profiles, you must use the profile creation wizard supplied by WebSphere Extended Deployment, and not the profile creation tool (6.0) or profile management tool (6.1) that comes with Network Deployment.

On the server for the deployment manager, execute the following command to start the Profile Creation Wizard:

```
<WAS_install_root>\bin\ProfileCreator\XD\pcatWindows.exe
```

In the sample topology, the deployment manager profile was created with the values shown in Table 5-2.

Table 5-2 Values for the deployment manager profile in the sample topology

Property Name	Value
Profile name	ITSOdmgr
Node name	ITSOdmgrnode
Host name	ITSOdmgr
Cell name	ITSO

When the profile creation process was complete, the deployment manager was started with the following command:

```
<WAS_install_root>\bin\startManager.bat
```

5.2.2 Configure custom profiles

The next step is to create a custom profile on each of the remaining WebSphere nodes (see Figure 5-1 on page 164).

To create each custom profile, enter the following command on the node where the profile will reside, and complete the steps in the Profile Creation Wizard. Once again, be sure to use the profile creation wizard that shipped with Extended Deployment:

```
<WAS_install_root>\bin\ProfileCreator\XD\PcatWindows.exe
```

In the sample topology, we opted to have the custom profiles federated to the ITSO cell as part of the profile creation process. After selecting this option in the wizard, we were asked for the host name and SOAP port of the deployment manager (Table 5-3 on page 167) so the federation could take place (the deployment manager must be running).

Table 5-3 Hostname and soap port for the deployment manger

Property name	Value
Host name for the deployment manager	ITSOdmgr
SOAP port for the deployment manger	8879

To verify that the cell environment is created correctly, log on to the administrative console and navigate to **System administration** → **Nodes.** You should see the deployment manager and federated nodes as shown in Figure 5-2.

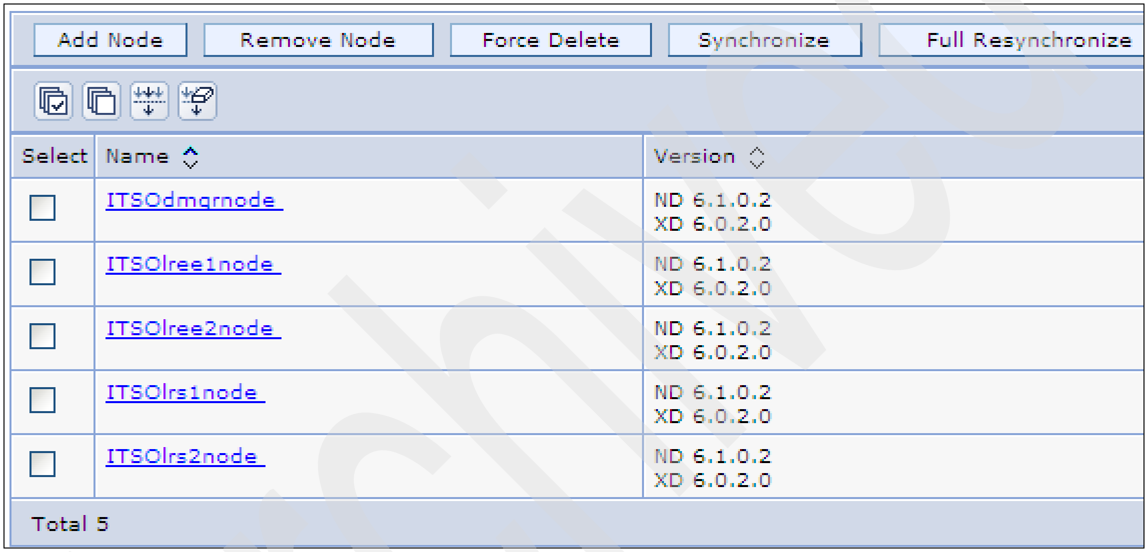


Figure 5-2 List in of nodes in the cell

5.3 Configure the long-running scheduler

The next step is to configure the long-running scheduler. WebSphere Extended Deployment provides a jython script named setupLongRunning.py to help configure the long-running environment. Before running this script you need to configure a node group, create a database for the long-running scheduler, and configure the JDBC provider and data source for the database. These steps are covered from section 5.3.1, “Create the node group” on page 168 to section 5.3.3, “Configure the JDBC provider and data source” on page 170.

If you use setupLongRunning.py for the remaining steps, skip to section 5.3.8, “Configure the long-running scheduler using a script” on page 178 directly. Otherwise follow the steps.

Table 5-4 shows the steps to follow depending on the method you choose.

Table 5-4 Using the script versus manual configuration

Configuration steps if using the script	**Manual configuration steps**
5.3.1, “Create the node group” on page 168	5.3.1, “Create the node group” on page 168
5.3.2, “Create the long-running scheduler database” on page 169	5.3.2, “Create the long-running scheduler database” on page 169
5.3.3, “Configure the JDBC provider and data source” on page 170	5.3.3, “Configure the JDBC provider and data source” on page 170
5.3.8, “Configure the long-running scheduler using a script” on page 178	5.3.4, “Create and configure the dynamic cluster” on page 173
---	5.3.5, “Enable the startup beans service” on page 174
---	5.3.6, “Configure default_host virtual host alias for the LRS” on page 175
---	5.3.7, “Configure and install the LongRunningScheduler application” on page 176
5.3.9, “Verify LongRunningScheduler application” on page 178	5.3.9, “Verify LongRunningScheduler application” on page 178

5.3.1 Create the node group

Create the node group that hosts the dynamic cluster for the long-running scheduler. To set up the node group for the sample topology, do the following:

1. Select **System administration** → **Node groups**, and click **New**.
2. Enter `LRS_NG` for the name of the new node group, and click **Apply**.
3. Click **Node group members**.
4. Click **Add**.
5. Select **ITSOlrs1node** and **ITSOlrs2node**, and click **Add**.
6. Save your changes to the master configuration.

5.3.2 Create the long-running scheduler database

The LongRunningScheduler application persists information about jobs submitted to a database. You need to create the database, and use the DDL supplied with Extended Deployment to initialize it. The following steps show how the database for the sample topology was created.

Create the LRS database

This section shows how to create the database for the LongRunningScheduler application. We refer to this as the LRS database.

1. Logon to the database node (ITSODATA) using the database administrator account.
2. Open a DB2 command window by navigating to **Start** → **Programs** → **IBM** → **IBMDB2** → **Command Line Tools** → **Command Window**.
3. Execute the following command.

   ```
   db2 create database <databaseName>
   ```

   We created a database named ITSOLRS as shown in Example 5-1.

Example 5-1 Create the LRS database

```
C:\>db2 create database ITSOLRS
DB20000I  The CREATE DATABASE command completed successfully.
```

Run the DDL file

WebSphere Extended Deployment provides DDL files that define the tables in the LRS database. The DDL file for DB2 can be found in the following:

```
<WAS_install_root>/longRunning/CreateLRSCHEDTablesDB2.ddl
```

1. Copy the CreateLRSCHEDTablesDB2.ddl file to the database node.
2. Modify the database name and authentication information in the DDL file to suit your environment as shown in Example 5-2.

Example 5-2 Modify CreateLRSCHEDTablesDB2.ddl to suit your environment.

```
...
-- CONNECT TO LRSCHED;
CONNECT TO ITSOLRS USER db2admin USING ********;

CREATE SCHEMA LRSSCHEMA;
...
```

3. In the DB2 command window, execute the following command to run the DDL file.

```
DB2 -tvf CreateLRSCHEDTablesDB2.ddl
```

5.3.3 Configure the JDBC provider and data source

You will next need to define the LRS database to the cell by creating a JDBC provider and data source.

Create the J2C authentication alias

The J2C authentication alias ensures that the data source can authenticate to the database server. The authentication alias contains the user ID and password required by the database server to allow access to the LRS database.

1. Navigate to **Security** → **Secure administrations, applications, and infrastructure**.
2. In the Authentication section at the right hand panel, expand **Java Authentication and Authorization Service**, and click **J2C authentication data**.
3. Click **New** and enter the properties listed in Table 5-5.
4. Click **OK**.

Table 5-5 J2C Authentication Alias properties for LRS data source

Property	Value
Alias	ITSOLRS
User ID	db2admin
Password	********
Description	Authentication alias for LRS data source

Create a JDBC provider

If you do not have a JDBC provider defined for your database type, do the following:

1. In the administrative console, navigate to **Resources** → **JDBC** → **JDBC Providers**.
2. Set the scope to the cell level, and click **New** to start creating a new JDBC provider.
3. In step 1, provide the values listed in Table 5-6 on page 171, and click **Next** to proceed.

Table 5-6 JDBC Provider settings

Property name	Values
Database type	DB2
Provider type	DB2 Universal JDBC Driver Provider
Implementation type	XA data source
Name	DB2 Universal JDBC Driver Provider (XA)

We chose to use a XA data source type so that we can use the same provider for the LREE database, but this is not necessary for the LRS database.

> **Note:** The LRS database can safely use a non-XA data source. In many cases, so can the LREE database. The use of an XA data source is recommended for the LREE database in case a batch application is deployed that updates other XA resources in the same checkpoint. If you only deploy applications that access the same transactional resource manager, then 2PC is not required. Hence XA is not required.

4. In step 2, enter the location of the driver files.

   In the test environment, we used the IBM DB2 Universal JDBC driver Type IV. This required that WebSphere have access to the following DB2 files:

   – db2jcc.jar
   – db2jcc_license_cisuz.jar

   On our test WebSphere systems, these files were located in C:SQLLIB\java.

   The information you enter here will become the value of the WebSphere environment variable that is displayed on this page at the cell scope, in the form of ${DATABASE_JDBC_DRIVER_PATH} (new with WebSphere Application Server V6.1).

   Because this is a type 4 driver, the native library path can be left blank. Click **Next** to proceed.

5. Review the summary, and click **Finish** to configure the JDBC provider.

Create the data source

Use the following steps to create the data source for the LRS database.

1. Navigate to **Resources** → **JDBC** → **Data sources.**
2. Set the scope to the cell level.
3. Click **New**.

4. At step 1 provide the values shown in Table 5-7.

Table 5-7 Data source settings for the LRS database

Data source name	LRS_DS
JNDI name	jdbc/lrsched
Component managed and authentication recovery alias	ITSOdmgrnode/ITSOLRS

5. Click **Next**.
6. Choose to select an existing JDBC provider. Pick the **DB2 Universal JDBC Driver Provider (XA)** from the drop-down, and click **Next**.
7. Enter the database-specific properties (Table 5-8). Select the option for **using this data source in CMP**, and click **Next**.

Table 5-8 Database specific properties for data source LRS_DS

Database property	Value
Database name	ITSOLRS
Driver type	4
Database server	ITSODATA
Port number	50000

8. Check the summary, and click **Finish** to create the data source.
9. Save your changes to the master configuration.

After configuring the data source, you should test it to ensure that it is working properly.

1. Go to **Resources** → **Data sources**, and check the box to the left of the data source.
2. Click **Test connection**. You should see a message that the test connection operation was successful.

5.3.4 Create and configure the dynamic cluster

> **Note:** If you plan to use the setupLongRunning.py script, skip to section 5.3.8, “Configure the long-running scheduler using a script” on page 178.

In this section, we create a dynamic cluster to which the LongRunningScheduler application is mapped. We will refer to this as the LRS dynamic cluster.

1. Select **Servers** → **Dynamic Clusters**, and click **New**.
2. At step 1, enter `LRS_DC` as the dynamic cluster name, and map this cluster to the node group LRS_NG. Accept the default for all other options, and click **Next**.
3. At step 2, select **defaultXD** as the server template, and click **Next**.
4. We need to ensure that there is always exactly one LRS application server running. At step 3, select **Keep one instance running at all times**, and set the limit for the number of instances that can start to `1` as shown in Figure 5-3 on page 174.

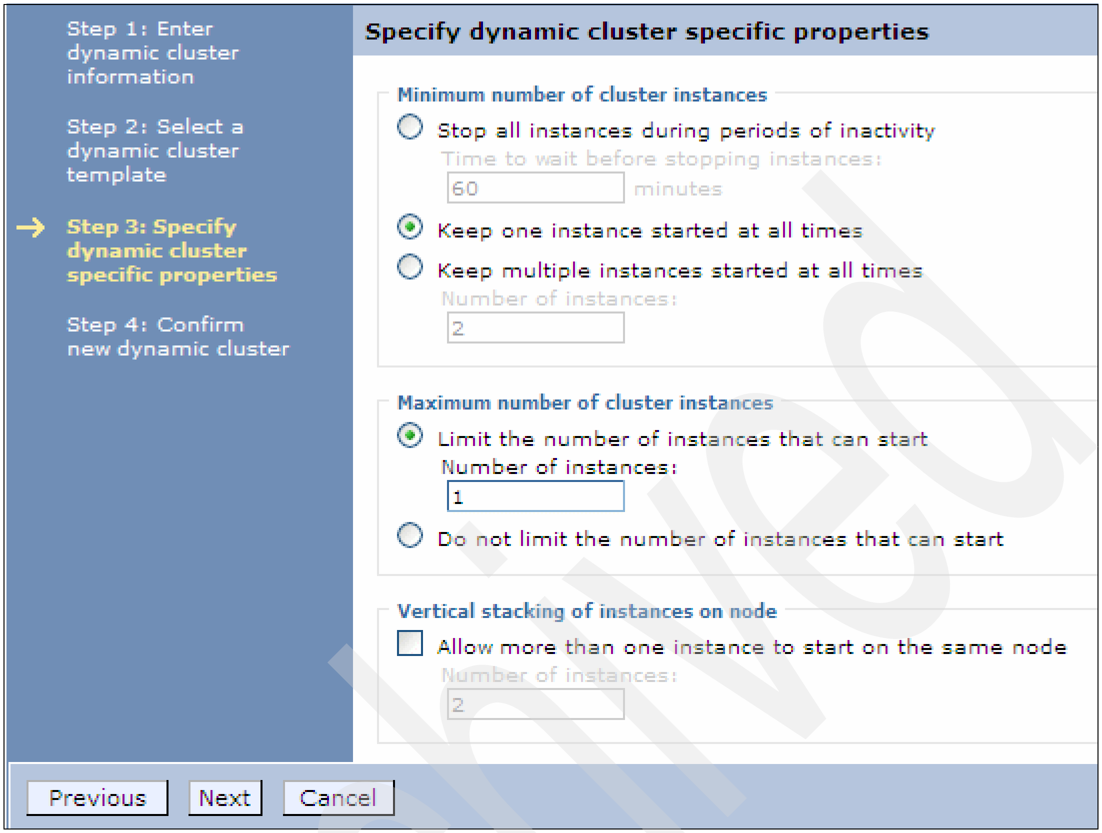


Figure 5-3 Dynamic cluster specific properties for LRS

5. Select **Next**, review the summary of actions, and click **Finish**.

   The dynamic cluster will be created and you will see the new cluster in the list of dynamic clusters.

6. Save your changes.
7. Check the box to the left of LRS_DC. Select **Automatic** or **Supervise** as the operational mode, and click **Set mode**.

5.3.5 Enable the startup beans service

Asynchronous beans are used to detect where the long-running scheduler and long-running execution environments are running and dispatch work to different endpoints in the cell. Therefore the startup beans service should be enabled on all cluster members in the LRS dynamic cluster.

Use the following steps to enable the startup beans service on all dynamic cluster members for the long-running scheduler:

1. Navigate to **Servers** → **Application server**, and select one of the cluster members. The cluster members are easy to recognize. Their names are in the following format:

   <dynamic_cluster_name>_<node_name>

2. In the right hand panel, navigate to **Container Services**, and select **Startup beans service**.
3. In the Startup beans service page, select **Enable service at server startup**, and click **OK**.
4. Repeat these steps for all other cluster member in the dynamic cluster.
5. Save your changes to the master configuration.

5.3.6 Configure default_host virtual host alias for the LRS

The LongRunningScheduler application has a Web service interface and an EJB interface to submit jobs. When you use the `lrcmd` command to submit jobs, this command uses the Web service interface. We need to ensure that the long-running scheduler can accept requests through SOAP over HTTP, which requires that the port and host name of long-running scheduler be registered to the HTTP Web container transport (WC_defaulthost) in the virtual host settings.

1. To determine the HTTP Web container transport, navigate to **Application servers**, and select one of your LRS application servers. In the right hand panel, navigate to **Communications**, and click **Ports**. Note the port number for WC_defaulthost. This is the HTTP Web container transport port for this application server. Repeat this for each server in the LRS dynamic cluster.
2. Then navigate to **Environment** → **Virtual Hosts**, and select **default_host**.
3. Select **Host Aliases** in additional properties and verify whether a definition exists for the ports you noted down earlier. The definition should match all host names (“*”) for the port.
4. If the port is not listed, click **New**. Type “*” in the Host Name field, and enter the port you noted down earlier.
5. Save your changes to the master configuration.

> **Note:** You can use a less generic host alias definition instead by defining two host aliases for the port, each matching only the host name of the machine hosting the LRS application server.

5.3.7 Configure and install the LongRunningScheduler application

The application for the long-running scheduler is deployed as an application into the dynamic cluster. There are also properties that must be updated using the administrative tools.

Install the LongRunningScheduler application

The LongRunningScheduler application EAR file is located in *<WAS_install_root>*/installableApps/LongRunningScheduler.ear.

1. Go to **Applications**, and click **Install New Application**.
2. Browse to the LongRunningScheduler EAR file and click **Open**.
3. Check the **Prompt me only when additional information is required** option (new in WebSphere Application Server V6.1). This installs the application with default settings (including the default_host virtual host), only prompting you for required values.
4. At step 1 “Select installation options”, accept all the defaults, and click **Next** to proceed.
5. At step 2 “Map modules to servers”, ensure that the LongRunningJobSchedulerEJBs and LongRunningJobSchedulerWebSvcRouter modules are mapped to the LRS dynamic cluster, LRS_DC, as shown in Figure 5-4. If this is not the case, check the boxes to the left of both modules, select LRS_DC from the drop-down, and click **Apply**. Then click **Next** to proceed to the summary.

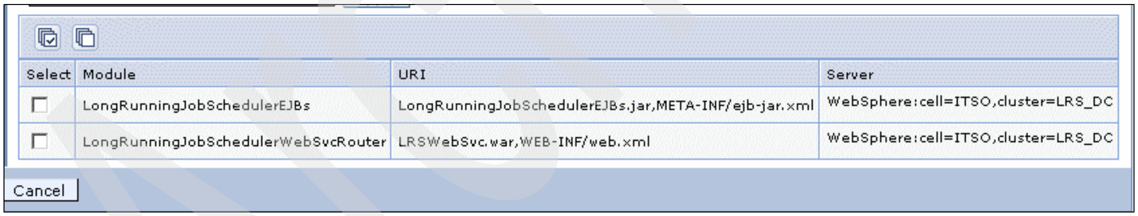


Figure 5-4 Mapping both LRS modules to DC_LRS.

6. At step 3, click **Finish** to start the installation.
7. The installation can take a few minutes. Look for messages indicating a successful install as shown in Example 5-3.

Example 5-3 Successful installation message

```
ADMA5005I: The application LongRunningScheduler is configured in the
WebSphere Application Server repository.
```

```
ADMA5011I: The cleanup of the temp directory for application
LongRunningScheduler is complete.

ADMA5013I: Application LongRunningScheduler installed successfully.
```

8. Save your changes.

Configure the long-running scheduler

Before starting the LongRunningScheduler application, the properties for the long-running scheduler need to be updated to reflect the values associated with the LRS database.

1. Using the administrative console, navigate to **System administration** → **Long-running scheduler**, and enter the values for the database schema name, data source JNDI name, and data source alias for the LRS database. The values for this scenario are shown in Figure 5-5.

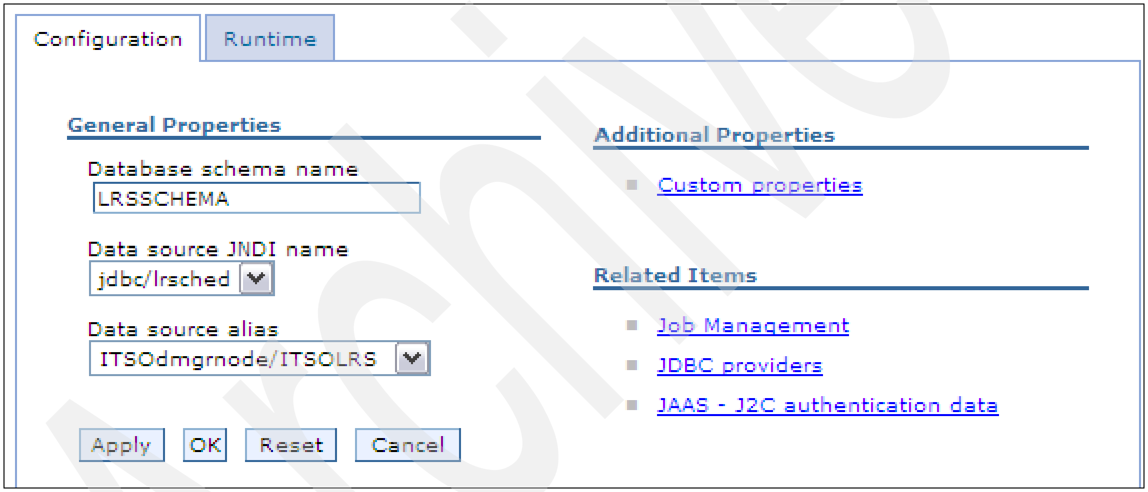


Figure 5-5 Settings for long-running scheduler

2. Click **OK**, and save your changes to the master configuration.

> **Note:** If you choose to use a different schema name for the long-running scheduler database tables, you need to configure it here and change the DDL accordingly.

5.3.8 Configure the long-running scheduler using a script

Note: Using the script is an alternative to the steps detailed in the following sections:

- 5.3.4, "Create and configure the dynamic cluster" on page 173
- 5.3.5, "Enable the startup beans service" on page 174
- 5.3.6, "Configure default_host virtual host alias for the LRS" on page 175
- 5.3.7, "Configure and install the LongRunningScheduler application" on page 176

The setupLongRunning.py script provided by WebSphere Extended Deployment helps to configure long-running environment. This script does the following for the long-running scheduler environment:

- Creates an LRS dynamic cluster
- Sets the dynamic cluster operational mode
- Sets the maximum number of instances to 1
- Enables the startup beans service for client instances
- Sets up virtual hosts for cluster instances
- Configures the long-running scheduler environment
- Deploys the LongRunningScheduler application

Because this script is also used to set up the long-running execution environment, you need to specify "lrs" as the first option. Details for using this script are in the WebSphere Extended Deployment Information Center at the following Web location:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp?topic=/com.ibm.websphere.xd.doc/info/scheduler/rbglrsched.html

For our sample topology, we would use the following commands:

```
cd C:\WebSphere\AppServer61\bin

wsadmin.bat -lang jython -f setupLongRunning.py lrs --ngName LRS_NG
--dcName DC_LRS --jndiName jdbc/lrsched
```

5.3.9 Verify LongRunningScheduler application

Verify that the application works by starting one of the servers in the LRS dynamic cluster. When the server starts, navigate to the list of enterprise applications, and verify that the LongRunningScheduler application started.

You can also see the status of the long-running scheduler by navigating to **System administration** → **Long-running scheduler**. Switch to the Runtime tab, and check the Status field for a "Started" status.

> **Note:** On first start we recommend that you monitor the SystemOut.log of the application server in order to spot potential issues.

5.4 Configure the long-running execution environment

Next we need to configure the long-running execution environment. The setupLongRunning.py script can also be used to configure the long-running execution environment. Before running this script, you need to configure a node group, create a database for the LREE application, and configure a JDBC provider and data source. These steps are covered from section 5.4.1, "Create the node group" on page 179 to section 5.4.3, "Create the data source" on page 180.

If you use setupLongRunning.py for the remaining steps, skip to section 5.4.8, "Configure LREE using setupLongRunning.py script" on page 184 directly; otherwise, follow the steps and skip that section.

5.4.1 Create the node group

Using the process in section 5.3.1, "Create the node group" on page 168, create a node group to host the dynamic cluster for the long-running execution environment. In this case, we created a node group called LREE_NG that contains ITSOlree1node and ITSOlree2node.

5.4.2 Create the database for LREE

Using the process in section 5.3.2, "Create the long-running scheduler database" on page 169, create a database for the LREE called ITSOLREE as shown in Example 5-4. We will refer to this as the LREE database.

Example 5-4 Create a database for LREE

```
C:\>db2 create database ITSOLREE
DB20000I  The CREATE DATABASE command completed successfully.
```

WebSphere Extended Deployment provides database-specific DDL files to define the LREE database tables. The DDL file for DB2 resides in the following location:

```
<WAS_install_root>/longRunning/CreateLREETablesDB2.ddl
```

1. Copy the CreateLREETablesDB2.ddl file to the database server.
2. Modify the database name and authentication information in the DDL file as shown in Example 5-5.

Example 5-5 Modify CreateLREETablesDB2.ddl to suit your environment.

```
...
-- CONNECT TO LREE;
CONNECT TO ITSOLREE USER db2admin USING ********;

CREATE SCHEMA LREESCHEMA;
...
```

3. In the DB2 command window, execute the following command to run the DDL file.

   `DB2 -tvf CreateLRSCHEDTablesDB2.ddl`

5.4.3 Create the data source

You need to create a data source for the LREE database. You can reuse the JDBC provider used for the LRS database, and if the user ID and password to access the database are the same, you can reuse the J2C authentication alias created for the LRS database.

Create the J2C authentication alias

Use the process outlined in “Create the J2C authentication alias” on page 170 to create the J2C authentication alias to be used to access the LREE database.

We used the properties listed in Table 5-9 for our scenario.

Table 5-9 J2C Authentication Alias properties for LREE data source

Property	Value
Alias	ITSOLREE
User ID	db2admin
Password	********
Description	Authentication alias for LREE data source

Create the data source

Use the same procedure we used in "Create the data source" on page 171 to create a data source for the LREE database. The values used for our scenario are shown in Table 5-10. The JDBC provider selected is the same as what we used for the LRS database.

Table 5-10 Data source settings for the LREE database

Data source name	LREE_DS
JNDI name	jdbc/lree
Component managed and authentication recovery alias	ITSOdmgrnode/ITSOLREE
Database name	ITSOLREE
Driver type	4
Database server	ITSODATA
Port number	50000

After configuring the data source, be sure to test it to make sure it is working.

5.4.4 Create the LREE dynamic cluster

Note: If you use the setupLongRunning.py script, skip to section 5.4.8, "Configure LREE using setupLongRunning.py script" on page 184 directly.

Use the following steps to create a dynamic cluster to which LREE are mapped.

1. Select **Servers** → **Dynamic Clusters**, and click **New**.
2. At step 1, enter `LREE_DC` as the dynamic cluster name and map this cluster to the LREE_NG node group. Accept the default for all other options, and click **Next**.
3. At step 2, select **defaultXD** as the server template, and click **Next**.
4. At step 3, accept the defaults, and click **Next** to proceed.

   Because we decided to allow only one LREE application server to run on each node, we did not select the vertical stacking option. If your system is rich in system resources and can afford to run multiple instances per node, you can select **Allow more than one instance to start on the same node**, and specify the number of instances to run.
5. Review the summary of actions, and click **Finish**.

6. (Optional) You can define a custom property for the LREE dynamic cluster to set the maximum number of jobs running concurrently in one LREE instance.

   To specify the number, create a custom property as shown in Table 5-11. If you do not set this custom property, the default maximum number to be executed concurrently per LREE instance is 2.

Table 5-11 Custom Property for LREE dynamic clusterTable 5-11

Property name	Value
com.ibm.websphere.longrun.EndpoingCapacity	(the number allowed to run concurrently)

7. Save your changes.

5.4.5 Enable the startup beans service on LREE application servers

Use the process in section 5.3.5, “Enable the startup beans service” on page 174 to enable the startup beans service for the servers in the LREE dynamic cluster.

> **Note:** If you decide to add more nodes to the node group that is running the LREE dynamic cluster, make sure you enable the startup beans service on the new cluster members.

5.4.6 Configure the default_host virtual host alias

When a job is dispatched to the LREE, the Web service interface is used. We need to ensure that each member of the LREE dynamic cluster can accept requests through SOAP over HTTP, which requires the port and host name of the LREE dynamic cluster members to be registered in the HTTP Web container transport (WC_defaulthost) in the virtual host settings.

To do this, use the process detailed in section 5.3.6, “Configure default_host virtual host alias for the LRS” on page 175.

5.4.7 Install the LREE application

The LREE application is located in *<WAS_install_root>*/installableApps/LREE.ear.

1. Go to **Applications**, and click **Install New Application**.
2. **Browse** to point to the EAR file, and click **Open**.

3. Select **Show me all installation options and parameters** for this application. This installation scenario is the same as V6.0 and you are asked in every option. Click **Next** to proceed.
4. Select **Generate Default Bindings**. Then scroll down to Virtual Host and select to use “default_host” as the default virtual host name for Web and SIP modules. Click **Next**.
5. Accept the Application Security Warning, and click **Continue** to proceed.
6. At step 1 “Select installation options”, ensure that the option **Deploy Enterprise Beans** is selected, and click **Next**.
7. At step 2 “Map modules to servers”, make sure that the modules BatchJobExecutionEnvironmentEJBs and EndpointWebService are mapped onto the LREE dynamic cluster. If this is not the case, check the box to the left of both modules, select the LREE_DC from the drop-down, and click **Apply**. Click **Next** to proceed.
8. At step 3 “Provide options to perform the EJB Deploy, select **DB2UDB_V82** as the database type (in our case), and specify **LREESCHEMA** as the schema name as shown in Figure 5-6.

EJB Deployment Options	Enable
Deploy EJB option - Class path	
Deploy EJB option - RMIC	
Deploy EJB option - Database type	DB2UDB_V82
Deploy EJB option - Database schema	LREESCHEMA

Figure 5-6 Specify the database type and schema

9. Proceed directly to step 8 by clicking **Step 8** in the vertical blue bar.
10. At step 8 ”Map default data source for modules containing 2.x entity beans”, do the following:
    – Check the box to the left of the BatchJobExecutionEnvironmentEJBs module, select **Per application** for Resource authorization, and click **Apply**.
    – Check the module again, and select **Use default method** under Specify authentication method. Select ITSOdmgrnode/ITSOLREE from the drop-down, and click **Apply** once more.
11. Go to step 14, and review the summary. Click **Finish** to start the installation. The installation might take a few minutes. Look for messages that indicate a successful install as shown in Example 5-6 on page 184.

Example 5-6 Successful installation message

```
ADMA5005I: The application LREE is configured in the WebSphere
Application Server repository.

ADMA5011I: The cleanup of the temp directory for application LREE is
complete.

ADMA5013I: Application LREE installed successfully.
```

12..Save the changes to the master configuration

5.4.8 Configure LREE using setupLongRunning.py script

Note: Using the script is an alternative to the steps detailed in the following sections:

- 5.4.4, "Create the LREE dynamic cluster" on page 181
- 5.4.5, "Enable the startup beans service on LREE application servers" on page 182
- 5.4.7, "Install the LREE application" on page 182

The setupLongRunning.py script can also be used to configure the long-running execution environment. This script will do the following:

- Create the LREE dynamic cluster
- Set the dynamic cluster operational mode
- Set up virtual hosts for cluster instances
- Deploy the LREE application

Because this script is also used to set up the long-running scheduler, you need to specify "lree" as the first option. Details for using this script reside in the WebSphere Extended Deployment Information Center at the following Web location:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp?topic=/com.ibm.websphere.xd.doc/info/scheduler/rbglrsched.html

The following commands invoke the script for our test environment:

```
cd C:\WebSphere\AppServer61\bin

wsadmin.bat -lang jython -f setupLongRunning.py lree --ngName LREE_NG
--dcName DC_LREE --jndiName jdbc/lree --dbType DB2UDB_V82
```

5.4.9 Verify the LREE application

Verify that the application works by starting one of the servers in the LREE dynamic cluster. Once the server starts, verify that the LREE application started.

> **Note:** We recommend that you monitor the SystemOut.log of the application server in order to spot potential issues.

5.5 Verify the environment

We completed the configuration of the long-running runtime environment. Next, we install a sample application called SimpleCI to verify the environment.

5.5.1 Install the compute-intensive sample application

The compute-intensive sample application, SimpleCI, provided with WebSphere Extended Deployment can be found in the following location:

`<WAS_install_root>/installableApps/SimpleCI.ear`

To install the application, use the following instructions:

1. Go to **Applications**, and click **Install New Application**.
2. **Browse** to the EAR file, and click **Open**.
3. Select **Prompt me only when additional information is required**, and click **Next**.
4. At step 1 “Select installation options”, deselect the **Deploy enterprise beans** option, and click **Next**.
5. At step 2 “Map modules to servers”, make sure that SimpleCIEJB is mapped to the LREE_DC dynamic cluster, and click **Next**.
6. At step 3 “Summary”, review the summary, and click **Finish** to start the installation.
7. Installation completes quickly because the deploy EJB option was not selected. Look for messages indicating that the installation was successful, as shown in Example 5-7.

Example 5-7 Successful installation message

```
ADMA5005I: The application SimpleCIEar is configured in the WebSphere
Application Server repository.
ADMA5011I: The cleanup of the temp directory for application
SimpleCIEar is complete.
```

```
ADMA5013I: Application SimpleCIEar installed successfully.
```

8. Save the changes to the master configuration.
9. Start the application.
10. Navigate to **Applications** → **Enterprise Applications**, select **SimpleCIEar**, and click **Start**. Within a few seconds, the application should be up and running.

5.5.2 Test the compute-intensive sample application

In order to be able to test the newly deployed sample application, ensure that the following applications are all running (Figure 5-7):

- LREE
- LongRunningScheduler
- SimpleCIEar

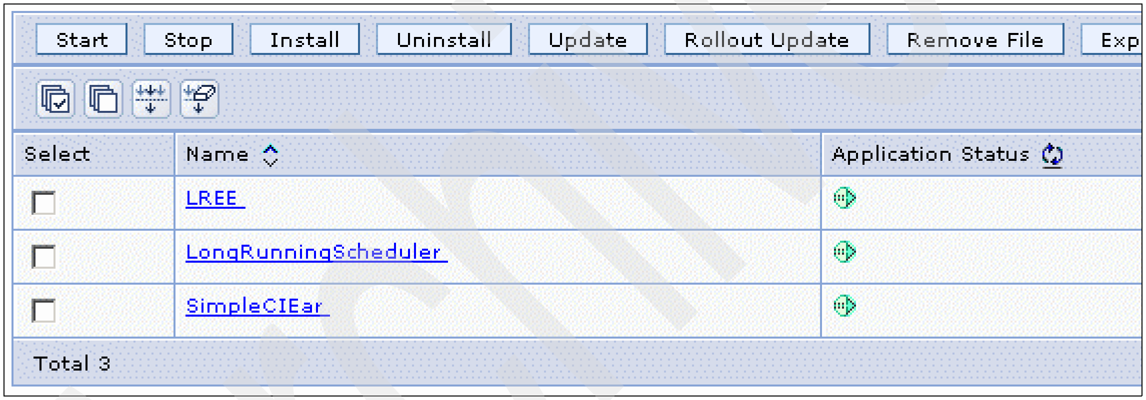


Figure 5-7 Check the status of application

Modify the xJCL

Before we submit the request to the sample application, we need to modify the sample xJCL file with the necessary information.

1. Log on to the deployment manager node, and create a directory to place the xJCL file in, for example "C:\xJCL\SimpleCI\".
2. Copy the file <*WAS_install_root*>\longRunning\SimpleCIxJCL.xml to the directory created and open this file for editing.
3. We need to specify an output file for the SimpleCI application. Note that this application is running on the LREE_DC dynamic cluster, hence the file name we specify needs to point to a valid location on the LREE nodes. Change the line containing the property "outputFileName" as the following shows:

```
<prop name="outputFileName" value="C:\logs\SimpleCI\output.txt" />
```

4. We can also specify how long this application executes the calculations. To change it, modify the value of the calculationTimeInSecs property as shown below.

```
<prop name="calculationTimeInSecs" value="30" />
```

Using the xJCL file allows us to change the application behavior without changing the code.

Submit the job

Now we are ready to submit this request (job) to the long-running scheduler. Note that LongRunningScheduler application accepts all long-running jobs (both batch and compute-intensive). When a job is submitted, the long-running scheduler dispatches it to an active LREE application server.

1. Open a command prompt, and go to “C:\WebSphere\AppServer61\bin” directory.
2. Execute the following command to submit a job. If you do not specify the host or port, the default values are localhost and 80 respectively.

```
lrcmd -cmd=submit -xJCL=<path_to_xJCL> -host=<lrs_hostname>
-port=<lrs_WC_defaulthost>
```

If the job is submitted successfully, the following message is returned with the assigned JOBID, as shown in Example 5-8.

Example 5-8 Submitting a job to the sample application SimpleCI

```
C:\WebSphere\AppServer61\bin>lrcmd -cmd=submit -xJCL=C:\xJCL\SimpleCI\test.xml
-host=itsolrs1 -port=9080


CWLRB4940I: com.ibm.ws.batch.wsbatch : -cmd=submit -xJCL=C:\xJCL\SimpleCI\test.xml
-host=itsolrs1 -port=9080

CWLRB4960I: Tue Oct 24 11:59:56 EDT 2006 : com.ibm.ws.batch.wsbatch : Job
[SimpleCIEar:1] is submitted.
```

When the sample application executes, it writes log entries to the SystemOut.log, as shown in Example 5-9 as well as to the file you specified earlier on in the xJCL file.

Example 5-9 SystemOut.log of LREE_DC cluster member on ITSOlree1node

```
[10/24/06 12:00:19:468 EDT] 00000058 SystemOut     O Tue Oct 24 12:00:19 EDT 2006:
SimpleCI application starting...
```

```
[10/24/06 12:00:19:468 EDT] 00000058 SystemOut     O -->Will loop processing a
variety of math functions for approximate
ly 30.0 seconds!
[10/24/06 12:00:49:468 EDT] 00000058 SystemOut     O Tue Oct 24 12:00:49 EDT 2006:
SimpleCI application complete!
[10/24/06 12:00:49:468 EDT] 00000058 SystemOut     O -->Actual Processing time = 30.0
seconds!
```

Note: You can see the state of jobs by navigating to **System Administration** → **Long-running Scheduler Job Management**.

6

Configuring a long-running development environment

This chapter describes how to configure a development environment to write applications that use the WebSphere Extended Deployment long running application extenders. Developers who need to write Batch or Compute Intense applications intended to run on WebSphere Extended Deployment V6.0.2.x can use this chapter as a reference when setting up their development environment.

This chapter contains the following topics:

- Development environment overview
- Preparing Rational Application Developer 6.0
- Installing WebSphere Extended Deployment
- Configuring the WebSphere test environment
- Configure the development environment
- Running the compute-intensive sample application
- Running the batch sample application

6.1 Development environment overview

Two IBM products are well-suited to develop applications for deployment in a WebSphere Extended Deployment environment: Rational Application Developer and the Application Server Toolkit.

When selecting an application development environment for long running applications, one important consideration is the test environment provided with the development tool. Because WebSphere Extended Deployment can be installed on a WebSphere Application Server Network Deployment 6.0 or 6.1 base, you should be aware of how the two development tools support these environments.

Rational Application Developer V6.0 provides a rich development environment, including a WebSphere test environment that supports WebSphere Application Server V6.0.x. WebSphere Extended Deployment can be installed on top of this test environment. However, at the time of this writing, Rational Application Developer V6.0 was not yet updated to support WebSphere Application Server V6.1 as an integrated test environment. You can still use Rational Application Developer V6.0 to develop and test long running applications on a WebSphere Application Server V6.1 test environment, but you need to export the application from the development environment as an EAR file and deploy that onto your server.

All editions of WebSphere Application Server include the Application Server Toolkit. In V6.0, the toolkit was an assembly and deployment tool rather than a development tool. In V6.1, the Application Server Toolkit is now a full-blown J2EE development tool. The Application Server Toolkit is targeted to support only the version of the WebSphere Application Server that it ships with. This means that Application Server Toolkit V6.1 supports all new features of WebSphere Application Server V6.1 and supports a V6.1 integrated test environment. The Application Server Toolkit does not, however, support any of the previous versions of WebSphere Application Server as integrated test environments. To use a WebSphere Extended Deployment server in the Application Server Toolkit test environment you need to install WebSphere Application Server V6.1 separately, install WebSphere Extended Deployment on top, and then register it as a test server to the Application Server Toolkit.

In our scenario, we chose to develop the long running applications using Rational Application Developer V6.0.

6.1.1 Using Extended Deployment in the integrated test environment

Integrating WebSphere Extended Deployment into the integrated test environment is a manual process involving the following steps:

1. Install WebSphere Extended Deployment V6.0.2 on top of the WebSphere Application Server Network Deployment integrated test environment.
2. Remove the gridendpointselector.jar from the lib directory of the integrated test environment so that the LongRunningScheduler and LREE applications work in a non-dynamic cluster environment.
3. Create and prepare the LRS and LREE databases.
4. Configure the integrated test server by creating the data sources for the databases and enabling the startup bean service for the server.
5. Deploy the LongRunningScheduler and LREE applications to the integrated test server.

Figure 6-1 illustrates an overview of the development environment.

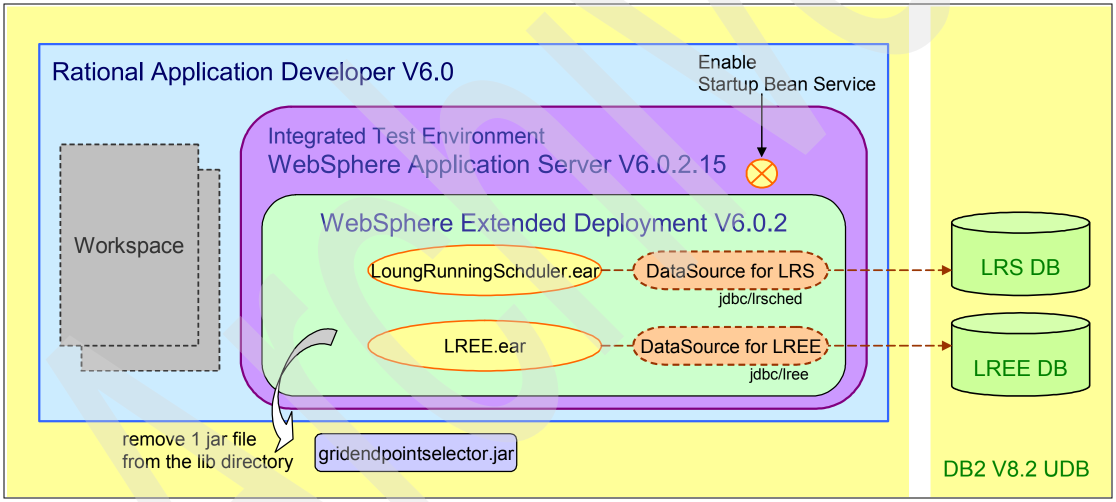


Figure 6-1 Extended Deployment in the development environment

6.2 Preparing Rational Application Developer 6.0

This section describes how to prepare a Rational Application Developer installation for developing and testing long-running applications for deployment on WebSphere Extended Deployment.

Note: We assume that you installed Rational Application Developer V6.0 with at least the following features:

- Integrated Development Environment
- IBM WebSphere Application Server V6.0 Integrated Test Environment

6.2.1 Update the development platform

Use the Rational Software Development Platform Product Updater to install the latest updates.

Note: The Rational Product Updater requires a connection to the Internet. If your machine does not have access to the internet, you can still install the required updates by downloading from another machine and installing them separately. For detailed instructions, visit the following Web page:

http://www.ibm.com/developerworks/rational/library/06/0718_saracevic/

1. Launch the tool by clicking **Start** → **Programs** → **IBM Rational** → **Rational Product Updater**.
2. Click **Find Updates** to search for product updates.
3. At the time of writing, we installed the following updates for IBM Rational Application Developer V6.0:
   - IBM Rational Application Developer Fix Pack 6.0.1.1
   - Interim Fix 001 for Rational Application Developer
   - Interim Fix 002 for Rational Application Developer
   - IBM WebSphere Application Server Test Environment 6.0.2.5
   - Java SDK Update for IBM WebSphere Application Server Test Environment

After completing the updates, the updater tool should look similar to Figure 6-2 on page 193.

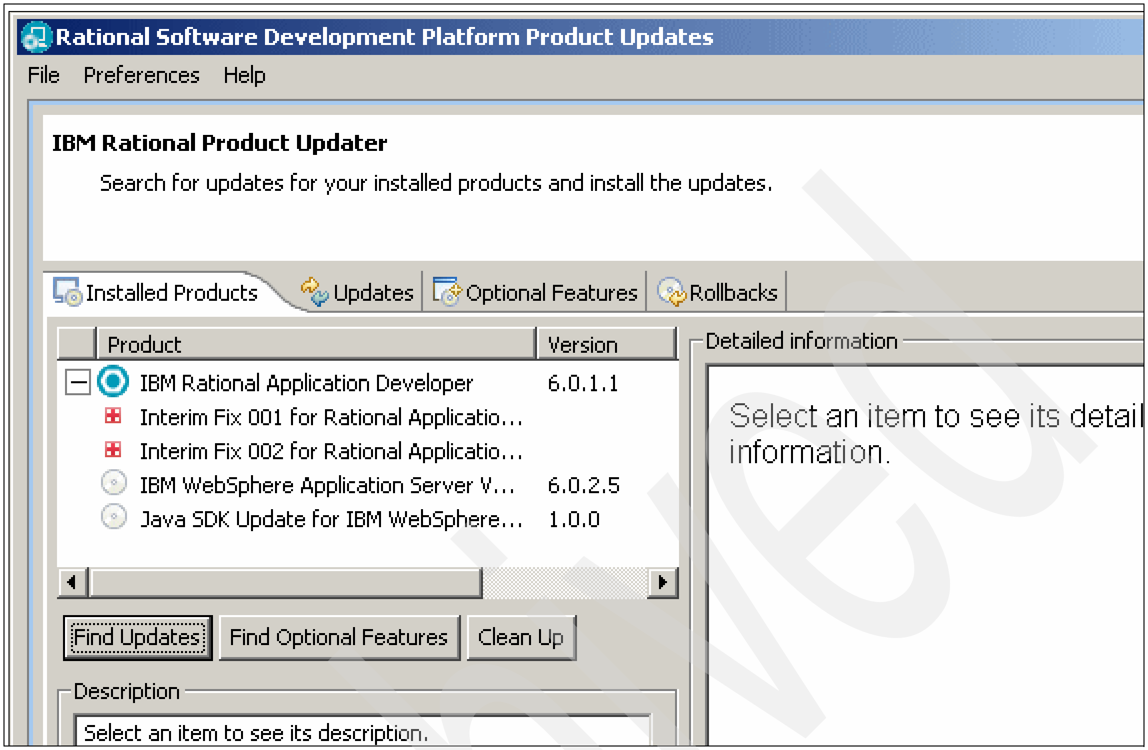


Figure 6-2 Installed updates for Rational Application Developer V6.0

6.3 Update the WebSphere test environment

WebSphere Extended Deployment 6.0.2 requires WebSphere Application Server Network Deployment 6.1.0.1 or higher or WebSphere Application Server Network Deployment 6.0.2.9 or higher.

http://www-1.ibm.com/support/docview.wss?uid=swg27008269

Rational Application Developer 6.0 only provides a WebSphere Application Server 6.0.x Test Environment. In order to be able to use this test environment for development of WebSphere Extended Deployment long-running applications, we need to upgrade this to the 6.0.2.9 level or higher.

At the time of writing, FixPack 15 was the most recent maintenance available so the first step in our test environment was to update it to 6.0.2.15.

Details on how to upgrade the WebSphere Test Environment in Rational Application Developer 6.0 can be found at the following Web site.

http://www.ibm.com/support/docview.wss?uid=swg21199423

6.4 Installing WebSphere Extended Deployment

The standard test environment is WebSphere Application Server Network Deployment. To use an Extended Deployment environment, you need to install Extended Deployment on top of the current test environment.

6.4.1 Installing WebSphere Extended Deployment 6.0.2

The process involves installing WebSphere Extended Deployment 6.0 into the existing Network Deployment V6.0.2 test environment and updating it with the latest maintenance. At the time of this writing, Refresh Pack 2 was available, bringing the level to 6.0.2.

Install WebSphere Extended Deployment 6.0

1. Launch the WebSphere Extended Deployment 6.0 Installation Wizard.
2. Select **WebSphere Extended Deployment Version 6.0**, and click **Next**.
3. Click **Next** at the Welcome window.
4. Accept the Software License Agreement.
5. Ensure that your system meets the system prerequisites, and click **Next**.
6. Select the directory for the WebSphere Application Server installed in the Rational Application Developer test environment:

   <*rad_install*>\runtimes\base_v6

7. Proceed through the rest of the wizard, accepting the defaults. When the installation finishes choose **Finish** to exit the wizard

Apply Refresh Pack 2

Download the Refresh Pack from the following support site:

http://www-1.ibm.com/support/docview.wss?rs=3023&uid=swg27005709

1. Copy the Refresh Pack to the updateinstaller maintenance directory:

   <*rad_install*>\runtimes\base_v6\updateinstaller

2. Launch the Update Installer by running the update.exe in the updateinstaller directory.

3. Click **Next** at the Welcome screen.
4. Accept the Software License Agreement.
5. Select the WebSphere Extended Deployment installation directory:

   `<rad_install>\runtimes\base_v6.`
6. Select **Install maintenance package**, and click **Next**.
7. Click **Browse** and select **6.0.0-WS-XD-RP0000002.pak**. Click **Open**, and click **Next** to proceed.
8. Accept the summary to start the installation.
9. Close the wizard after the installation completes.

6.5 Configuring the WebSphere test environment

Now that the installation of WebSphere Extended Deployment in the test environment is complete, there are several steps necessary to prepare the test environment for long-running applications.

6.5.1 Removing the business grid endpoint selector

Normally, the applications that provide the LRS and LREE functions are deployed to separate dynamic clusters and multiple application servers. However, in the test environment, they are deployed to one application server in a non-clustered environment.

When the LongRunningScheduler application dispatches a job, it uses the business grid endpoint selector to select an LREE endpoint. This mechanism does not work when both applications are on the same application server; therefore, jobs are not picked up by the LREE application.

In order to bypass this mechanism, we need to remove the file gridendpointselector.jar from the lib directory of the WebSphere test environment.

1. Change directory to *<rad_install>*\runtimes\base_v6\lib.
2. Make a backup copy of gridendpointselector.jar
3. Remove gridendpointselector.jar—it is not enough to rename it, you must remove it.

> **Important:** Keep in mind that applying a future Fix Pack or Refresh Pack for Extended Deployment is likely to restore the gridendpointselector.jar. This breaks the functionality of the development test environment.

6.5.2 Configuring and starting the server

Before starting the server, we updated the server definition for performance and usability:

1. Start Rational Application Developer. In the J2EE perspective, select the **Servers** view in the panel at the lower right corner.
2. Double-click the **WebSphere Application Server v6.0** server to open the configuration panel (Figure 6-3 on page 197):
   - Change the name of the server to `WebSphere Extended Deployment v6.0.2`.
   - Enable **Optimize server for testing and developing**.
   - Deselect **Enable automatic publishing**.

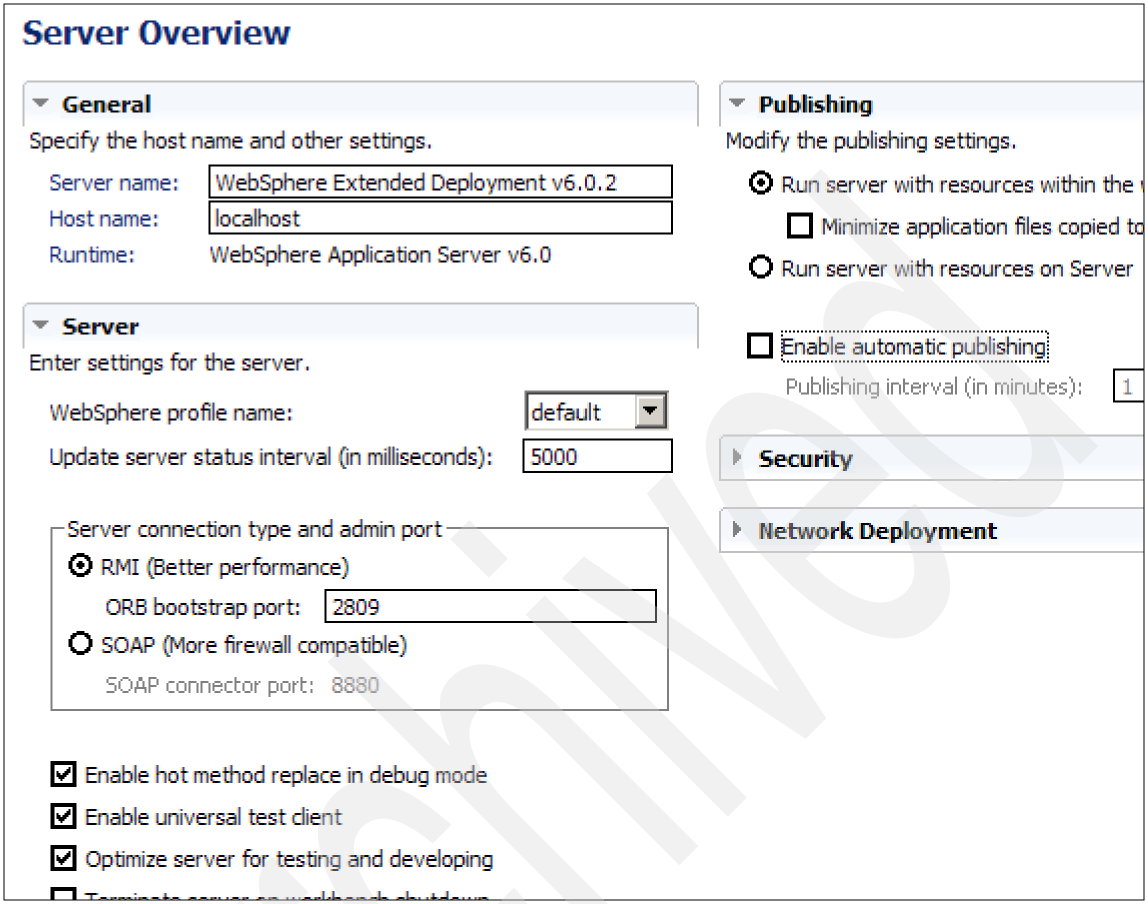


Figure 6-3 Configure the WebSphere test environment on Rational Application Developer

After we made and saved these changes, we started the server and logged on to the administrative console:

1. In the **Servers** view, right-click the server, select **Start**, and wait until it starts. You may switch to the Console view to look at the server's logs.

Note: We observed a number of exceptions during start up of the WebSphere Test Environment. This is caused by the lack of proxy.jar. This problem is documented in APAR PK30847. The fix is anticipated to be in Fix Pack 6.0.2.19 for WebSphere Application Server.

```
WSVR0501E: Error creating component com.ibm.ws.classify.filter.HttpRequestClassificationService
WSVR0501E: Error creating component com.ibm.wsmm.proxy.WsmmService
WSVR0501E: Error creating component com.ibm.ws.xd.dwlm.client.CountService
ODCF0002E: Exception: ODC violation from TreeBuilder.stateChanged
WSVR0501E: Error creating component null [classcom.ibm.ws.giop.filter.GiopFilterComponentImpl]
WSVR0501E: Error creating component null [class
com.ibm.ws.runtime.component.ApplicationServerImpl]
WSVR0501E: Error creating component null [class com.ibm.ws.xd.jms.arfm.JMSWSMMService]
```

2. In the Servers view, right-click the server, and select **Run administrative console**. This starts the console in a browser in the workbench.

With the exception of setting up the WebSphere cell and dynamic clusters, the process of configuring the runtime environment is very similar to that outlined in Chapter 5, “Configuring a long-running runtime environment” on page 163. There are some minor navigational differences in the administrative console because our production environment was based on WebSphere Application Server V6.1, and the development runtime is based on WebSphere Application Server V6. The following sections summarize the setup.

6.5.3 Creating and configuring the LRS and LREE databases

Create two databases for the development environment: one for the long-running scheduler and one for the long-running execution environment.

1. Using the process outlined in section 5.3.2, “Create the long-running scheduler database” on page 169, create the LRS database and run the DDL to create the tables. We called our database DEVLRS.

   The DDL resides in the `<`*`rad_install`*`>\runtimes\base_v6\longRunning` directory.

2. Using the process outlined in section 5.4.2, “Create the database for LREE” on page 179, create the LREE database and run the DDL to create the tables. We called our database DEVLREE.

3. Using the administrative console, create a JDBC provider.

Note: The runtime must have access to the JDBC drivers. In our case, we used DB2, so we made the following libraries available, located in the C:\SQLLIB\java library.

- db2jcc.jar
- db2jcc_license_cisuz.jar

To create the JDBC provider, navigate to **Resources** → **JDBC Providers**. Set the scope to cell, and click **New**.

> **Note:** The LRS database can safely use a non-XA data source. In many cases so can the LREE database. We recommend that you use an XA data source for the LREE database in case a batch application is deployed that updates other XA resources in the same checkpoint. If you only deploy applications that access the same transactional resource manager, then 2PC is not required; therefore, XA is not required.

Using the wizard, we created a JDBC provider with the following specifications:

- Database type: **DB2**
- Provider type: **DB2 Universal JDBC Driver Provider**
- Implementation type: **XA data source**

Note that the configuration defaults use WebSphere variables that point to the DB2 libraries. To check or set these variables, select **Environment** → **WebSphere Variables**. For example, we needed to create the following variables at the cell level:

- **DB2UNIVERSAL_JDBC_DRIVER_PATH**
- **UNIVERSAL_JDBC_DRIVER_PATH**
- **DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH**

We set the value for each variable to `C:\SQLLIB\java`.

4. Create a J2C authentication alias to ensure that the data source can authenticate to the database server. If the user ID and password required to access the LRS and LREE databases are not the same, you need to create an authentication alias for each.

   To create the authentication alias, navigate to **Security** → **Global Security** → **Authentication** → **JAAS Configuration** → **J2C Authentication data**. Click **New**.

   For our test environment, we created an alias called ITSOLRS containing a user ID and password to be used to access both the DEVLRS and the DEVLREE databases.

5. Create the data source for the LRS database.

   To define a data source, navigate to **Resources** → **JDBC Providers** → ***<your_JDBC_provider>*** → **Data sources**, and click **New.**

   Select the LRS data source, and specify the following options:

   - Name: `LRS_DS`

– JDNI name: `jdbc/lrsched`

– Enable the **Use this Data Source in container managed persistence (CMP)** option.

– Component-managed authentication alias: `ITSOLRS`

– Authentication alias for XA recovery: U**se component-managed authentication alias**.

– Under DB2 Universal data source properties, specify the following:

  - Database name: DEVLRS
  - Driver type: 4
  - Database server: ITSODATA
  - Port number: 50000

6. Repeat the previous step to create the data source for the LREE database, changing the following options:

   – Name: `LREE_DS`
   – JNDI name: `jdbc/lree`
   – Database name: DEVLREE

7. Save your changes to the master configuration.

8. Test LRS and LREE data sources by navigating to **Resources** → **JDBC Providers** → **Data sources**. Select both the LRS_DS and LREE_DS data sources, and click **Test connection**. You should see a message that the test connection operations were successful.

6.5.4 Enable startup beans service

Using the process outlined in section 5.4.5, “Enable the startup beans service on LREE application servers” on page 182, enable the startup beans server for server1. The option can be found by navigating to **Servers** → **Application server** → ***<server1>*** → **Container Services** → **Startup beans service**.

6.5.5 Install and configure the applications

The next step is to install the applications that provide the LRS and LREE functions.

Install the LongRunningScheduler application

To install the LongRunningScheduler application on the server, perform the following from the administrative console:

1. Go to **Applications**, and click **Install New Application**.

2. Select **Local file system**, and click **Browse** to point to *<rad_install>*\runtimes\base_v6\installableApps\LongRunningScheduler.ear.
3. Leave the field context root field empty, and click **Next**.
4. Select **Generate Default Bindings** and choose **Use default virtual host name for Web modules**. Click **Next**.
5. Ignore the Application Security Warnings, and click **Continue**.
6. At step 1 “Select installation options”, ensure that the option **Deploy Enterprise Beans** is selected.
7. Proceed to Step 3 “Provide options to perform EJB Deploy”.
8. Make sure the database type matches your underlying database version, in this case, **DB2UDB_V82**.
9. Proceed to Step 9 “Summary”, and click **Finish** to start the installation. The installation might take a few minutes.
10. Click **Save to Master Configuration** to save the changes.

Configure the long-running scheduler

Before starting the LongRunningScheduler application, we need to configure the database information.

1. From the administrative console, go to **System administration** → **Long-running scheduler**.
2. Change the configuration according as shown in Figure 6-4 on page 202.

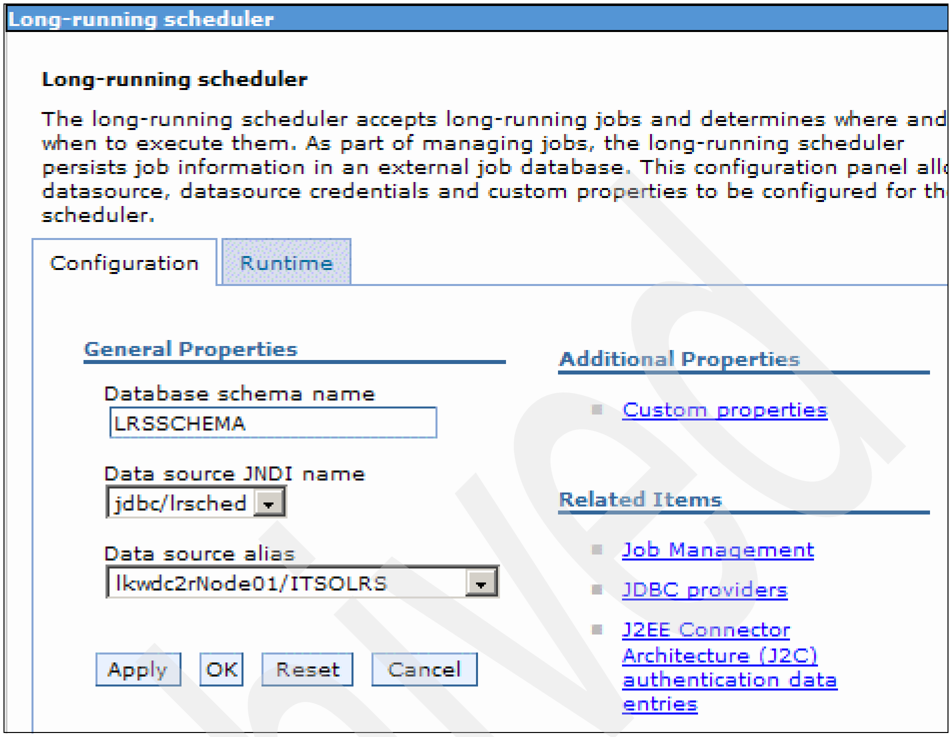


Figure 6-4 Configuring the Long Running Scheduler in the administrative console

a. Set the database schema name to LRSSCHEMA.

b. Select the data source JNDI name to `jdbc/lrsched`.

> **Note:** If you do not see the JNDI name for your LRS database in the drop-down for this option, make sure you defined the data source at the cell scope.

c. Select the LRS J2C authentication alias from the drop-down box.

3. Save the changes to the master configuration.

Install the LREE application

Use the following steps to configure the LREE application:

1. Go to **Applications**, and click **Install New Application**.
2. Select **Local file system**, and click **Browse** to point to *<rad_install>*\runtimes\base_v6\installableApps\LREE.ear.
3. Leave the field context root field empty, and click **Next**.

4. Select **Generate Default Bindings.** Scroll down to the Virtual Host field and select to use default_host as default virtual host name for Web modules. Click **Next**.
5. Accept the Application Security Warning, and click **Continue** to proceed.
6. At step 1 “Select installation options”, ensure that the option **Deploy Enterprise Beans** is selected.
7. Proceed to step 3 “Provide options to perform the EJB Deploy”.

   Select the database type, **DB2UDB_V82** in our case, and specify **LREESCHEMA** as the schema name.
8. Proceed directly to Step 5 “Provide default data source mapping for modules containing 2.x entity beans” by clicking **Step 5** in the vertical blue bar.

   Check the box to the left of BatchJobExecutionEnvironmentEJBs module, select **Per application** in the Resource authorization drop-down, and click **Apply**.

   Check the same module again. In the Specify authentication method section, select **Use default method**, and select ***<yournode>*/ITSOLREE** from the drop-down box. Click **Apply** once more.
9. Proceed to step 11 “Summary”, and click **Finish** to start the deployment.
10. When the installation completes, click **Save to Master Configuration** to change the changes.

Restart the server

Restart the server to ensure that both the LongRunningScheduler and the LREE applications start without errors, as shown in Example 6-1.

Example 6-1 Successful start of LongRunningScheduler and LREE

```
[10/30/06 16:52:44:281 EST] 0000000a ApplicationMg A   WSVR0200I: Starting
application: LongRunningScheduler
...
[10/30/06 16:52:57:031 EST] 0000000a JobSchedulerS I   CWLRB3220I: Long Running Job
Scheduler is initialized
[10/30/06 16:52:58:438 EST] 0000000a ApplicationMg A   WSVR0221I: Application
started: LongRunningScheduler
...
[10/30/06 16:52:58:453 EST] 0000000a ApplicationMg A   WSVR0200I: Starting
application: LREE
...
[10/30/06 16:53:07:078 EST] 0000000a BJEEStartupBe I   CWLRB1400I: Long Running Job
Execution Environment lkwdc2rNode01Cell/lkwdc2rNode01/server1 is initialized
```

```
[10/30/06 16:53:08:328 EST] 0000000a ApplicationMg A   WSVR0221I: Application
started: LREE
```

6.6 Configure the development environment

To develop the long-running applications, you need to add Extended Deployment JAR files in the application's Java build path.

For convenience, we start by defining two classpath variables for the business grid Java binaries as shown in Table 6-1.

Table 6-1 Classpath variables for business grid Java binaries

Classpath variable	Value
BATCH_RUNTIME	C:\RAD60\runtimes\base_v6\lib\batchruntime.jar
COMPUTE_INTENSE_RUNTIME	C:\RAD60\runtimes\base_v6\lib\gridapis.jar

To define these, do the following:

1. In Rational Application Developer V6.0, click **Windows** → **Preferences**.
2. Navigate to **Java** → **Build Path** → **Classpath Variables**.
3. Select **New**, and enter the value for the BATCH_RUNTIME variable from Table 6-1 (Figure 6-5).

.

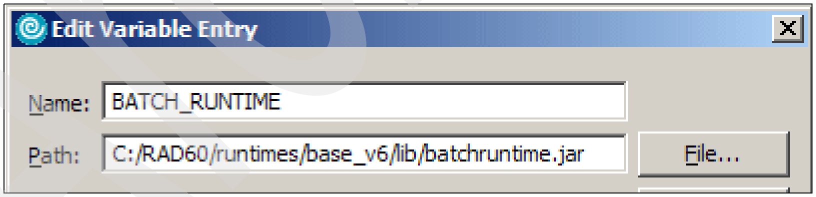


Figure 6-5 Specifying BATCH_RUNTIME Classpath Variable

4. Repeat this step for the COMPUTE_INTENSE_RUNTIME variable.
5. Select **OK** to close the Preferences dialog box.

When you develop a new application, you can use these variables to add the JAR files to the application classpath.

The variable you use depends on the application type:

- To develop batch type applications, use the BATCH_RUNTIME variable to add the JAR files to the Java build path.
- To develop compute-intensive type applications, use both the BATCH_RUNTIME and COMPUTE_INTENSE_RUNTIME variables to add the JAR files to the Java build path.

You will see how this is done in the next sections as we use the sample long-running applications shipped with WebSphere Extended Deployment to test our setup.

6.7 Running the compute-intensive sample application

In this section we import the sample compute-intensive application, SimleCI, to the test environment and submit a job to it.

6.7.1 Import SimpleCI into the workspace

Import the SimpeCI into the workspace and place the necessary JAR files in its build path by adding the classpath variables.

Import EAR file

1. In the J2EE perspective, right-click **Enterprise Applications** and select **Import** → **EAR file**.
2. Select *<rad_install>*\runtimes\base_v6\installableApps\SimpleCI.ear. Ensure that **WebSphere Application Server V6.0** is selected as the target server, and click **Finish**.
3. In the Project Explorer view, navigate to **EJB projects** → **SimpleCIEJB** and expand this module (as shown in Figure 6-6).

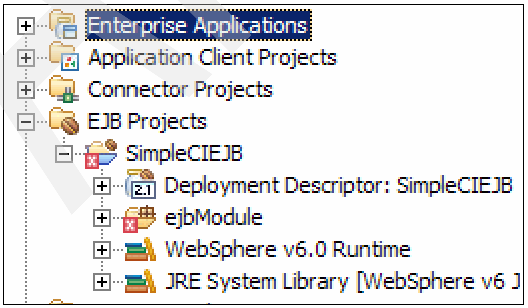


Figure 6-6 SimpleCI EJB Project in after import

Note the x in the red box that indicates that there are errors. More details are seen when navigating to the **Problems** tab in the lower right corner. In this case, the Extended Deployment Java libraries are missing from the Java build path.

4. To resolve the errors, right-click the SimpleCIEJB module and select **Properties**.
5. Select **Java Build Path**, and go to **Libraries**.
6. Click **Add Variable** and select *both* BATCH_RUNTIME and COMPUTE_INTENSE_RUNTIME (use CTRL to select multiple entries). Click **OK**.
7. You should see the results shown in Figure 6-7.

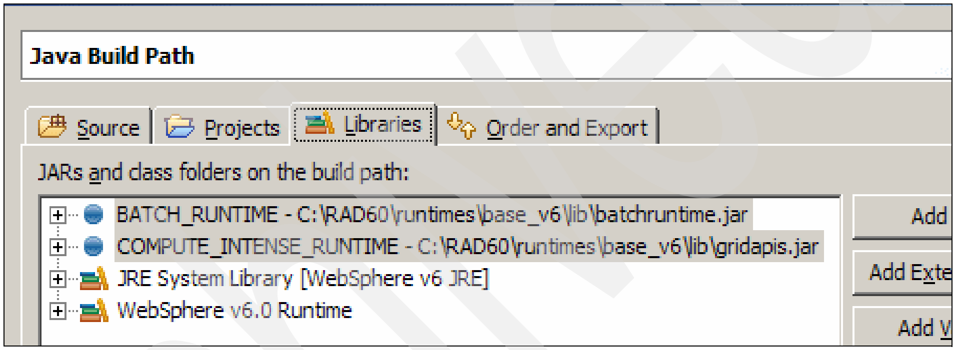


Figure 6-7 Configuring Java Build Path for SimpleCIEJB module

8. Click **OK** again.

   Rational Application Developer will rebuild the SimpleCIEJB project and the problems described earlier should disappear.

Note: Automatic build is enabled by default. If this is not the case, select **Project** → **Build Automatically** to enable this.

6.7.2 Modify the xJCL

Before we submit a job to the SimpleCI application, we need to modify the xJCL XML file.

Import xJCL into Rational Application Developer

For convenience, we want to import the xJCL files into Rational Application Developer. This allows us to edit the XML using the XML editor.

If during this process you are asked if you want to enable the XML Development capability, select Yes.

Start by creating a directory on the file system to hold the xJCL files.

1. Create a directory C:\xJCL\SimpleCI.
2. Copy the SimpleCI xJCL sample file to this directory. This file resides in *<rad_install>*\runtimes\base_v6\longRunning\SimpleCIxJCL.xml.

Now create a simple project that points to the directory above.

1. In Rational Application Developer, select **File** → **New** → **Project**.
2. Select **Simple** → **Project,** and click **Next**.
3. Enter a project name, for example SimpleCI_xJCL, and click **Finish**.

Now create a folder in the project that points to the xJCL.

1. Right-click the new project and select **New** → **Other** → **Simple** → **Folder**, and click **Next**.
2. Specify the following, as seen in Figure 6-8 on page 208:
   - Folder name: SimpleCI
   - Click **Advanced** and select **Link to folder in the filesystem**. Type *C:\xJCL\SimpleCI* as the folder.

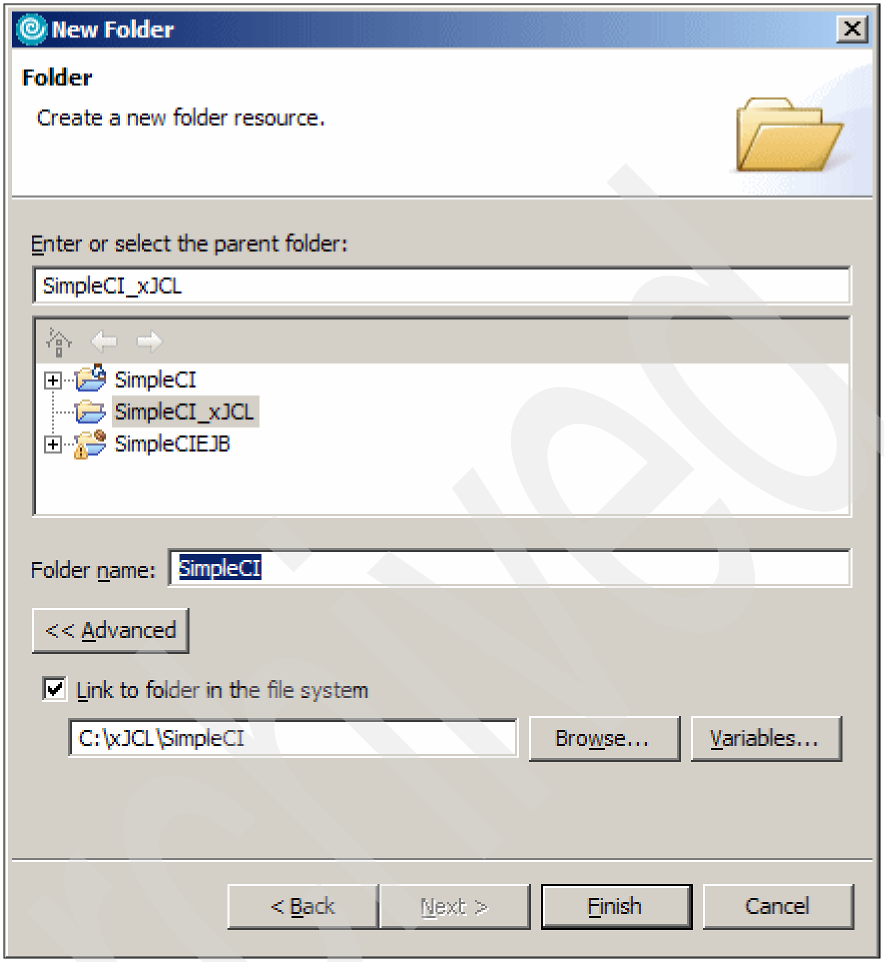


Figure 6-8 Creating a link to a folder in the filesystem in Rational Application Developer

3. Click **Finish**.

Modify the xJCL

Now we can easily edit the xJCL files from Rational Application Developer. We need to change the job name and property values.

The job name in the xJCL must match the name of the application. When deploying the SimpleCI application using Rational Application Developer, the name of the application is the same as the name of the project. In this case, the name of the SimpleCI application is "SimpleCI".

1. From the project SimpleCI_xJCL, expand the SimpleCI folder.

2. Double-click **SimleCIxJCL.xml** to open this file for editing.
3. Make the following changes as shown in Example 6-2.
   – Look for the <job> tag and ensure that the name specified matches the name of the Enterprise application project (SimpleCI).
   – Look for the <props> tag and change the value for the outputFileName property to point to a valid path on your system, for example C:\temp\SimpleCI.log.
   – You can also change the value for the calculationTimeInSecs property to specify how long SimpleCI continues to execute the calculation.

Example 6-2 SimpleCI xJCL XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="SimpleCI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <jndi-name>ejb/com/ibm/ws/ci/SimpleCIEJB</jndi-name>
   <job-step name="Step1">
      <classname>com.ibm.websphere.ci.samples.SimpleCIWork</classname>
       <props>
           <prop name="calculationTimeInSecs" value="30" />
           <prop name="outputFileName" value="C:\temp\SimpleCI.log" />
       </props>
   </job-step>
</job>
```

6.7.3 Verify SimpleCI

Deploy the SimpleCI application to the server, and submit a job using the modified xJCL file.

Deploy SimpleCI

Use the following steps to deploy the application:

1. Start the server.
2. Right-click the server and select **Add and remove project**.
3. Select **SimpleCI** from the list of available projects, and click **Add**.
4. Click **Finish** to start the deployment.

Submit a job

Now we are ready to submit a job using the xJCL XML file we just modified.

1. Open a command prompt.

2. Change the directory to *<rad_install>*\runtimes\base_v6\bin.
3. Run following command to submit a job:

   `lrcmd -cmd=submit -xJCL=C:\xJCL\SimpleCI\SimpleCIxJCL.xml -host=localhost -port=9080`

   This command assumes that you are submitting the job from the same system where the server is running and that the WC_defaulthost port for server1 is 9080.

Example 6-3 Command to submit a job to SimpleCI in the WebSphere Test Environment.

```
C:\RAD60\runtimes\base_v6\bin>lrcmd -cmd=submit
-xJCL=C:\xJCL\SimpleCI\SimpleCIxJCL.xml -host=localhost -port=9080

CWLRB4940I: com.ibm.ws.batch.wsbatch : -cmd=submit
-xJCL=C:\xJCL\SimpleCI\SimpleCIxJCL.xml -host=localhost -port=9080

CWLRB4960I: Mon Oct 30 13:46:12 EST 2006 : com.ibm.ws.batch.wsbatch : Job
[SimpleCI:0] is submitted.
```

4. Monitor the server messages using the Console view. The output should look similar to Figure 6-9.

> **Note:** Alternatively you might want to monitor the SystemOut.log directly:
>
> `<rad_install>\runtime\base_v6\profiles\default\logs\server1`

```
ApplicationMg A   WSVR0221I: Application started: SimpleCI
ServletWrappe A   SRVE0242I: [LongRunningScheduler] [/LongRunningJobSchedulerWeb
WSChannelFram A   CHFW0019I: The Transport Channel Service has started chain htt
ServletWrappe A   SRVE0242I: [LREE] [/EndpointWebService] [BatchGridDiscriminato
ComponentName W   CNTR0063W: A reference to an EJB could not be found in the dep
SystemOut     O Mon Oct 30 13:46:12 EST 2006: SimpleCI application starting...
SystemOut     O -->Will loop processing a variety of math functions for approxim
SystemOut     O Mon Oct 30 13:46:42 EST 2006: SimpleCI application complete!
SystemOut     O -->Actual Processing time = 30.0 seconds!
```

Figure 6-9 Server messages for SimpleCI

6.8 Running the batch sample application

Next, we will import the sample batch application, PostingsSample, to the test environment and submit a job to it.

6.8.1 Import PostingsSample into Rational Application Developer

The following sections show how to Import the PostingsSample into the workspace and place the necessary JAR files in its Java build path.

Import the EAR file

1. In the J2EE perspective, right-click **Enterprise Applications** and select **Import** → **EAR file**.
2. Select *<rad_install>*\runtimes\base_v6\installableApps\PostingsSample.ear. Ensure that **WebSphere Application Server V6.0** is selected as the target server, and click **Finish**.
3. Navigate to **EJB projects** → **PostingsSampleEJBs** and expand this module, as shown in Figure 6-10.

   Note the x inside the red box that indicates that errors exist. More details are visible when navigating to the **Problems** tab in the lower right corner. The problems are a result of the Extended Deployment Java libraries for batch applications that are missing from the Java build path.

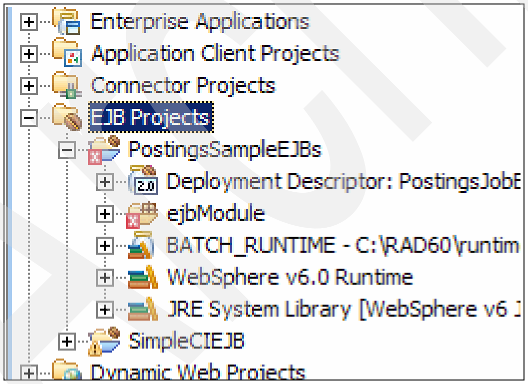


Figure 6-10 PosintgsSampleEJBs EJB Project after importing PostingsSample

4. To resolve this, right-click the PostingsSampleEJB module and select **Properties**.
5. Select **Java Build Path**, and go to **Libraries**.
6. Click **Add Variable**, and select BATCH_RUNTIME. Click **OK**.

7. You should see the results shown in Figure 6-11. Click **OK** again.

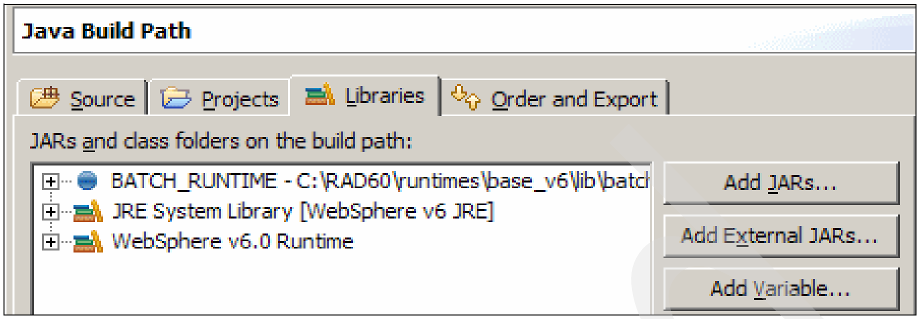


Figure 6-11 Configuring Java Build Path for PostingsSampleEJB module

8. Rational Application Developer will rebuild the PostingsSampleEJB project (assuming automatic build is enabled), but note that there are still problems remaining. We will take care of those problems next by generating the deploy code for the EJBs.

Generate the deploy code for CMP EJBs

Before we generate deploy code for the CMP EJBs, we need to configure a new EJB-to-RDB mapping. The module comes with mappings for DB2UDBOS390_V7_1 and DB2UDBOS390_V8_1, but we need a mapping for DB2 V8.2 on Windows (DB2UDBNT_V82_1).

1. To create a new mapping, right-click PostingsSampleEJBs and select **EJB to RBD Mapping → Generate Map**.
2. Select **Create a new backend folder**, and click **Next**.
3. Select **Top-Down** as the mapping option, and click **Next**.
4. Provide the following top-down mapping options, as shown in Figure 6-12 on page 213.

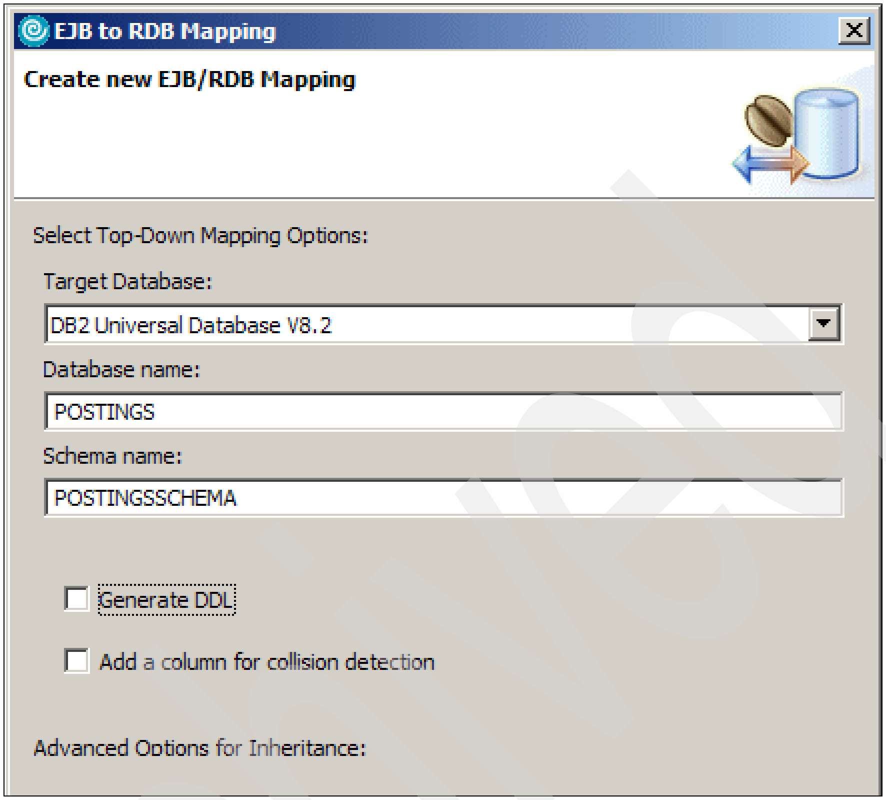


Figure 6-12 Top-Down Mapping options

   - Target Database is DB2 Universal Database V8.2
   - Database name is `POSTINGS`
   - Schema name is `POSTINGSSCHEMA`
   - Deselect **Generate DDL**
5. Select **Finish**. The new map opens in the editor. Close it.
6. Expand PostingsSampleEJBs and double-click the deployment descriptor to open it.
7. On the Overview tab, scroll down to WebSphere Bindings.
8. Under Backend ID, select **DB2UDBNT_V82_1,** and save the deployment descriptor.

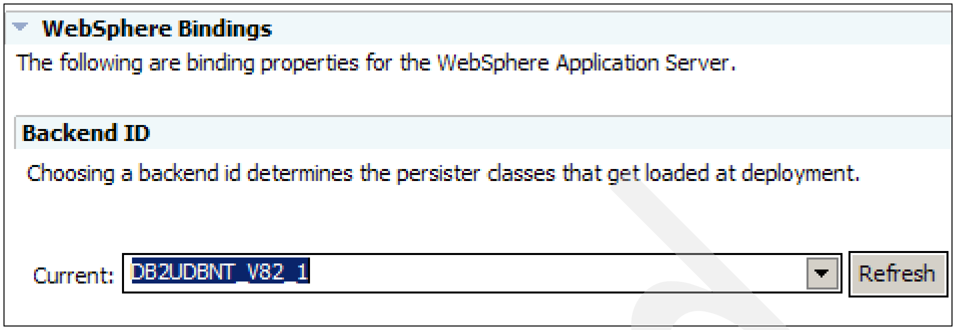


Figure 6-13 Changing the Backend ID in the deployment descriptor

9. Now we can generate the deploy code for the EJB module. Right-click PostingsSampleEJBs and select **Deploy**. When finished, the PostingsSample enterprise application is ready for deployment.

Import the xJCL into the workspace

Refer to "Import xJCL into Rational Application Developer" on page 206 for guidance on creating a new folder to hold the xJCL file and for instructions on importing the xJCL XML file for the PostingsSample application.

1. Copy *<rad_install>*\runtimes\base_v6\longRunning\PostingsSamplexJCL.xml to C:\xJCL\PostingsSample.
2. Create a simple project and folder, linking the folder to the file system containing the xJCLfile.

6.8.2 Create the database and data source for PostingsSample

The PostingsSampleEJBs EJB module requires a relational database. We need to create this database and create a data source in the server in order for PostingsSample to run.

1. Create the DB2 database:

   Logon to the DB2 server and create a database called POSTINGS. Modify *<rad_install>*\runtimes\base_v6\longRunning\CreatePostingsTablesDB2.ddl so that it connects to the new database, and use this to create your database tables.

2. Configure the J2C authentication alias:

   Configure a J2C authentication alias with the user ID and password needed to access the database. In our sample, we named this alias POSTINGS.

3. Define the data source:

   We used the following to create the data source for our POSTINGS database:

   – JDBC provider: **DB2 Universal JDBC Driver Provider (XA)**.
   – Name: `Postings`
   – JNDI name: `jdbc/postings`
   – Enable **Use this Data Source in container managed persistence (CMP)**
   – Component-managed authentication alias: POSTINGS (created in Step 2)
   – Authentication alias for XA recovery: **Use component-managed authentication alias**.
   – Database name: POSTINGS
   – Driver type: 4
   – Database server: ITSODATA
   – Port number: 50000

4. Test the data source Postings to ensure that it was set up correctly.

6.8.3 Verify PostingsSample

Next, deploy PostingsSample application to the server and submit a job.

Deploy PostingsSample application

Use the following steps to deploy the application:

1. Start the server.
2. Right-click the server in the Servers view and select **Add and remove project**.
3. Select **PostingsSample** from the list of available projects, and click **Add**
4. Click **Finish** to start the deployment.

Modify the xJCL

Before we submit a job to the PostingsSample application, we need to modify the xJCL XML file. Open the PostingsSamplexJCL.xml file for editing, as described in “Import xJCL into Rational Application Developer” on page 206. Make the changes shown in Example 6-4 on page 216.

1. Look for the <job> tag and ensure that the name specified matches the name of the Enterprise application project, PostingsSample.

2. Look for the <props> tag and change the value for the FILENAME property to point to a valid path on your system, for example C:\temp\postings.

> **Note:** Please note that this application has multiple job steps, so you need to change the FILENAME property both in Step1 and Step2!

Example 6-4 PostingsSample xJCL XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="PostingsSample" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <jndi-name>ejb/com/ibm/websphere/samples/PostingsJob</jndi-name>
...
   <job-step name="Step1">
      <jndi-name>ejb/DataCreationBean</jndi-name>
         <batch-data-streams>
            <bds>
               <logical-name>myoutput</logical-name>
               <props>
                     <prop name="FILENAME" value="c:\temp\batchjoboutput\postings" />
            </props>
         </bds>
      </batch-data-streams>
...
   </job-step>
</job>
```

Submit a job

Now we are ready to submit a job using the xJCL file we just modified.

1. Open a command prompt.
2. Change the directory to *<rad_install>*\runtimes\base_v6\bin
3. Run following command to submit a job:

   `lrcmd -cmd=submit -xJCL=C:\xJCL\Postings\postingSampleXJCL.xml -host=localhost -port=9080`

Example 6-5 Command to submit a job to SimpleCI in the WebSphere Test Environment.

```
C:\RAD60\runtimes\base_v6\bin>lrcmd -cmd=submit
-xJCL=C:\xJCL\Postings\postingSampleXJCL.xml -host=localhost -port=9080

CWLRB4940I: com.ibm.ws.batch.wsbatch : -cmd=submit
-xJCL=C:\xJCL\Postings\postingSampleXJCL.xml -host=localhost -port=9080
```

```
CWLRB4960I: Mon Oct 30 16:18:50 EST 2006 : com.ibm.ws.batch.wsbatch : Job
[PostingsSample:40] is submitted.
```

4. Monitor the server messages in the Console view. The output should look similar to the output shown in Figure 6-14.

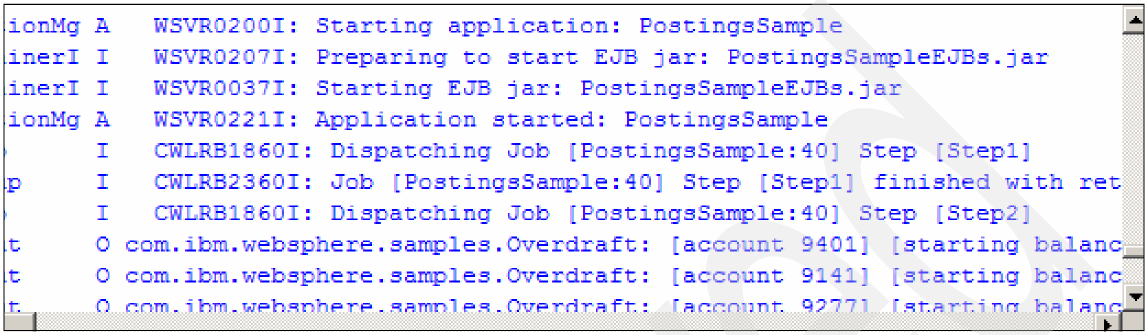


Figure 6-14 Server output for PostingsSample

7

Building batch applications

This chapter is intended for developers who need to build batch applications that run on WebSphere Extended Deployment. The topics covered in this chapter are as follows:

- Overview
- Building a sample batch application
- Using a batch data stream in a batch application
- Using JDBC in a batch data stream
- Batch application flow
- Batch application programming model
- Batch application syntax in xJCL

Details on how to setup a development environment for WebSphere Extended Deployment long-running applications are provided in Chapter 6, “Configuring a long-running development environment” on page 189.

7.1 Overview

A batch application is packaged as a standard EJB module inside a J2EE enterprise application. This application is deployed together with the LREE enterprise application onto the same dynamic cluster.

Section 7.1.1, “Components in batch applications” on page 220 describes the various components that a batch application consists of. The interaction of those components is described in section 7.1.2, “Interaction with the long-running execution environment” on page 222.

7.1.1 Components in batch applications

Every batch application consists of exactly one *Batch Job Controller bean*. This is a stateless session bean whose implementation is provided by the Extended Deployment runtime (see Table 7-1 on page 221). In other words, the bean only needs to be declared in the deployment descriptor of the EJB module. Upon deployment, the JNDI name of the Batch Job Controller bean should be used as a reference in the xJCL associated with a job.

Each batch application can consist of multiple steps that need to be carried out. A *Batch Job Step bean* is associated with each step of the batch job, as defined in xJCL. These Batch Job Step beans are actually CMP entity beans that implement the BatchJobStepInterface. The implementation of this interface provides the business logic of the batch job step.

Each Batch Job Step bean can use zero or more *batch data stream* objects for the input and output of data. A batch data stream object is a Plain Old Java Object (POJO) that implements the BatchDataStream interface provided by Extended Deployment. A sample implementation of a batch data stream is described in 7.4, “Using JDBC in a batch data stream” on page 251.

Figure 7-1 on page 221 shows the relationship between the various components of a batch application for Extended Deployment. Note that a batch application can have more than one Batch Job Step bean; however, having just one greatly simplifies the diagram.

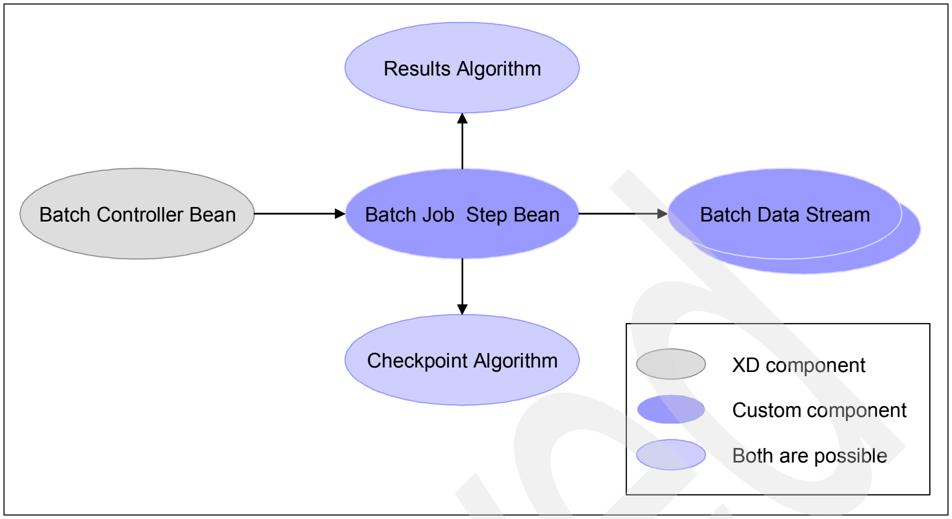


Figure 7-1 Various components in one Extended Deployment batch application

As shown in Figure 7-1, the long-running execution environment interacts with two algorithms while processing the batch job:

- Checkpoint algorithm
- Results algorithm

These algorithms *can* be implemented for a batch application, but Extended Deployment provides ready-to-use algorithms out-of-the-box. The out-of-the-box supplied implementations are listed in Table 7-1, together with all the implementation classes for the Batch Job Controller bean. The actual class files are packaged in batchruntime.jar, which can be found in the lib directory of Extended Deployment.

Table 7-1 J2EE and Java components that come with Extended Deployment

Component	Type	Implementation class

Batch Job Controller bean	Stateless session bean	`com.ibm.ws.batch.BatchJobControllerBean`
		`com.ibm.ws.batch.BatchJobControllerHome`
		`com.ibm.ws.batch.BatchJobController`

Time-based checkpoint algorithm	POJO	`com.ibm.wsspi.batch.checkpointalgorithms.timebased`
Record-based checkpoint algorithm	POJO	`com.ibm.wsspi.batch.checkpointalgorithms.recordbased`

Component	Type	Implementation class
Job sum results algorithm	POJO	`com.ibm.wsspi.batch.resultsalgorithms.jobsum`

Table 7-2 summarizes the J2EE and Java components that can be implemented for an Extended Deployment batch application. Standard J2EE development tools can be used to develop and package the batch application.

Table 7-2 J2EE and Java components that can be implemented for a batch application.

Component	Type	Interface implementation
Batch Job Step bean	CMP entity bean	`com.ibm.websphere.batch.BatchJobStepInterface`
Batch data stream	POJO	`com.ibm.websphere.batch.BatchDataStream`
Checkpoint algorithm	POJO	`com.ibm.wsspi.batch.CheckpointPolicyAlgorithm`
Results algorithm	POJO	`com.ibm.wsspi.batch.ResultsAlgorithm`

In this book we chose to use Rational Application Developer V6.0 as the development tool. Chapter 6, "Configuring a long-running development environment" on page 189, contains information about how to set up a development and test environment for batch applications using Rational Application Developer V6.0.

7.1.2 Interaction with the long-running execution environment

Let us assume that we are dealing with a batch application that has only one job step. Figure 7-2 on page 223 shows how the components of such an application interact with the LREE

> **Note:** When we talk about the LREE, we are referring to the LREE enterprise application and the associated business grid runtime components.

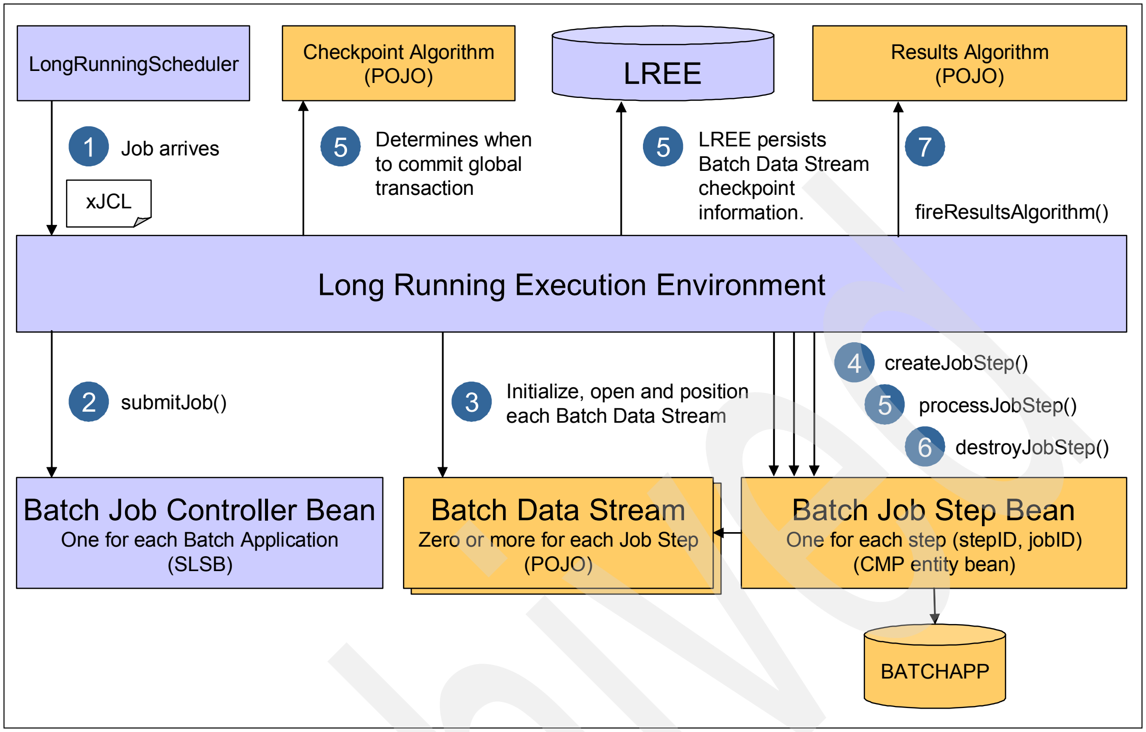


Figure 7-2 Interaction of a batch application with the long-running execution environment

1. The *LongRunningScheduler* dispatches a job to one of the LREE application servers.
2. The LREE looks up the *Batch Job Controller bean* of the batch application using the JNDI name provided in the xJCL. It will call the submitJob() method on the controller bean.
3. The LREE prepares the *batch data streams* associated with the only job step present in this application. Each batch data stream is initialized, opened and positioned at the correct checkpoint.
4. Now the LREE will look up the Batch Job Step bean using the JNDI reference specified in the xJCL for the job step. It then prepares the Batch Job Step bean for running its business logic by calling createJobStep(). In this step, the Batch Job Step bean also obtains handles to the batch data streams.
5. Once everything is prepared, the business logic can be executed in the batch loop. The business logic is in the processJobStep() method of the Batch Job Step bean, which will be called by the LREE. When running the batch loop, the LREE interacts with the *checkpoint algorithm* associated with the job step. This determines when the LREE will commit a global transaction and

start a new one. When a transaction boundary is reached, the LREE persists the current checkpoint of each batch data stream to the LREE database.

6. When the Job Step finishes, the LREE calls the destroyJobStep() method on the Batch Job Step bean. This gives the Batch Job Step bean the opportunity to release resources.
7. Finally the LREE calls the fireResultsAlgorithm() method on the *results algorithm* associated with the job step. The result obtained is returned to the LongRunningScheduler.

7.2 Building a sample batch application

As discussed in the previous section, a batch application for Extended Deployment consists of various components that are packaged as a J2EE EAR file. This section shows how to build a “Hello World!” style batch application using Rational Application Developer V6.0. It is a good starting point for building a real batch application, although more steps are required for that. The actual business logic we implement is very simple and does not use batch data streams. We also do not implement a custom checkpoint algorithm or a results algorithm.

The Extended Deployment batch programming model uses a CMP entity bean for the implementation of the Batch Job Step bean. This entity bean must implement com.ibm.websphere.batch.BatchJobStepInterface, and this is where the business logic needs to be implemented. Given these constraints, there are two possible implementations:

- Implementation in the actual bean class itself:

```
public abstract class SampleStep1Bean implements
javax.ejb.EntityBean, com.ibm.websphere.batch.BatchJobStepInterface
```

- Implementation of the BatchJobStepInterface in a separate Java class, for example com.ibm.sample.batch.SampleStep1. In this case, the actual bean class extends this class:

```
public abstract class SampleStep1Bean extends
com.ibm.itso.sample.batch.SampleStep1 implements
javax.ejb.EntityBean
```

For this sample application, we opted for the second option, which provides a way to implement the business logic outside of the Batch Job Step bean. Future releases of Extended Deployment are likely to provide a batch programming model that does not require implementation of CMP entity beans. Hence choosing the second option would ease a potential future migration as well.

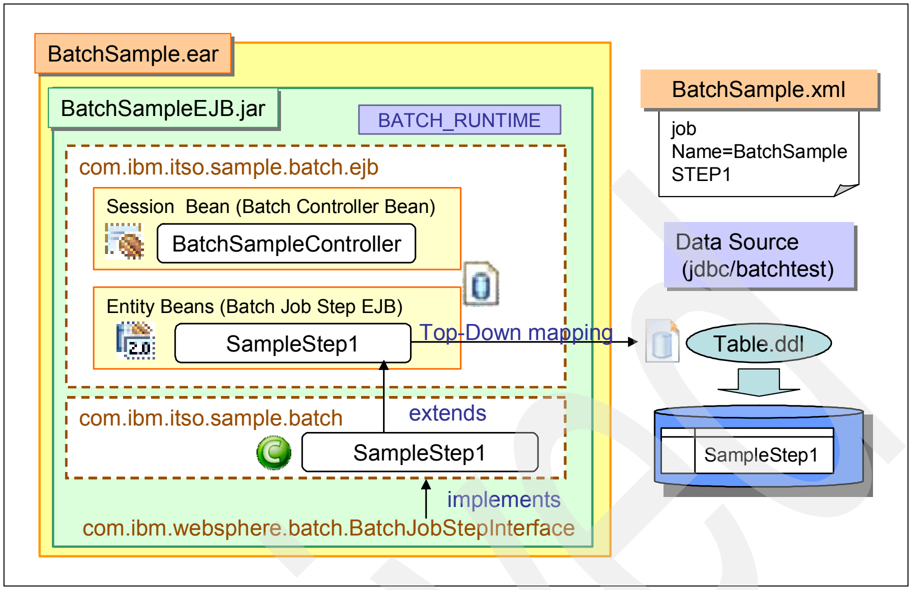


Figure 7-3 Sample batch application components and packaging

Below is an overview of the steps required to build and package this sample batch application. The different components involved are shown in Figure 7-3.

- Create an enterprise project and EJB project.
- Create the Batch Job Controller bean.
- Create the implementation class for the Batch Job Step bean.
- Create a Batch Job Step bean.
- Create the CMP mapping and database table.
- Configure EJB deployment descriptor.
- Implement sample business logic.

7.2.1 Create an enterprise project and EJB project

If you have not already done so, start Rational Application Developer V6.0 and switch to the J2EE perspective. We start by creating a new EJB project and enterprise project to hold our new batch application.

1. Right-click “EJB Projects” and select **New** → **EJB Project**.
2. Specify the following options for the new EJB project as shown in Figure 7-4 on page 226:
   a. Enter a name for the project, for example, `BatchSampleEJBs`.

b. Select **EJB version 2.0**.

c. Check the box labeled **Add module to an EAR project**.

d. Enter a name for the EAR project, for example, `BatchSample`.

e. Deselect the box labeled **Create an EJB Client Jar project...**.

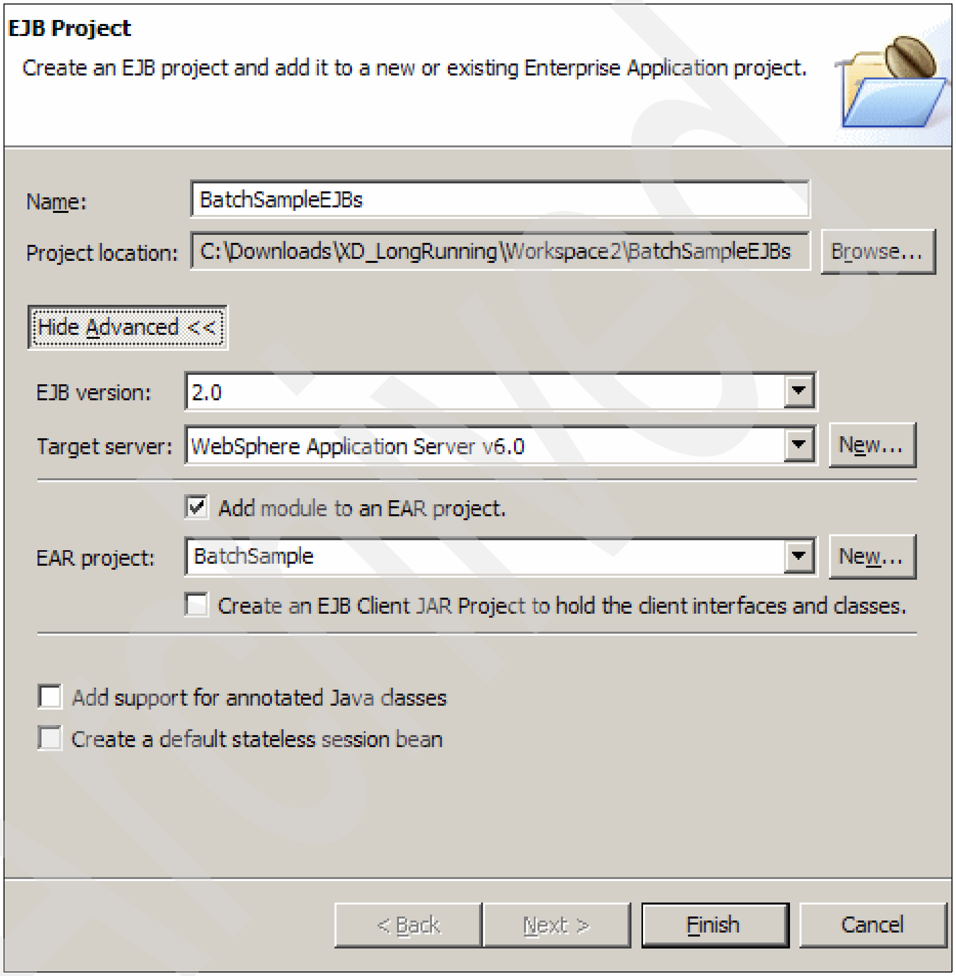


Figure 7-4 Options for the new BatchSampleEJBs EJB project

3. Click **Finish**.
4. Right-click the new EJB project and choose **Properties**.
5. In the Properties panel, select **Java Build Path**, and go to the Libraries tab.

6. Click **Add Variable**, and select **BATCH_RUNTIME** from the list, as shown in Figure 7-5. This step assumes that you set up your development environment to include this variable. See section 6.6, “Configure the development environment” on page 204.

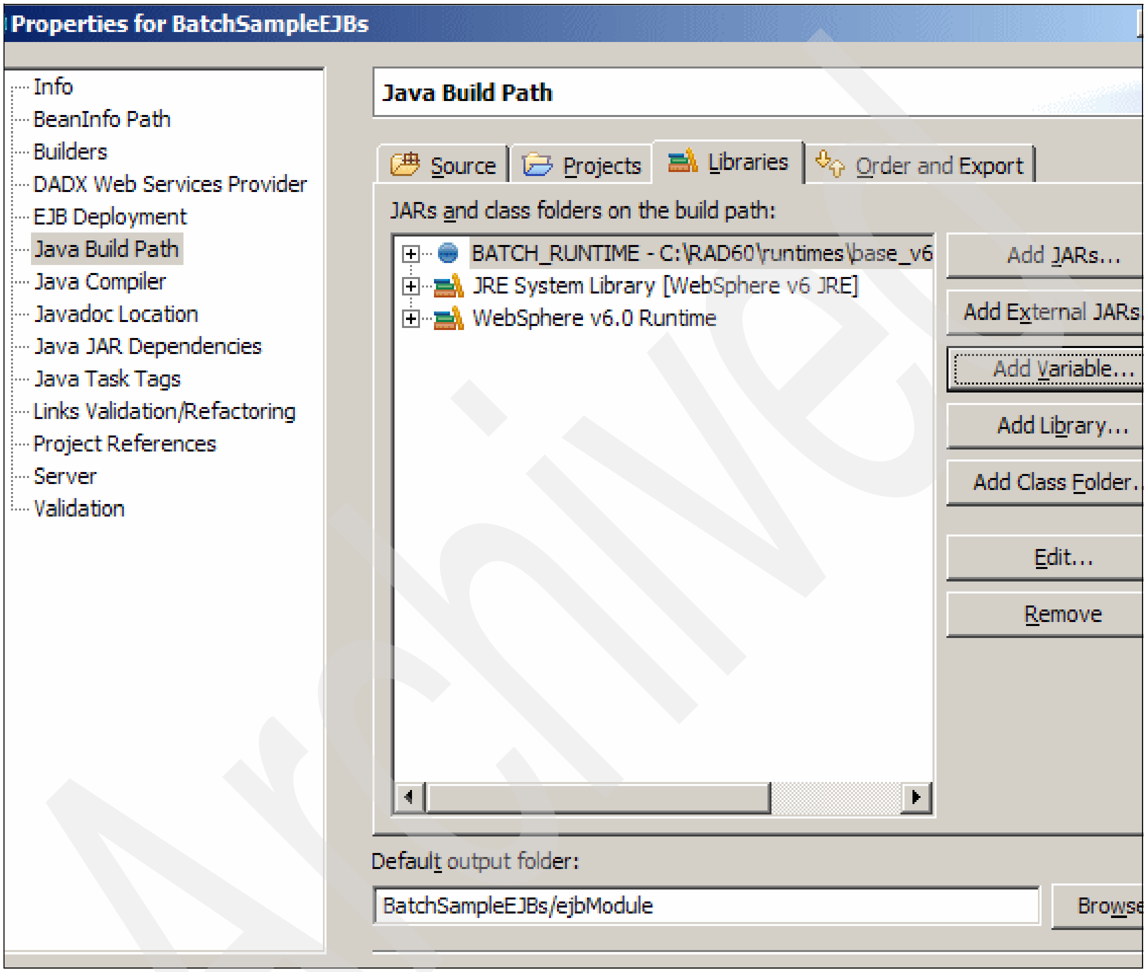


Figure 7-5 Configuring the Java build path of the new EJB project

7. Choose **OK** to continue.
8. Right-click the EJB project and select **New** → **Package** to define a package to hold the EJB classes.
9. Accept the default source folder and specify a package name as shown in Figure 7-6 on page 228, for example:

```
com.ibm.itso.sample.batch.ejb
```

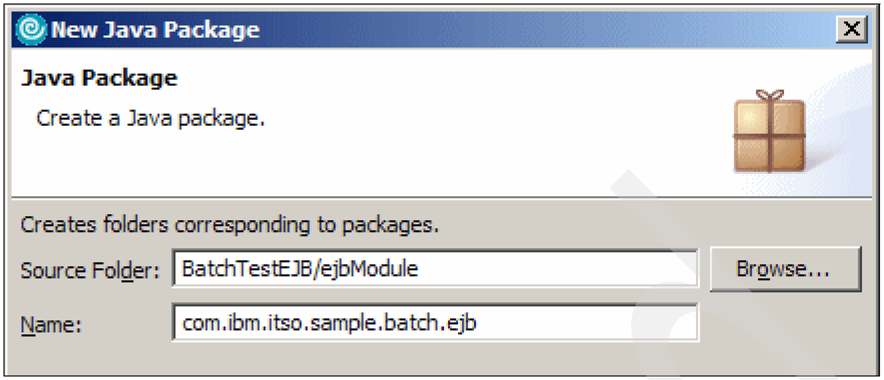


Figure 7-6 Creating a new Java package to hold the new EJB classes

10. Click **Finish**.

7.2.2 Create the Batch Job Controller bean

Declare the Batch Job Controller bean. Note that we do not need to implement this bean, we only have to declare it once in every batch application and point to the implementation classes that come with Extended Deployment.

1. Expand the deployment descriptor in your EJB project.
2. Right-click **Session beans** and select **New** → **Session Bean**.
3. Specify the following options for the new bean, as shown in Figure 7-7 on page 229:
   - Ensure that **Session bean** is selected.
   - Specify a name for the bean, for example, `BatchSampleController`.
   - Specify a package for the bean, for example, `com.ibm.itso.samples.batch.ejb`.

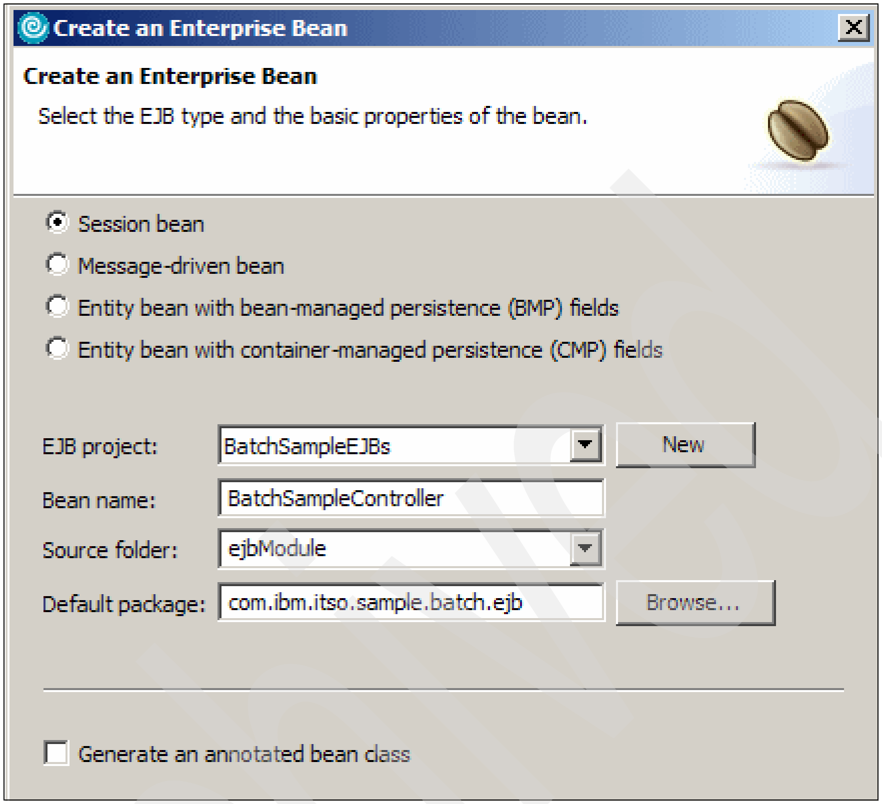


Figure 7-7 Specifying options for the BatchSampleController stateless session bean

4. Click **Next**.
5. Specify additional options for the bean, as shown in Figure 7-8 on page 230:
   – For the transaction type, select **Bean**.
   – Specify **com.ibm.ws.batch.BatchJobControllerBean** for the bean class.
   – Ensure that **Remote client view** is checked.
   – Specify **com.ibm.ws.batch.BatchJobControllerHome** for the remote home interface class.
   – Specify **com.ibm.ws.batch.BatchJobController** for the remote interface class.
   – Ensure that **Local client view** is not checked.

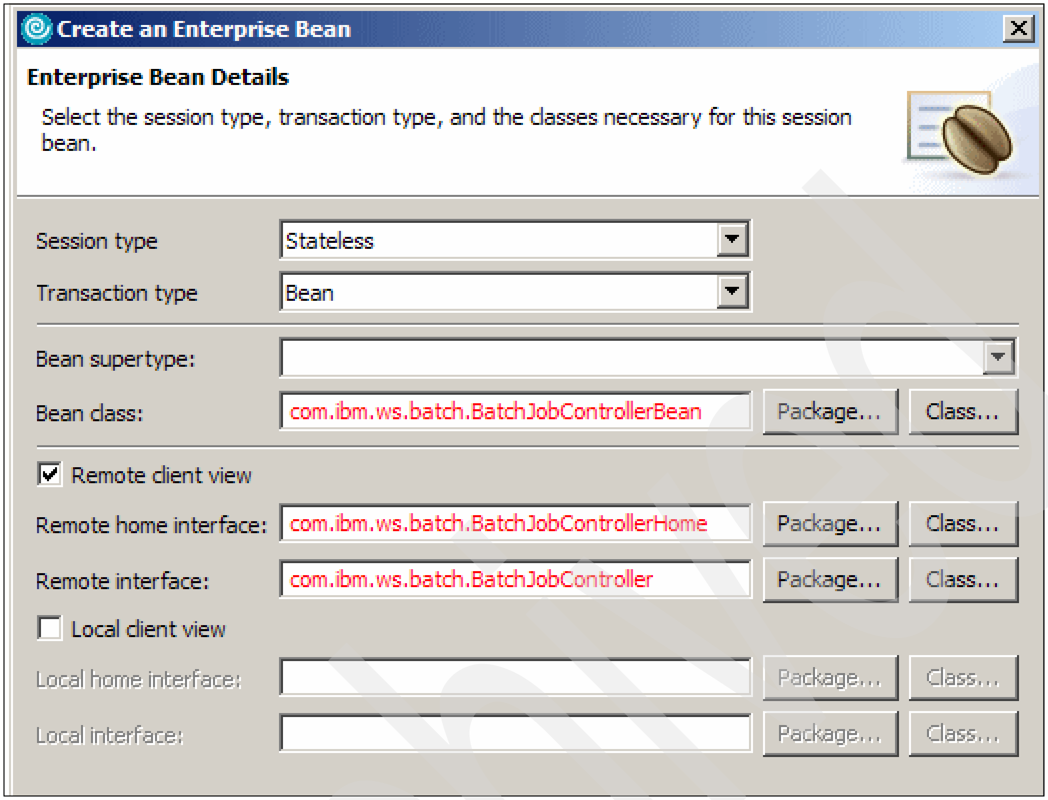


Figure 7-8 Specifying the implementation classes for the new bean

6. Select **Finish** to create the bean.

7.2.3 Create the implementation class for the Batch Job Step bean

Next, we create a simple Java class that implements BatchJobStepInterface. This is the class that will implement the business logic of our batch application. When we create our Batch Job Step bean, we will select this class as its superclass. This approach separates our business logic from the enterprise bean implementation. Hence we suggest creating this class in a separate Java package.

> **Note:** As discussed in the beginning of 7.2, "Building a sample batch application" on page 224, this is not the only way to build a batch application. You can also implement the business logic directly in the Batch Job Step bean. More details can be found in section 7.6.1, "Batch Job Step bean" on page 269.

1. Expand ejbModule in the EJB project.
2. Right-click the package (for example com.ibm.itso.samples.batch) and select **New → Class**.
3. Specify the following options for the new class, as shown in Figure 7-9:
   - Specify a package for the bean, for example `com.ibm.itso.samples.batch`.
   - Specify a name for the class, for example `SampleStep1`.
   - Under Interfaces, choose **Add** and add **`com.ibm.websphere.batch.BatchJobStepInterface`** as an interface.

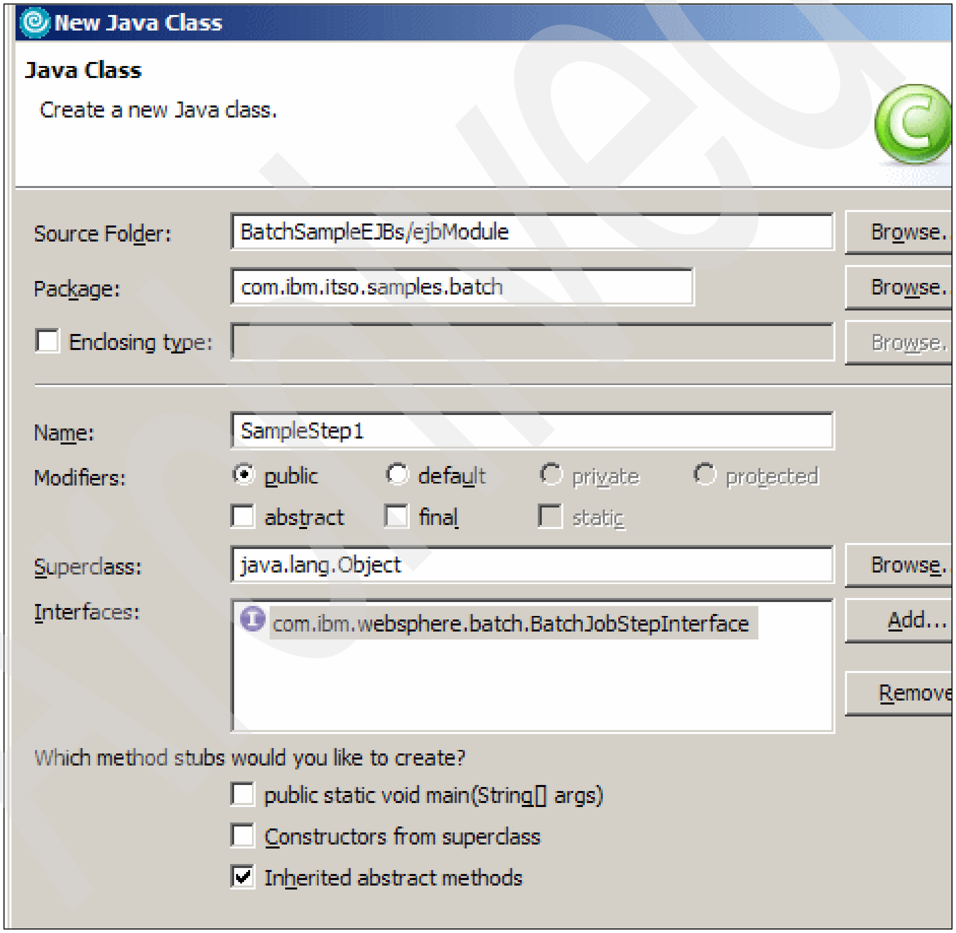


Figure 7-9 Creating the implementation class for the Batch Job Step bean

4. Click **Finish**.

7.2.4 Create a Batch Job Step bean

We are now ready to create our Batch Job Step bean. As discussed in the previous section, this bean uses our implementation class com.ibm.itso.samples.batch.SampleStep1 as a superclass.

> **Note:** A batch application can have multiple Batch Job Step beans, in which case you need to repeat this step several times. Make sure to create a separate implementation superclass for each new Batch Job Step bean.

1. Expand the deployment descriptor in your EJB project.
2. Right-click **Entity beans** and select **New** → **Entity Bean**.
3. Specify the following options for the new bean, as shown in Figure 7-10 on page 233:
   - Ensure that **Entity bean with container-managed persistence (CMP) fields** is selected.
   - Specify a name for the bean, for example `SampleStep1`.
   - Specify a package for the bean, for example `com.ibm.itso.samples.batch.ejb`.
   - Ensure that **CMP Version 2.x** is selected.

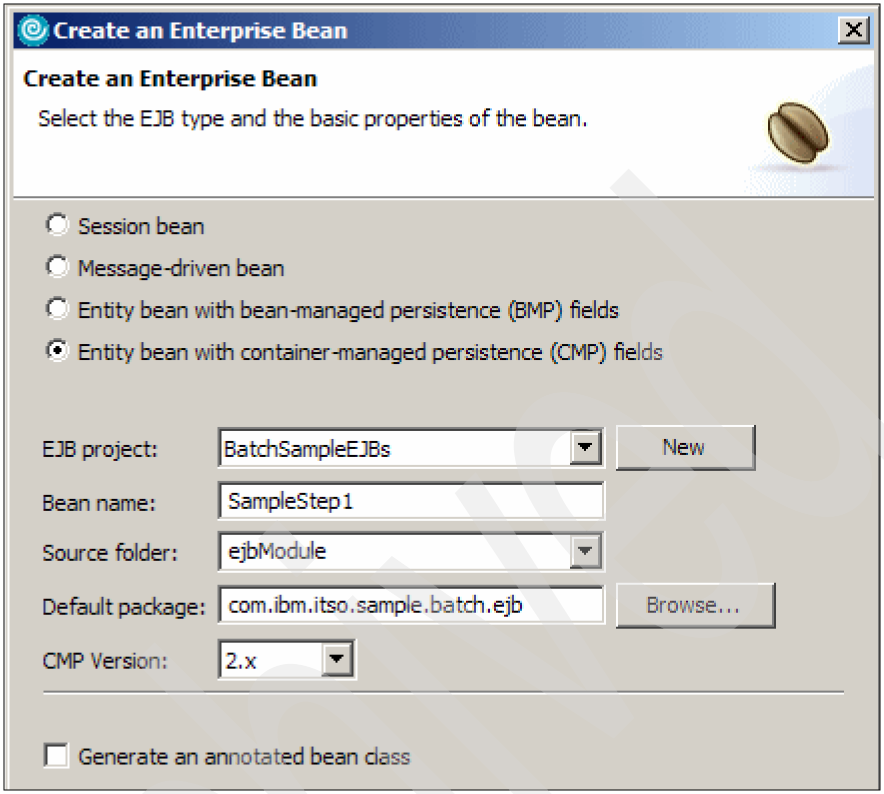


Figure 7-10 Specifying options for the SampleStep1 CMP entity bean

4. Choose **Next** to continue.
5. Specify additional options for the new enterprise bean:
   - Ensure that **Local client view** is checked.
   - Specify **com.ibm.websphere.batch.BatchJobStepLocalHomeInterface** for the local home interface:
   - Specify **com.ibm.websphere.batch.BatchJobStepLocalInterface** for the local interface.
   - Remove the **id : java.lang.Integer** CMP attribute.
   - Specify **com.ibm.websphere.batch.BatchJobStepKey** for the key class.
   - Now add two new CMP attributes
     - Select **Add**.
     - For the name of the first CMP attribute, specify **`jobID`**.
     - Enter **`java.lang.String`** for the type.

- Check the **Key field** box, and click **Apply**, as shown in Figure 7-11.

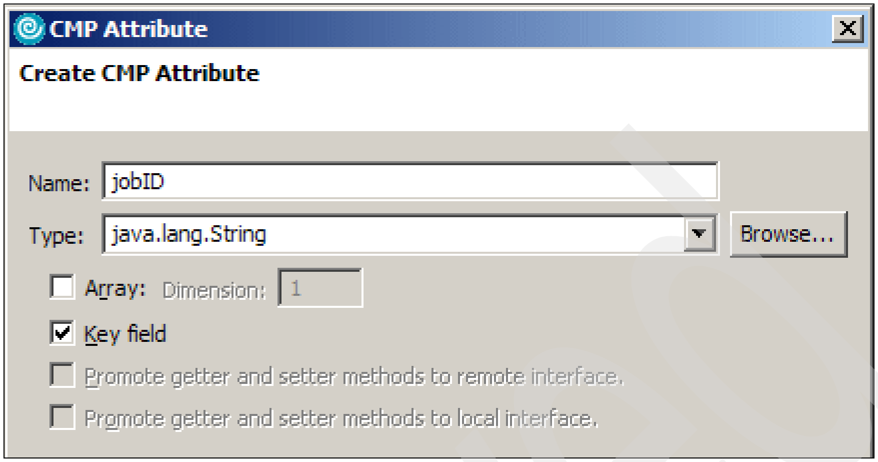


Figure 7-11 Creating a CMP attribute for the SampleStep1 CMP entity bean

- Now enter **`stepID`** for the name of the second CMP attribute.
- Enter **`java.lang.String`** for the type.
- Check the **Key field** box and click **Apply**.
- Click **Close.**

The results should look like Figure 7-12 on page 235.

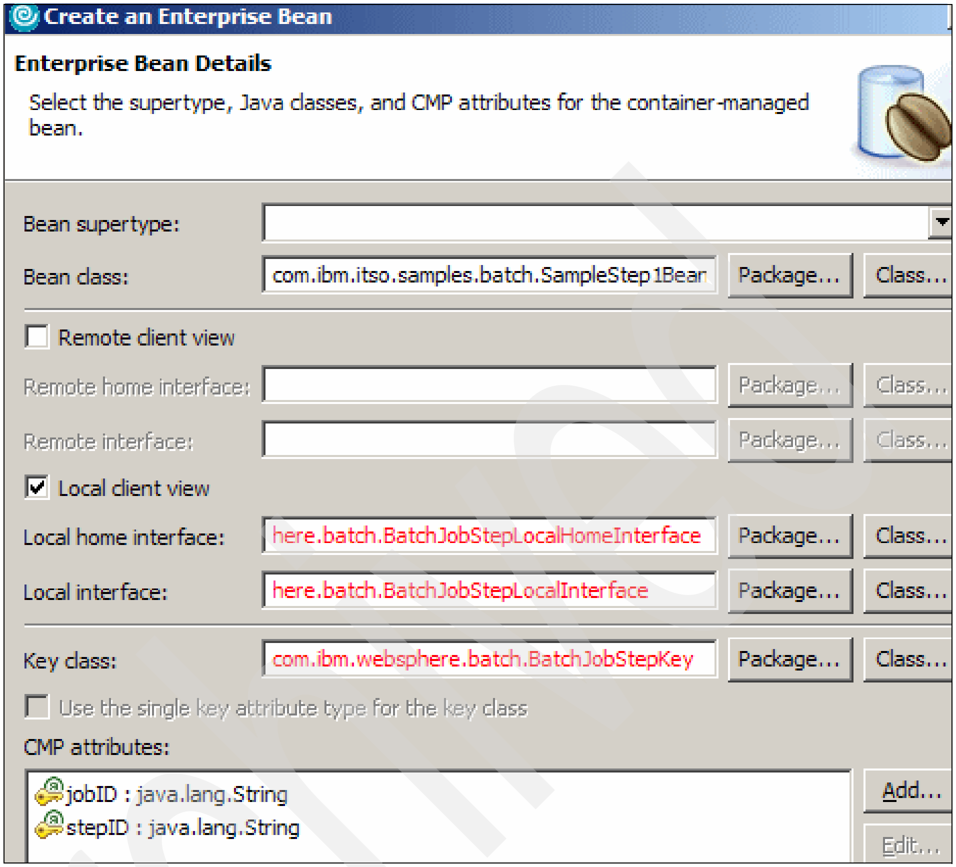


Figure 7-12 Specifying the interfaces and configuring CMP attributes for the entity bean

6. Click **Next**.
7. Enter **com.ibm.itso.samples.batch.SampleStep1** for the super class of the entity bean.
8. Click **Finish** to create the bean.

7.2.5 Create the CMP mapping and database table

The SampleStep1 CMP entity bean requires access to a database table in order to be able to persist its fields. The following steps are required to do this:

- Create the EJB to RDB mapping.
- Use the DDL to create schema and tables.

Since we intend to run the application in our Rational Application Developer Test Environment, we can use the LREE database, DEVLREE (see section 6.5.3, “Creating and configuring the LRS and LREE databases” on page 198) for this purpose. This database is also used by the LREE application so we will use two different schemas to separate the application data as shown in Table 7-3.

Table 7-3 Schemas in the database “DEVLREE”.

Application	Data source	Schema
LREE	`jdbc/lree`	`LREESCHEMA`
BatchSample	`jdbc/lree`	`SAMPLE`

Note that we still need to configure a JNDI name space binding to jdbc/lree in the EJB deployment descriptor. We will do this in 7.2.6, “Configure EJB deployment descriptor” on page 238.

> **Note:** The table for the CMP entity beans can be stored in a separate database if desired. In this case we chose to use an existing database to reduce complexity. We do not need to create an additional database and data source.

Create the EJB to RDB mapping

We need to define how the EJB container persists the CMP fields of our entity bean to the relational database (RDB). This step generates the Data Definition Language (DDL) that can create the required database schema and tables.

1. Right-click the EJB project and select **EJB to RDB Mapping → Generate Map**.
2. Select **Create a new backend folder**, and click **Next**.
3. Choose **Top-Down** to generate the database schema and map from the existing CMP entity beans (in this case there is only one, SampleStep1).
4. Specify the following top-down mapping options, as shown in Figure 7-13 on page 237:
   - Select **DB2 Universal Database V8.2** as the target database.
   - Specify `DEVLREE` as the database name.
   - Choose a schema name, for example `SAMPLE`.
   - Make sure **Generate DDL** is selected.

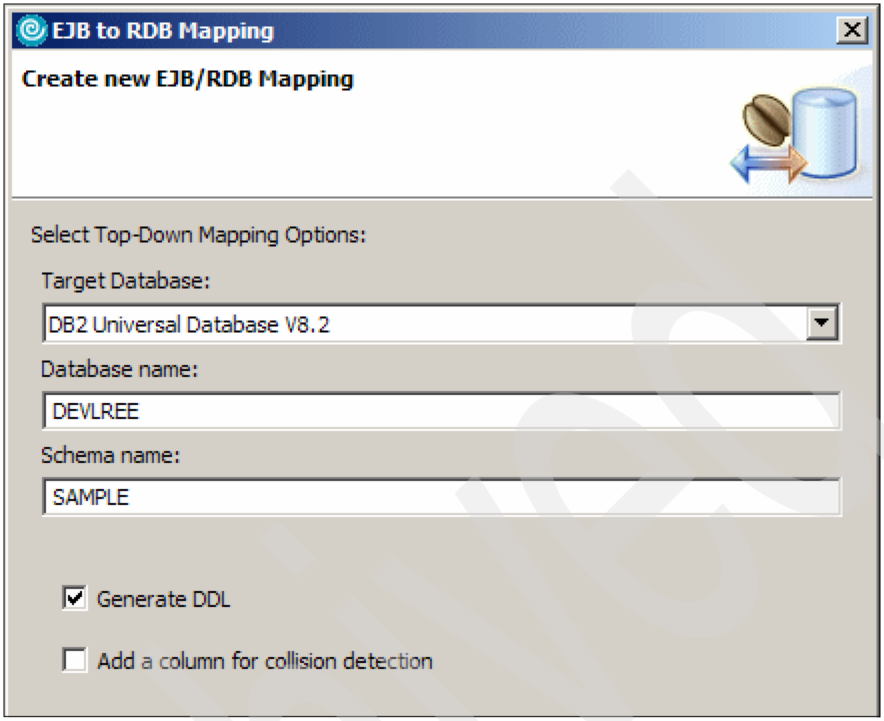


Figure 7-13 Creating the EJB to RDB mapping for the SampleStep1 CMP entity bean

5. Click **Finish** to create the mapping and to generate the DDL.

Use the DDL to create schema and tables

In the previous step, we generated the DDL to create the database schema and tables. Before we can use the DDL we need to export it from Rational Application Developer:

1. Make sure you are in the J2EE perspective, and navigate to your EJB project (BatchSampleEJBs).
2. Expand **ejbModule** and drill down to **META-INF** → **backends** → **DB2UDBNT_V82_1**.
3. Right-click the **Table.ddl** file and choose **Export**.
4. Select **File System**, and follow the steps in the wizard to export the file.
5. Copy the DDL file to your database system and use it to create the database schema and tables.

Note: For DB2, open a DB2 command line window, and use the following commands to connect to your database and to create the schema and tables:

```
db2 connect to DEVLREE user <username> using <password>
db2 -tvf Table.ddl
```

7.2.6 Configure EJB deployment descriptor

We need to configure the “BatchSampleEJBs EJB module to make sure it runs correctly in a long-running execution environment. Because there are quite a few steps, here is an overview of the different things we need configure:

- Configure the JNDI name for the Batch Job Controller bean.
- Configure the JNDI name for the Batch Job Step bean.
- Configure the EJB resource references.
- Configure the WorkManager resource reference.
- Configure the JNDI name for the JDBC data source.
- Configure the bean cache for the Batch Job Step beans.
- Configure transactions for the Batch Job Step beans.

Configure the JNDI name for the Batch Job Controller bean

Rational Application Developer V6.0 generates a default JNDI name for each EJB based on the name of the home interface. The Batch Job Controller bean home interface implementation is provided by Extended Deployment; therefore, the following default JNDI name space binding is used for each batch application:

```
ejb/com/ibm/ws/batch/BatchJobControllerHome
```

When deploying multiple batch applications, this results in JNDI name space conflicts. Hence, it is a best practice to change the name space binding to something that is unique, for example the name of the Batch Job Controller bean.

1. In the Beans tab of the deployment descriptor, select the BatchSampleController Batch Job Controller bean.
2. In the right panel, change the JNDI name under WebSphere Bindings to ejb/com/ibm/itso/samples/batch/BatchSampleController, as shown in Figure 7-14 on page 239.

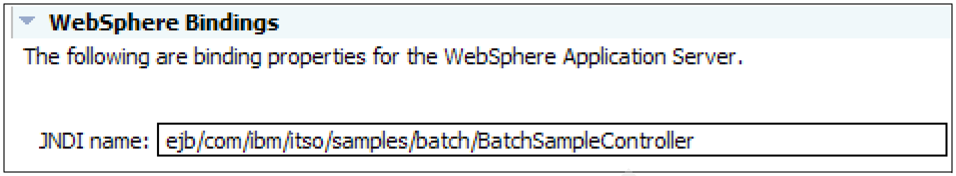


Figure 7-14 Changing the default JNDI name space binding of the Batch Job Controller bean

Configure the JNDI name for the Batch Job Step bean

A similar story applies to the default JNDI name space binding for the Batch Job Step bean, which is always the following:

```
ejb/com/ibm/websphere/batch/BatchJobStepLocalHomeInterface
```

Since multiple Batch Job Step beans can be defined—even in one batch application—it is best practice to always configure each Batch Job Step bean with its own, unique JNDI name space binding.

1. On the Beans tab of the deployment descriptor, select the SampleStep1 Batch Job Step bean.
2. In the right panel, change the JNDI name under WebSphere Bindings to ejb/com/ibm/websphere/batch/SampleStep1, as shown in Figure 7-15.

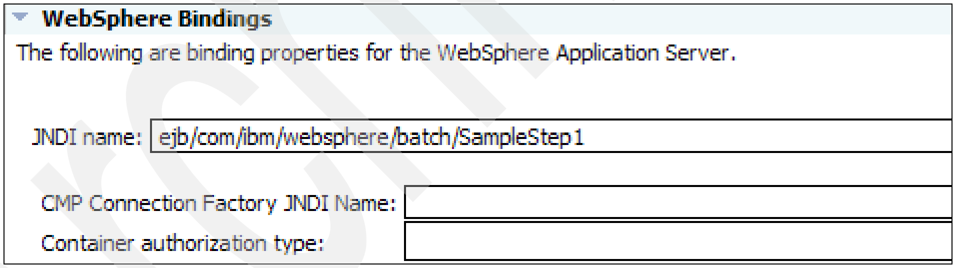


Figure 7-15 Changing the default JNDI name space binding of the Batch Job Step bean

Configure the EJB resource references

The LREE calls the Batch Job Step bean under the context of the Batch Job Controller bean. We need to make sure that the resource references used for the Batch Job Step bean(s) for the batch job are defined on the Batch Job Controller bean. Those references are used in the xJCL for the job. See section 7.7.1, “Main elements in xJCL file for batch jobs” on page 280.

> **Note:** It is important to point out that any resource references that are used in the batch application actually have to be declared on the Batch Job Controller bean. For example a resource reference for a JDBC data source.

1. Open the deployment descriptor for BatchSampleEJBs, and go to the References tab.
2. Select the BatchSampleController Batch Job Controller bean.
3. Click **Add** to add a reference for the SampleStep1 Batch Job Step bean.

   > **Note:** When building a batch application that consists of multiple steps, we need to add a reference for each Batch Job Step CMP entity bean here.

4. Choose **EJB reference**, and click **Next**.
5. Make sure that **Enterprise Beans in the workspace** are selected, and expand the BatchSample enterprise application.
6. Expand the BatchSampleEJBs EJB module and select the SampleStep1 CMP entity bean.
7. Enter a reference name, for example `ejb/SampleStep1`.

   > **Note:** When configuring xJCL for this batch application, we will use this reference to point to the different job steps.

8. Ensure that **Local** is selected as Ref Type, as seen in Figure 7-16 on page 241. Click **Finish**.

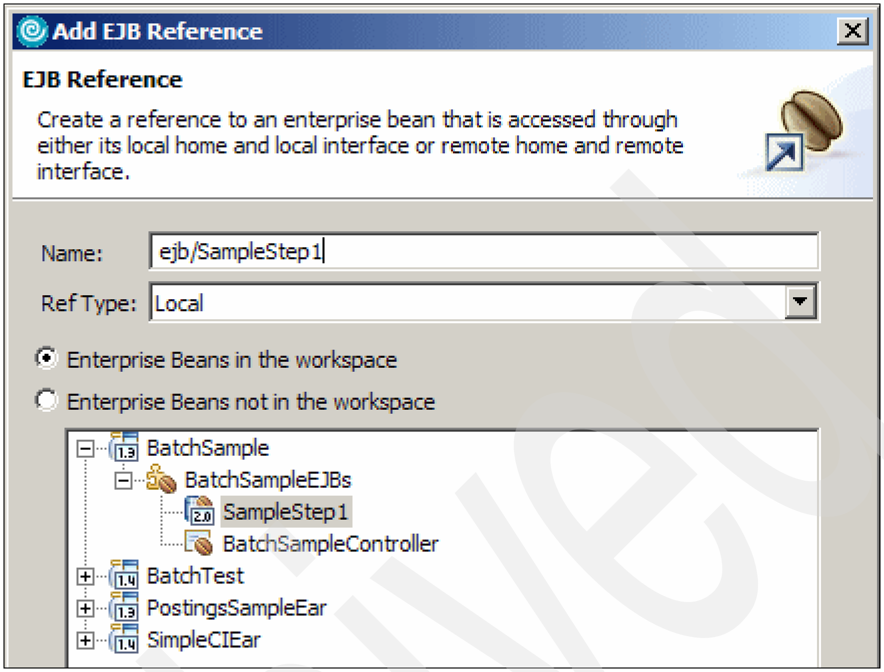


Figure 7-16 Creating an EJB reference for the Batch Job Step bean

Configure the WorkManager resource reference

The Batch Job Controller bean uses the *wm/BatchWorkManager* resource reference to look up a WorkManager. It uses this WorkManager to asynchronously run the job steps (the job steps are running in a separate thread). Therefore, we need to configure a resource reference on the Batch Job Controller bean that points to the default WorkManager on the LREE application servers.

1. Open the deployment descriptor for BatchSampleEJBs, and go to the References tab.
2. Select the BatchSampleController Batch Job Controller bean.
3. Click **Add** to add a reference.
4. Choose **Resource reference**, and click **Next**.
5. Enter the following, as shown in Figure 7-17 on page 242:
   a. Enter `wm/BatchWorkManager` as the name of the resource reference.
   b. For the type, select **commonj.work.WorkManager**.
   c. For Authentication, choose **Container**.
   d. Select **Shareable** as sharing scope.

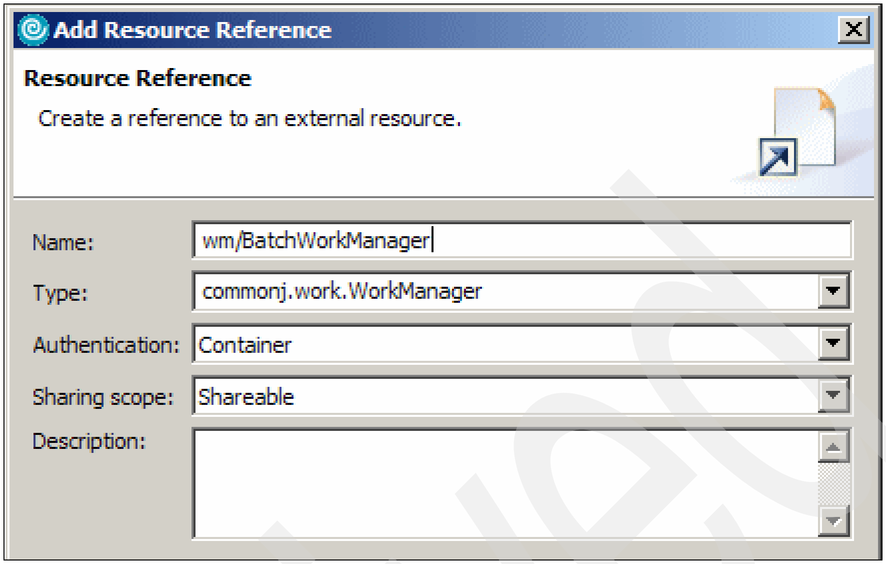


Figure 7-17 Creating a resource reference for the WorkManager

6. Click **Finish** to create the resource reference.
7. Provide a JNDI name space binding for this resource reference:
   a. Make sure that the new wm/BatchWorkManager resource reference is selected.
   b. Change the JNDI name under WebSphere Bindings to `wm/default`.

Configure the JNDI name for the JDBC data source

As discussed when creating the CMP mapping for the Batch Job Step bean, we need to configure it to use the LREE database, DEVLREE, for persistence. We specify jdbc/lree as the JNDI name, pointing to the data source that is already configured to access this database.

1. Open the deployment descriptor for the BatchSampleEJBs EJB module, and go to the Overview tab.
2. Scroll down to JNDI - CMP Connection Factory Mapping, and enter the following options, as shown in Figure 7-18 on page 243.
   a. Enter **`jdbc/lree`** for the JNDI name.
   b. Select **Use Default Method** under JAAS Logon Configuration.
   c. Specify the J2C Authentication Alias name for the LREE data source.

Note: This name can be found in the administrative console, under **Global Security → JAAS Configuration → J2C Authentication data**.

Note: This data source can be configured on each individual Batch Job Step bean as well, as shown in Figure 7-15.

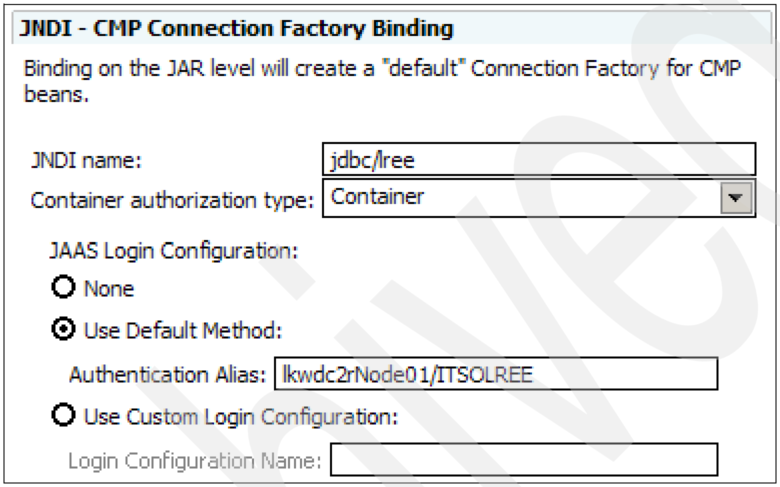


Figure 7-18 Configure the JDBC name space binding for the CMP entity bean

Configure the bean cache for the Batch Job Step beans

The Extended Deployment batch programming model requires that the Batch Job Step beans persist only two fields into the database (jobID and stepID). The actual bean might hold more state information, so we need to make sure that the bean instance is not passivated by the EJB container. In order to ensure this, we specify the EJB caching *option A* to be used for the Batch Job Step bean.

Note: More information about EJB caching in general can be found in the IBM Redbook *WebSphere Application Server V6.1: System Management and Configuration*, SG24-7304.

1. On the Beans tab of the deployment descriptor, select the SampleStep1 Batch Job Step bean.
2. In the right panel, scroll down to Bean Cache and select the following option as shown in Figure 7-19 on page 244:
   – For the Activate at: field, select **ONCE**.

– For the Load at: field, select **ACTIVATION**.

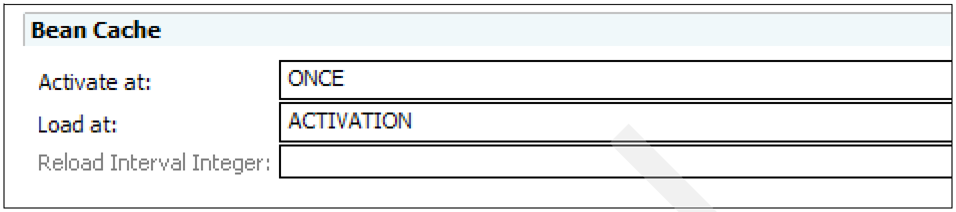


Figure 7-19 Configuring the bean cache for the Batch Job Step bean

Configure transactions for the Batch Job Step beans

The LREE always calls methods on the Batch Job Step beans under the scope of a global transaction. This is a requirement for the batch programming model; therefore, we will configure all Batch Job Step bean methods to require a transaction.

1. On the Assembly tab of the deployment descriptor, select the SampleStep1 Batch Job Step bean.
2. Under Container Transactions, select **Add**.
3. Select the SampleStep1 bean, and click **Next**.
4. Select **Required** as the container transaction type.
5. Click **Apply to All** to select all methods on the bean.
6. Click **Finish**.

Under Container Transactions you should now see that all methods on the SampleStep1 bean require a transaction, as shown in Figure 7-20.

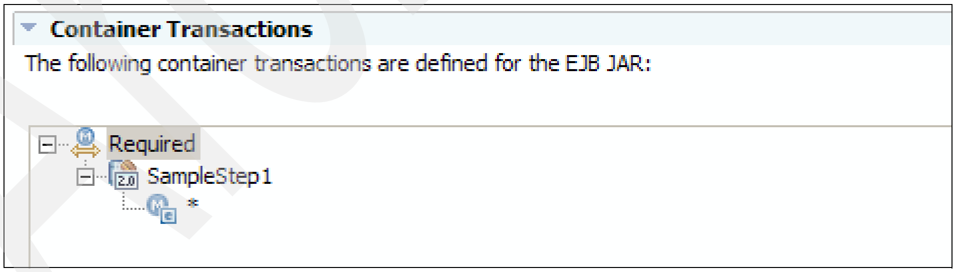


Figure 7-20 All methods on bean require a transaction

> **Note:** Remember to save the changes to the deployment descriptor!

7.2.7 Implement sample business logic

We now need to implement some business logic in the class com.itso.sample.batch.SampleStep1. Remember, this is the class that is being used as superclass by our SampleStep1 Batch Job Step bean. So it will inherit all the business logic we implement in the aforementioned class.

For a “Hello World!” style batch application, we chose to implement the code shown in Example 7-1. Basically we only log a few lines to SystemOut.log and the processJobStep() method returns the constant BatchConstants.STEP_COMPLETE the first time it is called.

> **Note:** The processJobStep() method of a typical batch application will not immediately return BatchConstants.STEP_COMPLETE. Instead, it returns BatchConstants.STEP_CONTINUE to tell the long-running execution environment that the batch loop should continue, and calls the processJobStep() method again.
>
> BatchConstants.STEP_COMPLETE should only be returned when the business logic implemented in processJobStep() determines that the batch loop has completed.
>
> This is discussed in more detail in section 7.6.1, “Batch Job Step bean” on page 269.

Example 7-1 “Hello World” style batch application business logic

```
package com.ibm.itso.sample.batch;

import java.util.Properties;
import com.ibm.websphere.batch.BatchJobStepInterface;
import com.ibm.websphere.batch.BatchConstants;

public class SampleStep1 implements BatchJobStepInterface {

   public void setProperties(Properties arg0) {
   }

   public Properties getProperties() {
   return null;
   }

   public void createJobStep() {
      System.out.println("SampleStep1: createJobStep");
   }
```

```
   public int processJobStep() {
      System.out.println("SampleStep1: processJobStep");
      return BatchConstants.STEP_COMPLETE;
   }

   public int destroyJobStep() {
      System.out.println("SampleStep1: destroyJobStep");
      return 0;
   }
}
```

Obviously, this business logic is not realistic for a batch application. However, the good news is that you can now start implementing business logic for *your* batch application in the com.ibm.itso.samples.batch.SampleStep1 implementation class. Subsequent steps might include any of the following:

- Implement additional Batch Job Step beans.
- Implement one or more batch data streams.
- Implement your own checkpoint algorithm.
- Implement your own results algorithm.

Please refer to section 7.6, "Batch application programming model" on page 269 for more details on the batch programming model and the interfaces you need to implement. Also, if you want to use a relational database for your batch data stream implementations, be sure to read section 7.4, "Using JDBC in a batch data stream" on page 251.

Last, but not least, we want to show some xJCL that you could use to submit a job to the sample batch application. The xJCL is shown in Example 7-2 on page 247.

> **Note:** We configured several JNDI names and resource references on the EJB deployment descriptor in section 7.2.6, "Configure EJB deployment descriptor" on page 238. Now is the time to recall what we configured on the BatchSampleController Batch Job Controller bean:
>
> - The JNDI name specified for the job in the xJCL matches the name space binding configured under WebSphere Bindings.
> - The ejb/SampleStep1 JNDI reference for the jobstep in the xJCL matches the resource reference to the SampleStep1 Batch Job Step bean.

Example 7-2 Sample xJCL to submit a job to the sample batch application

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <jndi-name>ejb/com/ibm/itso/samples/batch/BatchSampleController</jndi-name>
   <step-scheduling-criteria>
      <scheduling-mode>sequential</scheduling-mode>
   </step-scheduling-criteria>
   <checkpoint-algorithm name="timebased">
      <classname>com.ibm.wsspi.batch.checkpointalgorithms.timebased</classname>
      <props>
         <prop name="interval" value="15" />
      </props>
   </checkpoint-algorithm>
   <results-algorithms>
      <results-algorithm name="jobsum">
         <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
      </results-algorithm>
   </results-algorithms>
   <job-step name="Step1">
      <jndi-name>ejb/SampleStep1</jndi-name>
      <checkpoint-algorithm-ref name="timebased" />
      <results-ref name="jobsum"/>
      <batch-data-streams>
      </batch-data-streams>
   </job-step>
</job>
```

7.3 Using a batch data stream in a batch application

Batch applications should use a batch data stream to read or write records. In other words, the batch data stream provides access to the data from the business logic implemented in the Batch Job Step bean. A typical batch application reads records one by one, does something with each record, and then writes out the processed records. In that case, we need two batch data streams. One to read records and another one to write them.

7.3.1 Implementing a batch data stream

A batch data stream is a Java class that implements the com.ibm.websphere.batch.BatchDataStream interface. These objects provide an abstraction of the data, and more importantly, can be positioned by the LREE.

For more details on how to implement a batch data stream, please refer to section 7.6.2, “Batch data stream interface” on page 273.

7.3.2 Using a batch data stream

After implementing a batch data stream, it is important to understand how to use it in a batch application. In order to be able to use a batch data stream, it needs to be defined in the batch job xJCL. A batch data stream is not associated with a batch job, but with a single job step. Example 7-3 shows how to associate an input and an output batch data stream with job step “Step1”.

For more details on the actual xJCL syntax, please refer to section 7.7.5, “Batch data streams” on page 285.

Example 7-3 Configuring batch data streams for a job step in xJCL.

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
   <job-step name="Step1">
...
      <batch-data-streams>
         <bds>
            <logical-name>myinput</logical-name>
            <impl-class>com.ibm.itso.batch.BatchInputStreamImpl</impl-class>
            <props>
               <prop name="FILENAME" value="c:\temp\input" />
            </props>
         </bds>
         <bds>
            <logical-name>myoutput</logical-name>
            <impl-class>com.ibm.itso.batch.BatchOutputStreamImpl</impl-class>
            <props>
               <prop name="FILENAME" value="c:\temp\output" />
            </props>
         </bds>
      </batch-data-streams
   </job-step>
...
</job>
```

After associating the batch data stream with the job step through xJCL, the Batch Job Step bean associated with this job step needs to obtain a handle to the batch data stream. We recommend that you obtain this handle in the createJobStep() of the Batch Job Step bean, as shown in Example 7-4.

> **Note:** It is *very important* to point out that the handle to the batch data stream can only be obtained from within the Batch Job Step bean. This is because the JobStepID can only be obtained using the jobID and stepID of the Batch Job Step bean. The JobStepID is required when calling the BatchDataStreamMgr.

Example 7-4 Obtaining a batch data stream in createJobStep() method of the Batch Job Step bean

```
package com.ibm.itso.sample.batch.ejb;

import com.ibm.websphere.batch.BatchDataStreamMgr;
import com.ibm.websphere.batch.BatchContainerDataStreamException;
import com.ibm.websphere.batch.JobStepID;
import com.ibm.itso.sample.batch.BatchInputStreamImpl;
import com.ibm.itso.sample.batch.BatchOutputStreamImpl;

public abstract class SampleStep1Bean implements javax.ejb.EntityBean {
   private JobStepID id;
   private String inputBDSLogicalName = "inputBDS";
   private String outputBDSLogicalName = "outputBDS";
   private BatchInputStreamImpl inputBDS;
   private BatchOutputStreamImpl outputBDS;

   public void createJobStep() {
      id = new JobStepID(this.getJobID(),this.getStepID());

      try{
         inputBDS = (BatchInputStreamImpl)
BatchDataStreamMgr.getBatchDataStream(inputBDSLogicalName, id.getJobstepid());
         outputBDS = (BatchOutputStreamImpl)
BatchDataStreamMgr.getBatchDataStream(outputBDSLogicalName, id.getJobstepid());
      } catch(BatchContainerDataStreamException bcdse){
         System.out.println(bcdse);
      }
   }
```

7.3.3 Using a batch data stream from a separate implementation class

A handle to a batch data stream can only be obtained from within the Batch Job Step bean itself; however, this is a problem when implementing the actual business logic in a separate class. In our sample batch application, the actual Batch Job Step bean extends the class SampleStep1, which implements the com.ibm.websphere.batch.BatchJobStepInterface.

In order to make this work, the method used to obtain the handle to the batch data stream in the SampleStep1 class needs to be overridden in the Batch Job Step bean. This is demonstrated in Example 7-5, where the method createJobStep() is overridden.

Example 7-5 Obtaining a batch data stream in createJobStep() method of the Batch Job Step bean

```
package com.ibm.itso.sample.batch.ejb;

import com.ibm.websphere.batch.BatchDataStreamMgr;
import com.ibm.websphere.batch.BatchContainerDataStreamException;
import com.ibm.websphere.batch.JobStepID;
import com.ibm.itso.sample.batch.BatchInputStreamImpl;
import com.ibm.itso.sample.batch.BatchOutputStreamImpl;

public abstract class SampleStep1Bean
   extends
      com.ibm.itso.sample.batch.SampleStep1 implements javax.ejb.EntityBean {
   private JobStepID id;

   public void createJobStep() {
      id = new JobStepID(this.getJobID(),this.getStepID());

      try{
         inputBDS = (BatchInputStreamImpl)
BatchDataStreamMgr.getBatchDataStream(inputBDSLogicalName, id.getJobstepid());
         outputBDS = (BatchOutputStreamImpl)
BatchDataStreamMgr.getBatchDataStream(outputBDSLogicalName, id.getJobstepid());
      } catch(BatchContainerDataStreamException bcdse){
         System.out.println(bcdse);
      }
   }
```

In order to use the batch data stream in the SampleStep1 implementation class, we define a BatchDataStream object, inputBDS, in SampleStep1. This is shown in Example 7-6 on page 251. Note that the createJobStep() method is empty since we decided to override it. The createJobStep() method in the Batch Job Step bean uses inputBDS to store the handle to the actual batch data stream.

Example 7-6 Using a batch data stream in the implementation class of the Batch Job Step bean

```
package com.ibm.itso.sample.batch;

import java.util.Properties;
import com.ibm.websphere.batch.BatchJobStepInterface;
import com.ibm.websphere.batch.BatchConstants;
import com.ibm.websphere.batch.BatchDataStream;
import com.ibm.itso.sample.batch.BatchInputStreamImpl;
import com.ibm.itso.sample.batch.BatchOutputStreamImpl;

public class SampleStep1 implements BatchJobStepInterface {
   private String inputBDSLogicalName = "inputBDS";
   private String outputBDSLogicalName = "outputBDS";
   private BatchInputStreamImpl inputBDS;
   private BatchOutputStreamImpl outputBDS;

public void createJobStep() {
}

public int processJobStep() {
...
inputBDS.getNextRecord();
...
outputBDS.putNextRecord();
...
return BatchConstants.STEP_CONTINUE;
}
```

7.4 Using JDBC in a batch data stream

A typical implementation of a batch data stream might retrieve data from—or insert data into—a relational database. This section describes a sample implementation of such a batch data stream. A number of best practices when using JDBC in a batch data stream are highlighted as well.

7.4.1 Overview

Let us assume that our sample batch data stream reads records from a large database table. In other words, we use it as input for our business logic implemented in the processJobStep() method of our Batch Job Step bean.

In our sample batch data stream, we want to implement a getNextRecord() method that returns the data from the next row in the database table. Instead of returning a primitive data type, we want this method to return a simple Java bean (InputDataRecord). Of course we also need to implement all the methods defined in the com.ibm.websphere.batch.BatchDataStream interface as discussed in section 7.6.2, “Batch data stream interface” on page 273.

> **Note:** From a technical point-of-view, we could also return the JDBC ResultSet. However a batch data stream should be treated as a data abstraction layer, so we chose to return a simple Java bean containing the data.

7.4.2 JDBC resource reference

We configure a resource reference for the JDBC data source in order to ease the lookup from within the batch data stream implementation. All methods on the batch data stream object are called under the context of the Batch Job Controller bean; therefore, we configure a resource reference for the data source on the deployment descriptor of this bean.

1. Open the deployment descriptor for the EJB module containing the Batch Job Controller bean definition, and go to the References tab.
2. Select the controller bean.
3. Click **Add** to add a reference.
4. Choose **Resource reference**, and click **Next**.
5. Enter the following, as shown in Figure 7-21 on page 253:
   a. Enter a name for the resource reference, for example `jdbc/bds`.
   b. For the type, select **javax.sql.DataSource**.
   c. For Authentication, choose **Application**.
   d. Select **Unshareable** as the sharing scope.

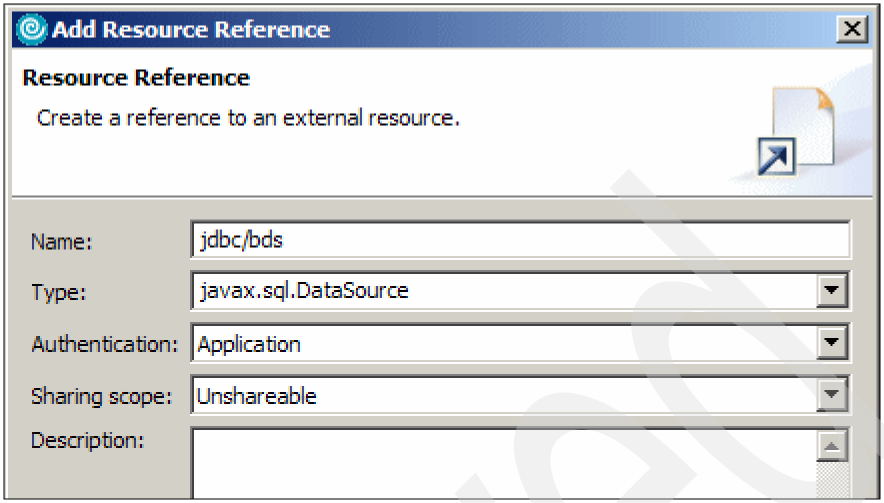


Figure 7-21 Creating a resource reference for the batch data stream data source

6. Click **Finish** to create the resource reference.
7. Now provide a JNDI name space binding for this resource reference to the data source you intend to use.
   a. Make sure that the new resource reference, jdbc/bds, is selected.
   b. Change the JNDI name under WebSphere Bindings to the JNDI name of the data source you want to use.

7.4.3 Implementation

Note: The full implementation of this sample is shown in section 7.4.4, "Source code of sample batch data stream implementation" on page 257.

It makes sense to open the JDBC connection in the open() method of the BatchDataStream implementation, and to close it again in the close() method. We can perform a SQL query in the open method and use the ResultSet in the getNextRecord() method to step through each of the records. However, the problem is that we frequently commit global transactions in the batch loop. This closes the ResultSet and breaks our batch data stream.

The most obvious solution is to obtain the ResultSet again in the intermediateCheckpoint() method. This method is called by the LREE every time, just after committing a global transaction in the batch loop.

However this means that we need to execute the query on each commit, hence the number of SQL statements that the database needs to execute will be higher than necessary.

Another solution is to perform a SQL query on each call to processJobStep(), and obtain the correct record (**x** in this case) from the ResultSet:

```
someConnection = someDataSource.getConnection();
someStatement = someConncetion.createStatement();
someResultSet = someStatement.execute("SELECT * FROM ...");
someResultSet.getRow(x);
```

While this is a perfectly viable solution, it is far from optimal. The number of SQL statements that the database needs to execute will be tremendously higher compared to when we can re-use the ResultSet on each processJobStep() method call.

Under certain circumstances, *cursor holdability* can provide an alternative approach where the ResultSet can be re-used. This will minimize the number of SQL statements for the database and is likely to enhance performance.

Solution using cursor holdability

The JDBC 3.0 API provides the option to hold the cursor in a ResultSet across global transaction boundaries so that after a commit the ResultSet is not closed and the cursor remains at the same record. This option can be set on the java.sql.Connection object. The default depends on the database vendor's JDBC provider implementation:

```
Connection.setHoldability(ResultSet.HOLD_CURSORS_OVER_COMMIT)
```

> **Note:** When using the DB2 Universal Database Type IV JDBC Provider, the cursor holdability can also be set through a *custom property* on the data source. The name of this property is **`resultSetHoldability`** and the possible values are the following:
>
> - **`HOLD_CURSORS_OVER_COMMIT`**
> - **`CLOSE_CURSORS_OVER_COMMIT`** (default)

You cannot specify cursor holdability on a *shareable* connection when using global transactions. Upon commit, all statements and ResultSets are closed. Hence when enabling cursor holdability for the reasons described earlier, you should make sure that the sharing scope is set to *unshareable*. This can be configured on the EJB deployment descriptor (Figure 7-22 on page 255), since the JDBC resource references are configured there as well.

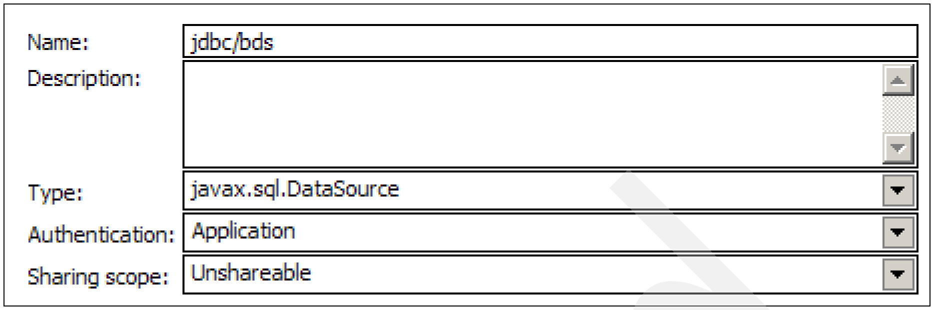


Figure 7-22 Setting the sharing scope on the data source resource reference

> **Note:** DB2 Universal Database V8.2 currently only supports cursor holdability on non-XA resources. In other words, when using the DB2 Type IV JDBC Provider, cursor holdability is only supported when using com.ibm.db2.jcc.DB2ConnectionPoolDataSource as the implementation class.

For more information, visit the following Web sites:

- Cursor holdability in the WebSphere Application Server Information Center:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/rdat_cursorhold.html`

- Last participant support in WebSphere Application Server Information Center:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/lao/concepts/cla_ovrvw.html`

Solution with cursor holdability using DB2

As mentioned above, the cursor holdability only works when using the DB2 JDBC provider in one-phase commit transactions. When running long-running batch applications in Extended Deployment, we have a requirement to run global transactions that involve the LREE data source to be able to commit checkpoint information, as discussed in section 7.1.2, “Interaction with the long-running execution environment” on page 222. Using another data source in a batch data stream basically requires two-phase commit capable data sources in order to run these global transactions.

Last participant support (LPS) in WebSphere Application Server V6.0 and above provides a solution to this problem. Applications in WebSphere can be configured to allow a single one-phase commit capable resource to participate in two-phase commit global transactions. There are two configuration steps required in order for this to work:

1. Configure the transaction service of all application servers involved to support LPS.
   a. Select a member in the LREE dynamic cluster.
   b. Go to Container Services, and select **Transaction Service**.
   c. Check **Enable logging for heuristic reporting**, as shown in Figure 7-23. Click **OK**.
   d. Repeat these steps for all application servers in the LREE dynamic cluster running these transactions.

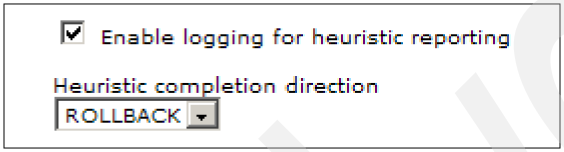


Figure 7-23 Configure the transaction service of the application server to support LPS

2. Configure all applications involved in these two-phase commit global transactions to support LPS.

   > **Note:** Typically this includes your batch application and the LREE application.

   LPS is enabled in the application deployment descriptor. You can do this using the Application Server Toolkit (Rational Application Developer V6.0 currently does not support this).

   To change the setting using the toolkit:

   a. Open the application deployment descriptor.
   b. Click the Extended Services tab.
   c. Check the last participant support box as shown in Figure 7-24 on page 257.

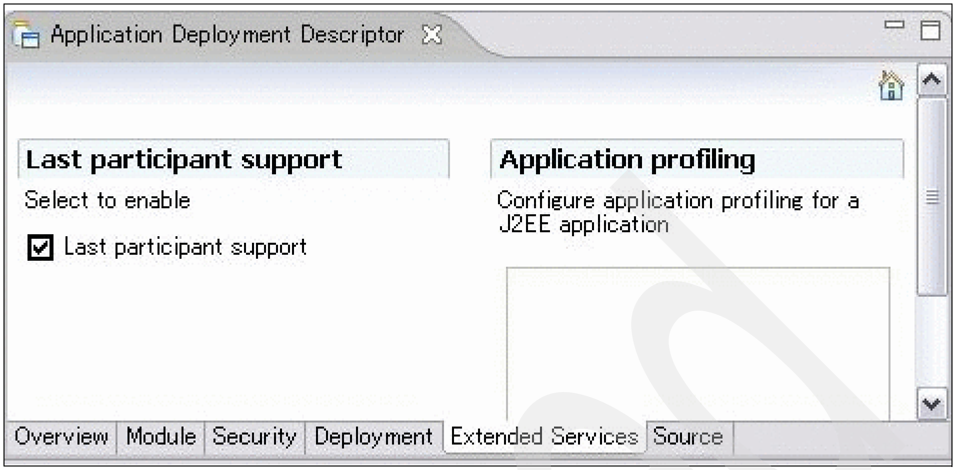


Figure 7-24 Enabling LPS in the application deployment descriptor

Repeat steps a through c for all applications.

> **Note:** This approach does not work if you are using multiple JDBC resources, since last participant support only works with one non-XA resource.

7.4.4 Source code of sample batch data stream implementation

Example 7-7 shows the source code of a sample batch data stream implementation that uses a JDBC data source. Without going into too much detail, it is important to point out that this batch data stream implementation uses the **record** field to keep track of the position in the batch data stream.

For more details on how to implement a batch data stream, or how to obtain a handle to a batch data stream in the Batch Job Step bean, please refer to section 7.3, "Using a batch data stream in a batch application" on page 247 and section 7.6.2, "Batch data stream interface" on page 273.

Example 7-7 Sample implementation of batch data stream

```
package com.ibm.itso.sample.batch;

import java.util.Properties;
import com.ibm.websphere.batch.BatchContainerDataStreamException;
import com.ibm.websphere.batch.BatchDataStream;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;
import java.sql.*;
```

```
public class InputBDS implements BatchDataStream {
   Properties props;// holds the batch data stream properties set in xJCL
   String bds_name;// holds the batch data stream logical name used in xJCL
   String job_step;// holds the Batch Job Step id
   Connection conn;// java.sql.Connection for the JDBC connection
   ResultSet resultSet;// java.sql.ResultSet for the JDBC connection
   DataSource ds;// javax.sql.DataSource for the JDBC connection
   String ds_name;// resource reference of the JDBC datasource, as set on the Batch
Job Controller Bean
   InputDataRecord idr;// com.ibm.itso.sample.batch.InputDataRecord to hold input
record data
   int record;// record number

   public String externalizeCheckpointInformation() {
      return new Integer(record).toString();
   }

   public void internalizeCheckpointInformation(String arg0) {
      record = Integer.parseInt(arg0);
   }

   public void initialize(String arg0, String arg1)
         throws BatchContainerDataStreamException {
      bds_name = arg0;
      job_step = arg1;
      ds_name = props.getProperty("ds_name");
   }

   public void positionAtInitialCheckpoint()
         throws BatchContainerDataStreamException {
      record = 1;
   }

   public void positionAtCurrentCheckpoint()
         throws BatchContainerDataStreamException {
      try {
         resultSet.absolute(record);
      }
      catch(SQLException se){
         log("SQLException occurred in InputBDS.open()");
         throw new BatchContainerDataStreamException(se);
      }
   }
```

```
   private void log(String arg0)
   {
      System.out.println("BDS name : " + bds_name + " Job Step ID : " + job_step + "
Message : " + arg0);
   }

   public void open() throws BatchContainerDataStreamException {
      try {
         // lookup the datasource using the resource reference of the JDBC
datasource, as set on the Batch Job Controller Bean
         ds = (DataSource) (new InitialContext().lookup("java:comp/env/" + ds_name));
         conn = ds.getConnection();
         conn.setHoldability(ResultSet.HOLD_CURSORS_OVER_COMMIT);
         Statement statement = conn.createStatement();
         String sql = "select * from INPUTDATA for read only with cs";
         if (statement.execute(sql)) {
            resultSet = statement.getResultSet();
         }
      }
      catch(SQLException se){
         log("SQLException occurred in InputBDS.open()");
         throw new BatchContainerDataStreamException(se);
      }
      catch(NamingException ne){
         log("NamingException occurred in InputBDS.open()");
         throw new BatchContainerDataStreamException(ne);
      }
   }

   public InputDataRecord getNextRecord() throws BatchContainerDataStreamException {
      record++;
      try {
         if (resultSet.next())
         {
            idr.setdata0(resultSet.getString(1));
            idr.setdata1(resultSet.getString(2));
            return idr;
         }
         else {
            return null;
         }
         } catch(SQLException se){
            log("SQLException occurred in InputBDS.getNextRecord()");
            throw new BatchContainerDataStreamException(se);
         }
```

```
    }

    public void close() throws BatchContainerDataStreamException {
        try {
            conn.close();
        }
        catch(SQLException se){
            log("SQLException occurred in InputBDS.close()");
            throw new BatchContainerDataStreamException(se);
        }
    }

    public String getName() {
        return bds_name;
    }

    public void setProperties(Properties arg0) {
        props = arg0;
    }

    public Properties getProperties() {
        return props;
    }

    public void intermediateCheckpoint() {
    }
}
```

7.4.5 Closing thoughts

The use of cursor holdability can be very effective when building batch applications that act on data from a database. The DB2 JDBC driver currently does not support two-phase commit transactions with cursor holdability, but in many cases you will find that one batch data stream for input and one for output are sufficient. The batch data stream used for output is unlikely to need to hold on to a ResultSet since it performs inserts in a database (if using a database at all).

7.5 Batch application flow

Before we cover the interfaces for the various J2EE and Java components in the Batch programming model, we will discuss how the components interact with the LREE.

After a job is dispatched to the LREE by the LongRunningScheduler, it looks up the Batch Job Controller bean associated with the job and calls the submitJob() method on this bean. The Batch Job Controller bean will use the wm/BatchWorkManager resource reference to look up the WorkManager. It then calls the schedule() method on an internal LREE class BatchJobControllerWork that implements commonj.work.Work interface. This internal LREE class performs a couple of other tasks to set up the job step and looks up or creates the Batch Job Step bean.

> **Note:** The use of the WorkManager is important. When the BatchJobControllerWork is scheduled, the WorkManager calls the run() method of this class asynchronously in a separate thread.

As discussed in section 7.1, "Overview" on page 220, a batch application can consist of multiple job steps, each of them with its own Batch Job Step bean as specified in the xJCL file used to submit the job. The execution of one of those job steps can be split up in three different phases, and we will discuss those in detail in the next three sections:

- **Prepare for execution phase**

  Preparing for execution of the business logic implemented in the Batch Job Step bean.

- **Executing the batch loop phase**

  Execution of the business logic in the batch loop.

- **Close after Execution phase**

  Closing resources after the batch loop finishes.

7.5.1 Prepare for execution phase

In the first execution phase of a batch step, the LREE creates a new instance for each of the batch data streams defined in the xJCL for this step. Then the LREE performs the following steps, as shown in the sequence diagram in Figure 7-25 on page 263:

1. Calls the initialize() method of the checkpoint algorithm associated with this step in the xJCL.

2. Calls setProperties() on each of the batch data streams to pass in the BatchDataStream properties specified in the xJCL.
3. Calls the initialize() method on each batch data stream.
4. Calls the open() method on each batch data stream.
5. Now the LREE positions each batch data stream at its initial checkpoint by calling positionAtInitialCheckPoint().

> **Note:** When a job is restarted, the LREE does not call positionAtInitialCheckPoint(). Instead, it retrieves the last checkpoint of the batch data stream from the LREE database (java.lang.String) and makes the following method calls:
>
> - internalizeCheckpointInformation(java.lang.String)
>
>   This provides the batch data stream with the checkpoint information.
>
> - positionAtCurrentCheckpoint()
>
>   This tells the batch data stream to position itself using the checkpoint information provided in the previous call.
>
> This forces each batch data stream to reposition itself to the position at the last commit during the previous run of the job step.

6. The Batch Job Step bean is created, and the EJB container calls the ejbCreate() method on the bean.

> **Note:** The LREE calls the findByPrimaryKey() method on the local home interface. If the jobID and stepID are not found in the database, the create() method is called to create the Batch Job Step bean.

7. Calls setProperties() on the Batch Job Step bean to set any properties configured in the xJCL.
8. Calls createJobStep() on the Batch Job Step bean. This method allows for any resources required for the business logic to be looked up or initialized. For example, this is where you should obtain a handle to the BatchDataStreams used by the business logic that is implemented in processJobStep().

Now everything is prepared in order to start executing the batch loop.

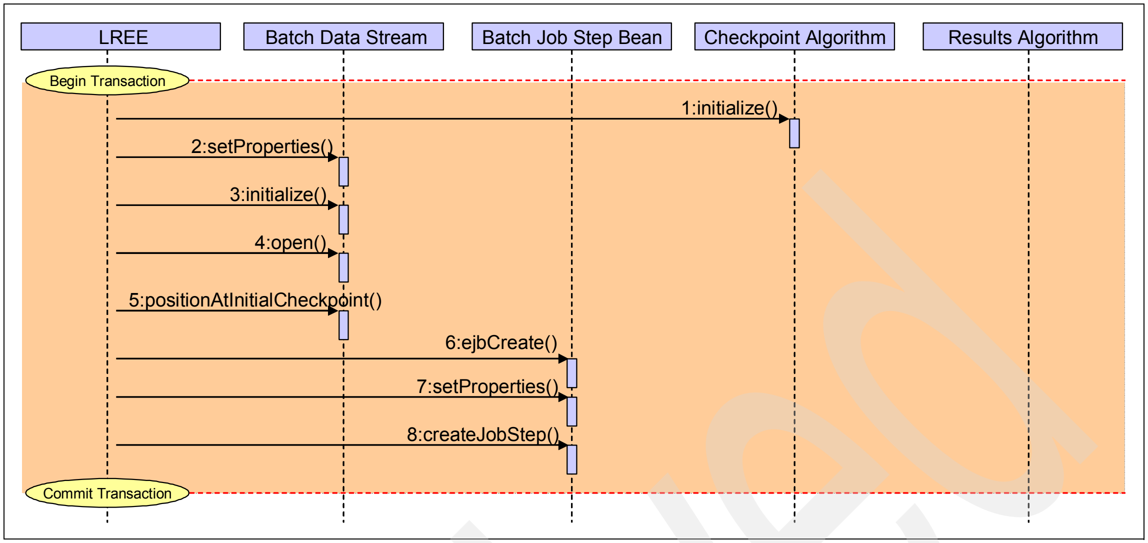


Figure 7-25 Batch job initialization sequence diagram

7.5.2 Executing the batch loop phase

The LREE has now finished preparing the batch job step for execution. Before it can start executing the processJobStep() method in the batch loop, the LREE will persist the initial checkpoint information to the LREE database (steps 2-4).

This batch job uses record-based checkpoint algorithms, with the checkpoint being committed after every second record. The data this BDS accesses has only three records.

1. Calls getRecommendedTimeOutValue() on the checkpoint algorithm. The LREE uses this value as timeout for the global transactions it starts from now on.
2. Calls startCheckpoint() on the checkpoint algorithm. This is to ensure that the algorithm knows that a global transaction is about to begin.
3. Calls externalizeCheckpointInformation() on each batch data stream involved in this job step. The LREE persists this information in the LREE database upon the global transaction commit.
4. Calls stopCheckpoint() on the checkpoint algorithm. This is to ensure that the algorithm knows that a global transaction was just committed.

Note: Even though the processJobStep() method was not called yet, the startCheckpoint() and stopCheckpoint() methods were called on the checkpoint algorithm. Keep this in mind if you are implementing your own checkpoint algorithm!

Now this is where the LREE really enters the batch loop and will keep running until processJobStep() returns BatchConstants.STEP_COMPLETE.

5. The LREE calls startCheckpoint() on the checkpoint algorithm. This is to ensure that the algorithm knows that a global transaction is about to begin.
6. The LREE calls intermediateCheckpoint() on each batch data stream involved in this job step.
7. The LREE calls processJobStep() on the Batch Job Step bean for the first time and the method returns BatchConstants.STEP_CONTINUE. This indicates to the LREE that it should continue running in the batch loop and should call processJobStep() again.
8. Before calling processJobStep() again, the LREE needs to verify whether a checkpoint needs to be executed (for example, should it commit the global transaction and start a new one). It calls shouldCheckpointBeExecuted() on the checkpoint algorithm. The algorithm returns false in this case, hence the LREE can call processJobStep() again.
9. The LREE calls processJobStep() on the Batch Job Step bean again, which again returns BatchConstants.STEP_CONTINUE.
10. The LREE calls shouldCheckpointBeExecuted() on the checkpoint algorithm. This time it returns true, indicating that the LREE should execute a checkpoint.

To execute a checkpoint, the LREE will now carry out these steps in order to execute the checkpoint:

11. The LREE calls externalizeCheckpointInformation() on each batch data stream. This method returns a java.lang.String, which is persisted in the LREE database by the LREE.
12. After committing the global transaction, the LREE calls the method stopCheckpoint() on the checkpoint algorithm.
13. The LREE is about to start another global transaction and calls startCheckpoint() on the checkpoint algorithm.
14. The LREE calls getRecommendedTimeOutValue() on the checkpoint algorithm and uses this value as a timeout on the global transaction that it is about to start.

After the LREE starts a new global transaction, the steps carried out are exactly the same as previously. In a typical batch application flow, many checkpoints are executed before a job step completes. However, in this example we assume that the step completes before it reaches the next checkpoint.

15. The LREE calls intermediateCheckpoint() on each batch data stream involved in this job step.
16. The LREE calls processJobStep() on the Batch Job Step bean again. This time it returns BatchConstants.STEP_COMPLETE. This indicates to the LREE that the batch job step was completed.
17. The LREE still calls the method shouldCheckpointBeExecuted() on the checkpoint algorithm to see whether a checkpoint should be executed right now. The algorithm returns false in this case; however, the LREE will execute a checkpoint anyway because the job step was completed.
18. The LREE calls externalizeCheckpointInformation() on each batch data stream. This method returns a java.lang.String, which is persisted in the LREE database by the LREE.
19. After committing the global transaction, the LREE will call stopCheckpoint() on the checkpoint algorithm.

The LREE has not finished executing the batch loop.

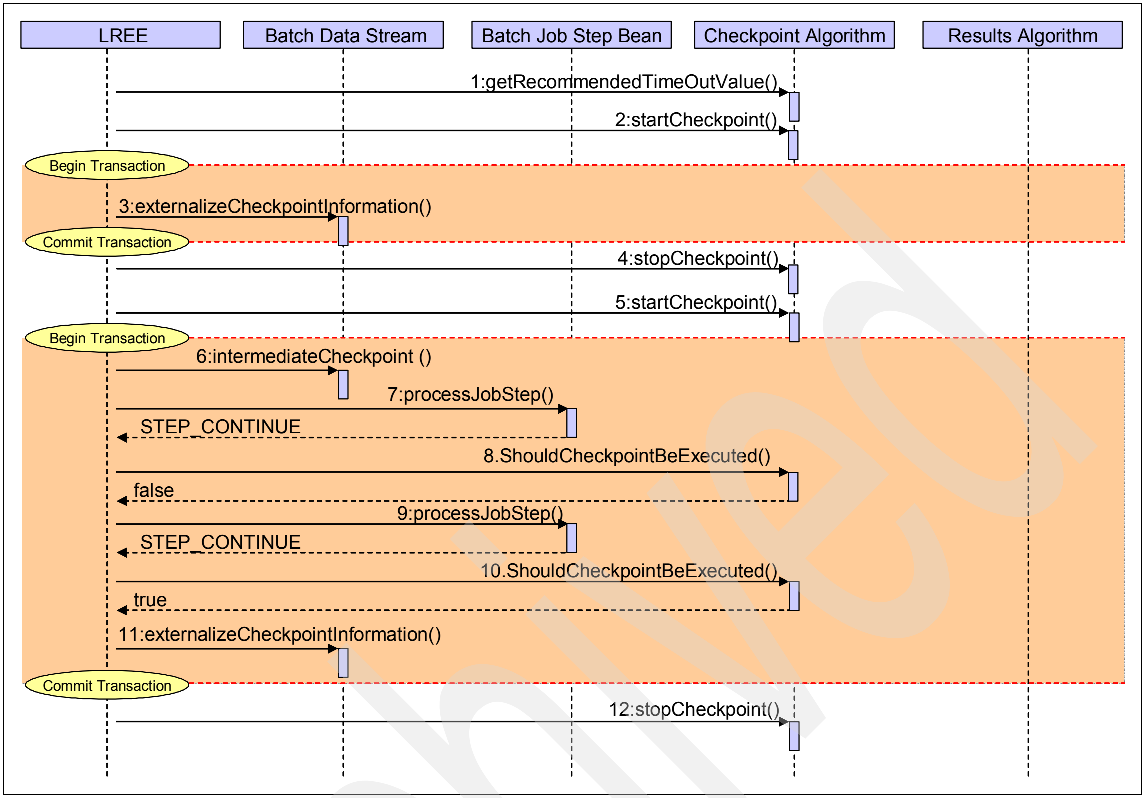


Figure 7-26 Batch job loop sequence diagram (continued)

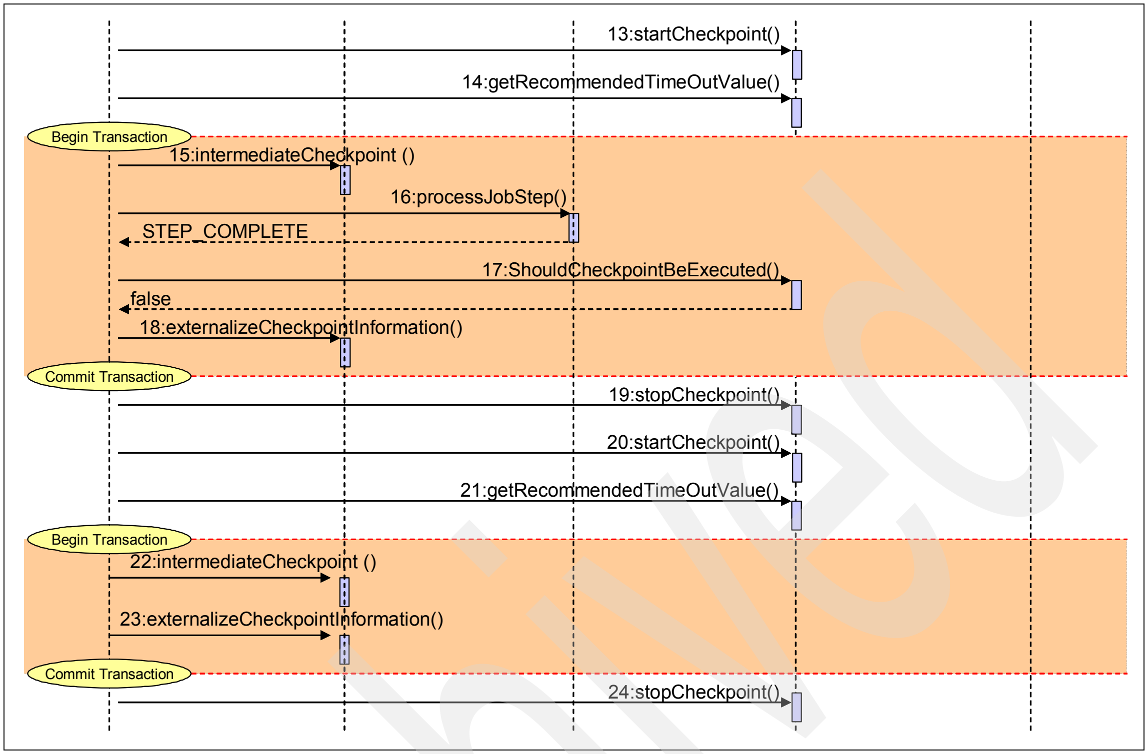


Figure 7-27 Batch job loop sequence diagram (continued)

7.5.3 Close after Execution phase

After the execution of the job step finishes, the LREE still needs to ensure that any resources used by the job step are released.

1. The LREE is about to start another global transaction and calls startCheckpoint() on the checkpoint algorithm.
2. The LREE calls getRecommendedTimeOutValue() on the checkpoint algorithm and uses this value as a timeout on the global transaction that it is about to start.
3. The LREE calls intermediateCheckpoint() on each batch data stream involved in this job step.
4. The LREE calls destroyJobStep() on the Batch Job Step bean. This tells the bean to release any resources it may have obtained in createJobStep().

Note: The return code of destroyJobStep() is passed to the results algorithm associated with the step.

5. The LREE calls close() on each batch data stream involved in this job step.
6. The LREE calls initialize() on every results algorithm associated with this job step in the xJCL.

Note: More than one results algorithm can be associated with a job step, more details can be found in the Extended Deployment infocenter at the following Web address:

`http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp?topic=/com.ibm.websphere.xd.doc/info/scheduler/cxdbatchres.html`

7. The LREE calls fireResultsAlgorithms() on each results algorithm and passes in the return code from destroyJobStep().
8. The LREE calls externalizeCheckpointInformation() on each batch data stream. This method returns a java.lang.String, which is persisted in the LREE database by the LREE.
9. After committing the global transaction, the LREE finally calls stopCheckpoint() on the checkpoint algorithm.

This completes the execution of a single job step. Any subsequent job steps are carried out in exactly the same way.

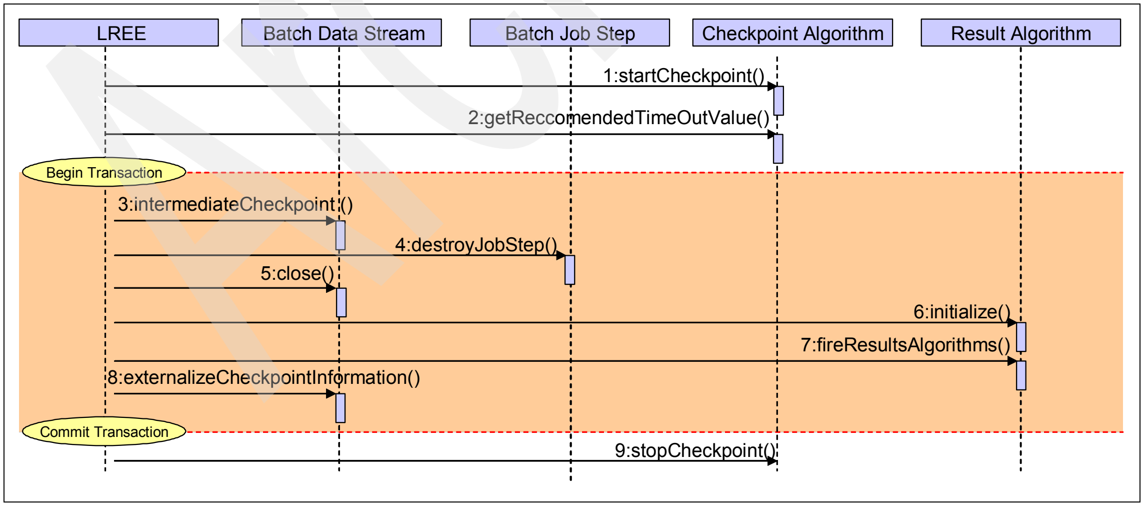


Figure 7-28 Batch job destroy sequence diagram

7.6 Batch application programming model

This section discusses the following elements of the application programming model for batch applications:

- Batch Job Step bean
- Batch data stream interface
- Checkpoint algorithm
- Results algorithm

7.6.1 Batch Job Step bean

The Batch Job Step bean is a CMP entity bean. The fields that are persisted to the database are jobID and stepID, so during the lifetime of a Batch Job Step bean the data in those fields do not change.

Extended Deployment comes with a key class implementation for the Batch Job Step bean. This is necessary because the bean uses two CMP fields as primary key: jobID and stepID. This key class, called com.ibm.websphere.batch.BatchJobStepKey, can be found in batchruntime.jar.

> **Note:** There is no strong rationale for using a CMP entity bean for the Batch Job Step bean. However, the batch programming model for Extended Deployment V6.0 mandates this. Extended Deployment V6.1 will most likely provide a batch programming model that does not involve CMP entity beans.

The Batch Job Step bean is actually a *local* CMP entity bean: it can only be called locally by the LREE. The interface that the bean needs to implement is com.ibm.websphere.batch.BatchJobStepInterface. Since this interface cannot be changed, the same holds true for the local interface and the local home interface. These interfaces are provided by the Extended Deployment runtime and are packaged in batchruntime.jar:

- com.ibm.websphere.batch.BatchJobStepLocalHomeInterface
- com.ibm.websphere.batch.BatchJobStepLocalInterface

As discussed in the beginning of section 7.2, “Building a sample batch application” on page 224, there are two different options for implementing the business logic for a Batch Job Step bean:

1. Implementation in the actual bean class itself, in other words the bean class will read similar to the following:

```
public abstract class SampleStep1Bean implements
javax.ejb.EntityBean, com.ibm.websphere.batch.BatchJobStepInterface
```

2. Implementation of the BatchJobStepInterface in a separate Java class, for example com.ibm.sample.batch.SampleStep1. In this case, the actual Bean class extends this class:

   ```
   public abstract class SampleStep1Bean extends
   com.ibm.itso.sample.batch.SampleStep1 implements
   javax.ejb.EntityBean
   ```

The second option provides a way to implement the business logic outside of the Batch Job Step CMP entity bean. Also, this option is likely to ease a migration to a future batch programming model for Extended Deployment that does not use CMP entity beans.

As mentioned previously, the data in jobID and stepID does not change during the life cycle of the Batch Job Step bean. Since no other applications can update these fields in the database either, it is very attractive to enable CMP caching. However, there is another more compelling reason that requires us to enable CMP caching. The jobID and stepID CMP fields do not contain all the state held in the Batch Job Step bean. Hence, when the EJB container decides to passivate a bean instance of the Batch Job Step bean, any information that is not contained in jobID and stepID is lost. An example is a handle to a batch data stream that you might have obtained in the createJobStep() method. In other words, this would *break* the programming model.

In order to prevent this from happening, we need to make sure that the bean instance is always kept in memory by the EJB container. Either CMP caching option A or option B will accomplish this; however, since our application LREE has exclusive access to the database we should opt for option A. This will only read the data from the database once upon activation of the bean, instead of at the start of every transaction. This significantly reduces the total number of reads from the database, thus improving performance.

> **Note:** Although theoretically, option B would work, it has not been tested and the product documentation only recommends option A.
>
> More information about EJB caching in general can be found in section 15.4.1 of the IBM Redbook *WebSphere Application Server V6 System Management & Configuration Handbook*, SG24-6451.

CMP caching can be configured on the EJB deployment descriptor, as shown in Figure 7-29 on page 271. Details on how to configure this exactly are provided in section 7.2.6, “Configure EJB deployment descriptor” on page 238.

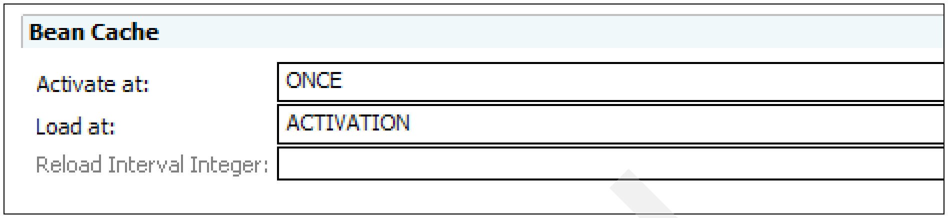


Figure 7-29 Configuring CMP caching option A on the EJB deployment descriptor

Interface com.ibm.websphere.batch.BatchJobStepInterface

`public void setProperties(Properties arg0)`

This method is called by the LREE to pass in the properties that are defined in the xJCL. This happens before createJobStep() is called. You need to implement this method in order to be able to obtain any properties you have set in the xJCL.

`public Properties getProperties()`

This method is not called by the LREE but is provided for consistency reasons.

`public void createJobStep()`

This method is called by the LREE before it starts processing the business logic of the batch job, for example, **`processJobStep()`**. Typically you would lookup any resources you need for your business logic.

For example, you would obtain a handle to your batch data stream here (as shown in Example 7-8 on page 272).

`public int processJobStep()`

This method is called by the LREE in the batch loop and contains the actual business logic of this step of the batch job. Any resources used by the business logic should have been obtained in createJobStep() and should be released in destroyJobStep().

Note that this method is always called under the scope of a global transaction. For transactional work, we recommend that you use the same transaction. When calling EJBs for example, the transaction attributes should be “required”, “supported” or “mandatory”.

The return code of this method is used by the LREE to determine what to do next. The return codes available are listed in Table 7-4 on page 272. For example, if a job step completes, the processJobStep() method returns BatchConstants.STEP_COMPLETE.

Table 7-4 Different return codes for processJobStep()

Return code	Job step finished?	Execute checkpoint?
STEP_CONTINUE	No	Depends on checkpoint algorithm
STEP_COMPLETE	Yes	Depends on checkpoint algorithm
STEP_CONTINUE_FORCE_CHECK POINT	No	Always
STEP_COMPLETE_FORCE_CHECK POINT	Yes	Always

`public int destroyJobStep()`

This method is called by the LREE after the batch loop completes. You should release/close any resources that you might have opened in **`createJobStep()`**.

The return code of this method is used by the LREE when it calls **`fireResultsAlgorithms()`** on the results algorithms associated with the job step.

Example 7-8 Obtaining a batch data stream in createJobStep() method of the Batch Job Step bean

```
package com.ibm.itso.sample.batch.ejb;

import com.ibm.websphere.batch.BatchDataStreamMgr;
import com.ibm.websphere.batch.BatchContainerDataStreamException;
import com.ibm.websphere.batch.JobStepID;

public abstract class SampleStep1Bean
   extends
      com.ibm.itso.sample.batch.SampleStep1 implements javax.ejb.EntityBean {

   private JobStepID id = new JobStepID(this.getJobID(), this.getStepID());
   private String inputBDSLogicalName = "inputBDS";

   public void createJobStep() {
      try{
         inputBDS = BatchDataStreamMgr.getBatchDataStream(inputBDSLogicalName,
id.getJobstepid());
      } catch(BatchContainerDataStreamException bcdse){
         System.out.println(bcdse);
      }
```

```
    }
}
```

7.6.2 Batch data stream interface

Access to data from the business logic in the Batch Job Step bean is provided through batch data stream objects. These objects are implementations of the batch data stream interface, which is described in detail below. The batch data stream objects provide an abstraction layer between your business logic and the actual data it works with.

A typical batch job step might read records from one batch data stream, do something with the data, and write (modified) data records to another batch data stream. An important feature of these objects is that the LREE can *position* them. Before the batch job step enters the batch loop, the LREE will position all associated batch data streams to either their initial checkpoint or—in the case of a restarted batch job step—to their last committed checkpoint.

Interface com.ibm.websphere.batch.BatchDataStream

`public String externalizeCheckpointInformation()`

This method is called by the LREE to obtain a checkpoint just before the commit of a global transaction when executing the batch loop. The LREE will persist the `String` to the LREE database at commit.

`public void internalizeCheckpointInformation(String arg0)`

This method is called by the LREE to pass the last executed checkpoint information to the batch data stream. The LREE retrieves this information from the LREE database and passes it in as a `String`. This typically happens when a job is restarted. Note that this does not position the batch data stream to the checkpoint, this method will only pass the checkpoint information!

`public void initialize(String arg0, String arg1) throws BatchContainerDataStreamException`

This method is called by the LREE during the initialization of the job step, right after calling **`setProperties()`**. The **`initialize()`** method is where you can first use the properties set on the batch data stream in the xJCL.

`arg0` - The logical name of the batch data stream, as defined in the xJCL.
`arg1` - String representation of the **`com.ibm.websphere.batch.JobStepID`**, which is a pair of **`jobID`** and **`stepID`**.

`public void positionAtInitialCheckpoint() throws BatchContainerDataStreamException`

This method is called by the LREE after the batch data stream is opened and the job step is running for the first time, for example, the batch job was not restarted.

`public void positionAtCurrentCheckpoint() throws BatchContainerDataStreamException`

This method is called by the LREE after the batch data stream is opened when a batch job is restarted. Note that the LREE will call **`internalizeCheckPointInformation()`** first to provide the checkpoint information to the batch data stream.

`public void open() throws BatchContainerDataStreamException`

This method is called by the LREE after the **`initialize()`** method is called. Any resources used by your batch data stream implementation should be opened here. For example, you might open a connection to a database using a JDBC data source here.

`public void close() throws BatchContainerDataStreamException`

This method is called by the LREE after **`destroyJobStep()`** is called on the Batch Job Step bean. Any resources opened by your batch data stream implementation in the **`open()`** method should be closed here.

`public String getName()`

This method returns the logical name of the batch data stream, as configured in the xJCL.

`public void setProperties(Properties arg0)`

This is the first method that is being called by the LREE on each batch data stream, just after it is created. It is used to pass in the batch data stream properties that were configured in the xJCL.

`public Properties getProperties()`

This method is not called by the LREE but is provided for consistency reasons.

`public void intermediateCheckpoint()`

This method provides a placeholder for any methods you need to call just after the LREE commits a global transaction.

`public void initialize (Properties arg0)`

This method is called by the LREE to pass in the properties that are defined in the xJCL. This happens before createJobStep() is called. You need to implement this method in order to obtain any properties you set in the xJCL.

7.6.3 Checkpoint algorithm

As discussed in section 7.5.2, “Executing the batch loop phase” on page 263, the LREE uses a checkpoint algorithm to determine when to commit global transactions when running the business logic in the batch loop.

You can implement your own checkpoint algorithm, and we discuss this in detail below. However, Extended Deployment also ships two checkpoint algorithms that are ready to be used directly. Table 7-5 lists both checkpoint algorithms.

Table 7-5 Checkpoint algorithms provided with Extended Deployment

Checkpoint algorithm	Implementation class
Time-based	**com.ibm.wsspi.batch.checkpointalgorithms.timebased**
Record-based	**com.ibm.wsspi.batch.checkpointalgorithms.recordbased**

Time-based checkpoint algorithm

The time-based checkpoint algorithm commits transactions at certain time intervals. This interval can be configured using the *interval* property, as shown in Example 7-9. Another property that can be configured for the time-based checkpoint algorithm is *TransactionTimeout*. If it is not configured in the xJCL file, a default of 60 seconds is used.

Note: Make sure that the value for the TransactionTimeout always exceeds the interval configured for the time-based checkpoint algorithm!

Example 7-9 Configuring the Extended Deployment time-based checkpoint algorithm in xJCL.

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
   <checkpoint-algorithm name="timebased">
      <classname>com.ibm.wsspi.batch.checkpointalgorithms.timebased</classname>
      <props>
         <prop name="interval" value="15" />
         <prop name="TransactionTimeout" value="30" />
      </props>
   </checkpoint-algorithm>
...
</job>
```

Record-based checkpoint algorithm

The time-based checkpoint algorithm commits transactions after processing a certain number of records. That is, the batch loop calls the processJobStep() method a certain number of times before committing the transaction. This number can be configured using the recordcount property, as shown in Example 7-10. If not specified, the algorithm uses a value of 10.000.

Another property that can be configured is TransactionTimeout. If it is not configured in the xJCL file, a default of 60 seconds is used.

> **Note:** Make sure that the value for the TransactionTimeout always exceeds the time it takes for the job step to process recordcount records!

Example 7-10 Configuring the Extended Deployment record-based checkpoint algorithm in xJCL

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
   <checkpoint-algorithm name="recordbased">
      <classname>com.ibm.wsspi.batch.checkpointalgorithms.recordbased</classname>
      <props>
         <prop name="recordcount" value="1000" />
         <prop name="TransactionTimeout" value="60" />
      </props>
   </checkpoint-algorithm>
...
</job>
```

Interface com.ibm.wsspi.batch.CheckpointPolicyAlgorithm

`public boolean initialize(CheckpointAlgorithm arg0)`

This method is called by the LREE to allow the checkpoint algorithm to retrieve associated properties defined in the xJCL. The argument **`arg0`** is a **`CheckpointAlgorithm`** object, which is described in more detail in the next section. Note that any other initialization should be implemented in this method as well.

`public String getAlgorithmName()`

This method returns the name of the algorithm. It is primarily used for logging and debugging.

`public boolean ShouldCheckpointBeExecuted()`

The checkpoint algorithm returns **`true`** if a checkpoint should be executed at this point. This method is called by the LREE after the **`processJobStep()`** call on the Batch Job Step bean returns. This statement is true in *most* cases. Please refer to “Executing the batch loop phase” on page 263 for more details.

`public int getRecommendedTimeOutValue()`

Just before the LREE starts a new global transaction in the execution flow of a job step, it obtains the TransactionTimeout property for the associated checkpoint algorithm. The value returned by this method is set as transaction timeout by the LREE on the new global transaction.

`public void startCheckpoint()`

This method is called by the LREE just before it starts a new global transaction in the execution flow of a job step. This call is made to inform the checkpoint algorithm that a transaction is about to be started.

`public void stopCheckpoint()`

This method is called by the LREE just after it commits the global transaction in the execution flow of a job step. This call is made to inform the checkpoint algorithm that a transaction was committed.

Class com.ibm.wsspi.batch.xjcl.CheckpointAlgorithm

This class contains the following public fields to provide access to the data in the xJCL.

`public String classname`

Name of the class as specified in the element **`<class>`**.

`public String name`

Name of the checkpoint algorithm as specified in the **`name`** attribute of the **`<checkpoint-algorithm>`** element.

`public int numofprops`

Number of attributes specified under the **`<props>`** element.

`public String[] propname`

String array containing the *names* of the attributes specified under the **`<props>`** element.

`public String[] propvalue`

String array containing the *values* of the attributes specified under the **`<props>`** element.

7.6.4 Results algorithm

Results algorithms are an *optional* feature of the batch programming model. A results algorithm allows for two types of actions to occur at the end of a batch step:

- To influence the *return code of the batch job* based on the return code of the batch step that just ended.

> **Note:** There are two types of return codes: the return code of an individual batch step and the return code of the batch job to which the step belongs.

- To provide a *place holder for triggers* or actions to take based on various step return codes.

Results algorithms are declared in xJCL and then applied to batch steps in the xJCL. More details can be found in section 7.7.4, "Results algorithm" on page 284.

> **Note:** Multiple results algorithms can be applied to a batch step.

A results algorithm is implemented as a simple Java class. This class has to implement the com.ibm.wsspi.batch.ResultsAlgorithm interface. This provides customers with flexibility to implement their own results algorithms. WebSphere Extended Deployment 6.0 comes with a simple and ready-to-use results algorithm out-of-the-box. Its implementation class is com.ibm.wsspi.batch.resultsalgorithms.jobsum, and all the results algorithm does is return the highest job step return code as return code for the job.

At the end of a batch step, the LREE checks the xJCL of the batch job to see which results algorithms to invoke. For each results algorithm specified, the LREE passes to the algorithm the return code of the batch step and the current return code of the batch job in the LREE database. The results algorithm can then take any action based upon the return codes passed in. The algorithm then passes a return code for the batch job back to the LREE which is persisted to the LREE database as the current return code of the batch job. This return code can be the same as the return code that the LREE passed to the results algorithm in the first place or it can be different depending on logic coded into the results algorithm.

Interface com.ibm.wsspi.batch.ResultsAlgorithm

`public boolean initialize(ResultsAlgorithm arg0)`

This method is called by the LREE to allow the results algorithm to retrieve associated properties defined in the xJCL. Note that any other initialization should be implemented in this method as well. A boolean return value of **`true`** indicates that the initialization was successful. The argument **`arg0`** is a **`ResultsAlgorithm`** object, which is described in more detail in the next section.

public String getAlgorithmName()

This method returns the name of the algorithm. It is primarily used for logging and debugging.

public int fireResultsAlgorithms(String arg0, String arg1, int arg2 int arg3)

This method is called by the LREE at the end of the batch step to which the results algorithm is applied. The return code returned by the **destroyJobStep()** method of the step is passed in together with the current return code of the batch job. This enables the results algorithm to fire triggers for certain return codes. The integer returned by this method becomes the new return code for the batch job.

arg0 - The id of the job.
arg1 - The name of step.
arg2 - The return code of the batch step for which this algorithm is being fired.
arg3 - The current return code of the batch job.

Class com.ibm.wsspi.batch.xjcl.ResultsAlgorithm

This class contains the public fields listed below to provide access to the data in the xJCL.

public String classname

Name of the class as specified in the element **<class>**.

public String name

Name of the results algorithm as specified in the **name** attribute of the **<results-algorithm>** element.

public int numofprops

Number of attributes specified under the **<props>** element.

public String[] propname

String array containing the *names* of the attributes specified under the **<props>** element.

public String[] propvalue

String array containing the *values* of the attributes specified under the **<props>** element.

7.7 Batch application syntax in xJCL

In Extended Deployment, batch jobs are defined in xJCL, so we need to write an xJCL file that contains information about the job we want executed.

Example 7-11 contains a condensed version of an xJCL file that can be used to submit a batch job. Note that a “+” denotes a collapsed element.

Example 7-11 Overview of xJCL batch sample XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <jndi-name>ejb/com/ibm/websphere/samples/BatchJob</jndi-name>
   + <step-scheduling-criteria>
   + <checkpoint-algorithm name="timebased">
   <results-algorithms>
      + <results-algorithm name="jobsum">
   </results-algorithms>
   <job-step name="Step1">
      <jndi-name>ejb/Step1Bean</jndi-name>
      <checkpoint-algorithm-ref name="timebased" />
      <results-ref name="jobsum" />
      + <batch-data-streams>
      + <props>
   </job-step>
   <job-step name="Step2">
      <jndi-name>ejb/Step2Bean</jndi-name>
      <checkpoint-algorithm-ref name="timebased" />
      <results-ref name="jobsum" />
      + <batch-data-streams>
      + <props>
   </job-step>
</job>
```

7.7.1 Main elements in xJCL file for batch jobs

We will now go through the elements shown in Example 7-11 and explain those in detail. Obviously, the first line points out that we are dealing with a file in XML format. The main element for xJCL is **<job>**, which has the following two attributes:

- The **`name`** attribute defines the name of the application to which we are submitting a job. The LongRunningScheduler application will match this name against the names of deployed enterprise applications in the long-running execution environment before dispatching a job.
- The **`xmlns:xsi`** attribute defines the namespace to use for the element **`<job>`** and its sub-elements.

We will now go through the different sub-elements for the <job> element in the xJCL file. After that we will zoom in on the <job-step> sub-element since this is the most complex one.

Sub-elements for `<job>`

`<jndi-name>` This element acts as a reference to the. Batch Job Step bean. Note that this JNDI name is a name in the WebSphere namespace and not an EJB reference! When deploying the enterprise application the Batch Job Step bean needs to be referenced by this JNDI name.

`<step-scheduling-criteria>` This element sets the scheduling criteria for the job. At the moment WebSphere Extended Deployment 6.0 only provides sequential as scheduling mode. Refer to "Scheduling criteria" on page 282 for more details.

`<checkpoint-algorithm>` This configures a checkpoint algorithm. The **`name`** attribute provides a logical name that can be used to refer to this algorithm from a batch step.

`<results-algorithms>` Configures one or more results algorithms for the job. Each sub-element **`<results-algorithm>`** has a **`name`** attribute that can be used to refer to this algorithm from a batch step.

`<job-step>` Each batch job can have one or more batch steps. Each of these steps is defined as a **`<job-step>`** sub-element.

Sub-elements for `<job-step>`

`<jndi-name>` This is a logical JNDI name for the batch step. It has to match the ejb-reference declared in the Batch Job Controller bean for this Batch Job Step bean.

`<checkpoint-algorithm-ref>` The **`name`** attribute of this element defines which checkpoint algorithm to apply to this batch step. Note that this refers to a **`<checkpoint-algorithm>`** sub-element of **`<job>`**. Refer to "Checkpoint algorithm" on page 283 for more details.

`<results-ref>` The **`name`** attribute of this element defines which results algorithm to apply to this batch step. Note that this refers to one of the sub-elements of the **`<results-algorithms>`** sub-element of **`<job>`**. Multiple results algorithms can be associated with one job step. Refer to “Results algorithm” on page 284 for more details.

> **Attention:** Be careful when using multiple results-algorithms for a job step. The algorithms are processed in the order they are specified.

`<batch-data-streams>` Zero or more batch data streams for the job step can be defined here. Refer to “Batch data streams” on page 285 for more details.

`<props>` Additional properties that are used by the job-step implementation can be provided here. These properties can be accessed through the method getProperties() of the Batch Step Entity Bean.

`<step-scheduling>` Conditions on when to execute the job step can be configured here. For details refer to “Scheduling criteria” on page 282.

7.7.2 Scheduling criteria

Scheduling criteria provides a mechanism to control how to execute the various job steps defined in a batch job. At the moment, WebSphere Extended Deployment 6.0 only provides one scheduling criteria: the *sequential* scheduling mode. The scheduling mode is set through the element **`<step-scheduling-criteria>`** in the xJCL file.

> **Note:** The element **`<step-scheduling-criteria>`** and its subelement **`<scheduling-mode>`** are required in each batch xJCL file.

The sequential scheduling mode basically determines that each job step is carried out sequentially. However, one can configure conditions whether a certain step is executed or not. This is done using the sub-element **`<step-scheduling>`** on a job step.

Example 7-12 on page 283 shows how to configure a batch job that only carries out the second job step if certain conditions are met. In this example, job step “Step2” is only executed if “Step1” returns 0 or 1 as a return code.

> **Note:** The return code of a job step is the integer returned by the method **`destroyJobStep()`** of the Batch Job Step bean associated with the job step.

Example 7-12 Using scheduling criteria in a batch job

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
<step-scheduling-criteria>
   <scheduling-mode>sequential</scheduling-mode>
</step-scheduling-criteria>
...
<job-step name="Step1">
...
</job-step>
<job-step name="Step2">
<step-scheduling condition="OR">
   <returncode-expression step="Step1" operator="eq" value="0" />
   <returncode-expression step="Step1" operator="eq" value="1" />
</step-scheduling>
...
</job-step>
</job>
```

7.7.3 Checkpoint algorithm

Multiple **`<checkpoint-algorithm>`** elements can be defined in a single xJCL file. The only attribute of this element is **`name`**. It is used in the xJCL to refer to the algorithm.

> **Note:** If you define multiple checkpoint algorithms, the name attribute for each one has to be unique in the xJCL file.

Sub-elements for `<checkpoint-algorithm>`

`<classname>` This is the name of the class that provides the algorithm. This class has to implement the interface **`com.ibm.wsspi.batch.CheckpointPolicyAlgorithm`**.

`<props>` Additional properties that are used by the checkpoint algorithm implementation can be provided here. These properties are set on the checkpoint algorithm.

Instead of writing your own checkpoint algorithm, you could use one of the two checkpoint algorithms that come with Extended Deployment. Refer to 7.6.3, “Checkpoint algorithm” on page 275 for more details. Example 7-13 on page 284 shows how to define a checkpoint algorithm called “timebased” using the time-based checkpoint algorithm provided by Extended Deployment.

Example 7-13 Checkpoint algorithm

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
   <checkpoint-algorithm name="timebased">
      <classname>com.ibm.wsspi.batch.checkpointalgorithms.timebased</classname>
      <props>
         <prop name="interval" value="15" />
      </props>
   </checkpoint-algorithm>
...
</job>
```

7.7.4 Results algorithm

Multiple **`<results-algorithm>`** elements can be defined in a single xJCL file. The only attribute of this element is **`name`**. It is used in the xJCL to refer to the algorithm.

> **Note:** If you define multiple checkpoint algorithms, the name attribute for each one has to be unique in the xJCL file.

Sub-elements for `<results-algorithm>`

`<classname>` This is the name of the class that provides the algorithm. This class has to implement the interface **`com.ibm.wsspi.batch.CheckpointPolicyAlgorithm`**.

`<props>` Additional properties that are used by the checkpoint algorithm implementation can be provided here. These properties are set on the checkpoint algorithm.

As discussed in section 7.6.4, “Results algorithm” on page 277, instead of writing your own results algorithm you could use the jobsum results algorithm that comes with WebSphere Extended Deployment 6.0. Example 7-14 shows how to define this algorithm in xJCL.

Example 7-14 Results algorithm

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
   <results-algorithms>
      <results-algorithm name="jobsum">
```

```
        <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
      </results-algorithm>
   </results-algorithms>
...
</job>
```

7.7.5 Batch data streams

Each job step in a batch application can have zero or more batch data streams associated with it. In the xJCL, each job step can have multiple **<bds>** elements defined as sub-elements of the element **<batch-data-streams>.** See Example 7-15.

Sub-elements for <bds>

<logical-name> This is the name of the class that provides the algorithm. This class has to implement the interface **com.ibm.wsspi.batch.CheckpointPolicyAlgorithm**.

<impl-class> Additional properties that are used by the checkpoint algorithm implementation can be provided here. These properties are set on the checkpoint algorithm.

<props> Additional properties that are used by the batch data stream implementation can be provided here. The LREE sets these properties by calling setProperties() on the BatchDataStream object, just before initialization. Refer to section 7.5.1, “Prepare for execution phase” on page 261 for more details.

Example 7-15 Configuring batch data streams for a job step in xJCL

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="BatchSample"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
   <job-step name="Step1">
...
      <batch-data-streams>
         <bds>
            <logical-name>myinput</logical-name>
            <impl-class>com.ibm.itso.batch.BatchInputStreamImpl</impl-class>
            <props>
               <prop name="FILENAME" value="c:\temp\input" />
            </props>
         </bds>
         <bds>
```

```
            <logical-name>myoutput</logical-name>
            <impl-class>com.ibm.itso.batch.BatchOutputStreamImpl</impl-class>
            <props>
               <prop name="FILENAME" value="c:\temp\output" />
            </props>
         </bds>
      </batch-data-streams
   </job-step>
...
</job>
```

8

Building compute-intensive applications

This chapter is intended for developers who need to build compute-intensive applications that run on WebSphere Extended Deployment. The topics covered in this chapter are as follows:

- Overview of compute-intensive applications
- Building a compute-intensive application
- Compute-intensive application programming model
- Compute-intensive application syntax in xJCL

8.1 Overview of compute-intensive applications

A compute-intensive application is packaged as a standard J2EE EJB module inside a J2EE enterprise application. This application is deployed together with the LREE application on the same dynamic cluster.

This section describes the components that a compute-intensive application consists of. The interaction of those components is described in 8.1.2, “Interaction with the long-running execution environment” on page 289.

8.1.1 Components in compute-intensive applications

A compute-intensive application is packaged as a J2EE application in which an EJB module is included. Every compute-intensive application consists of exactly one controller bean, CIController. This is a stateless session bean, the implementation of which is provided by Extended Deployment runtime. The bean only needs to be declared in the deployment descriptor of the EJB module. Upon deployment, the JNDI name of the controller bean is used as a reference in the xJCL.

Each compute-intensive application can consists of one or more CIWork classes. Using xJCL, you can associate the job and the CIWork class to be executed. The CIWork class is a POJO class that implements **`com.ibm.websphere.ci.CIWork`** interface. This interface is an extension of **`commonj.work.Work`** interface and adds two accessor methods to be able to handle the properties via xJCL.

Figure 8-1 shows the components of a compute-intensive application for WebSphere Extended Deployment.

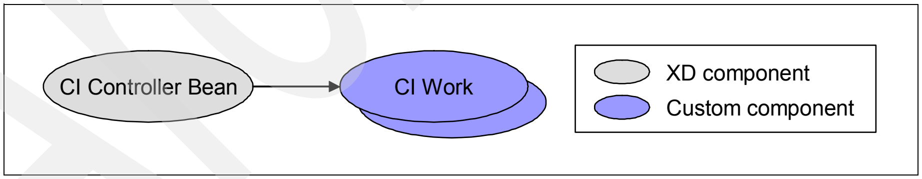


Figure 8-1 Components in a computer intensive application

The code you need to develop for the compute-intensive application consists of the CIWork classes that contain the business logic. As for the CIController bean, you need only to declare it in the deployment descriptor and point to the implementation class provided by Extended Deployment.

8.1.2 Interaction with the long-running execution environment

A compute-intensive application job in WebSphere Extended Deployment is submitted as xJCL to the LRS, which dispatches it to the LREE. Figure 8-2 shows how the components of a compute-intensive application interact with the LREE.

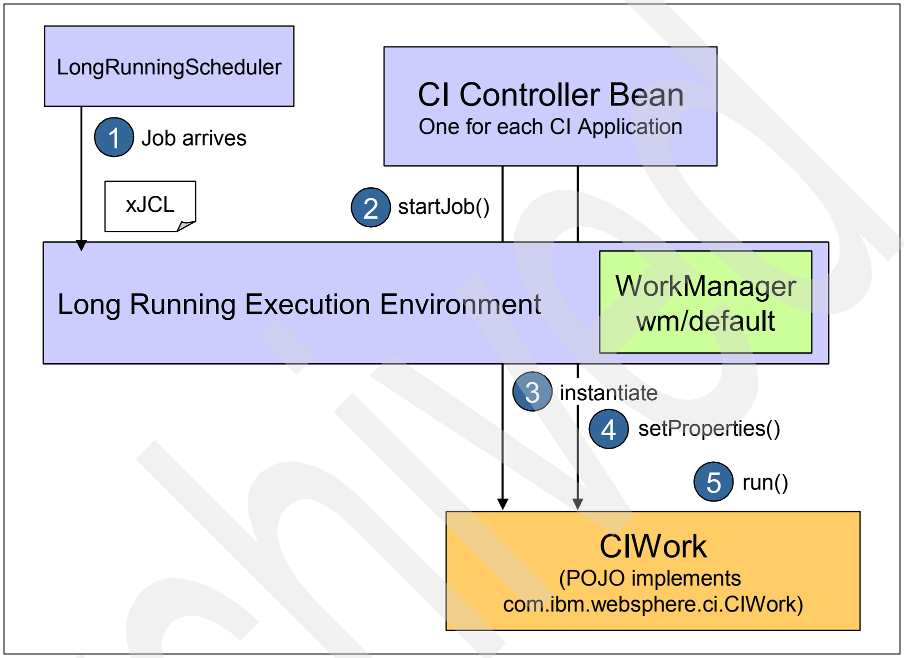


Figure 8-2 Interaction of a compute-intensive application with the LREE

The flow is as follows:

1. The LRS dispatches a job to one of the LREE application servers:
2. The LREE will lookup the controller bean of the application using the JNDI name provided in the xJCL. It calls the startJob() method on the controller bean.
3. The controller bean instantiates the CIWork object specified in the xJCL for the job step using the no-argument constructor. Note that CIWork is not called directly by controller bean but called by a class in the LREE.
4. The controller bean calls the setProperties() method to pass any properties defined in the xJCL.
5. The controller bean looks up the work manager defined in the deployment descriptor of the application's EJB module, and uses it to call the run() method of the CIWork object asynchronously.

If the job is not cancelled and the run() method returns without throwing an exception, the job is deemed to have completed successfully.

8.2 Building a compute-intensive application

This section shows how to build and implement a very simple compute-intensive application in Rational Application Developer V6.0. A compute-intensive application consists of a controller bean and one or more CIWork classes. These are packaged as a J2EE EAR file. The components involved are shown in Figure 8-3.

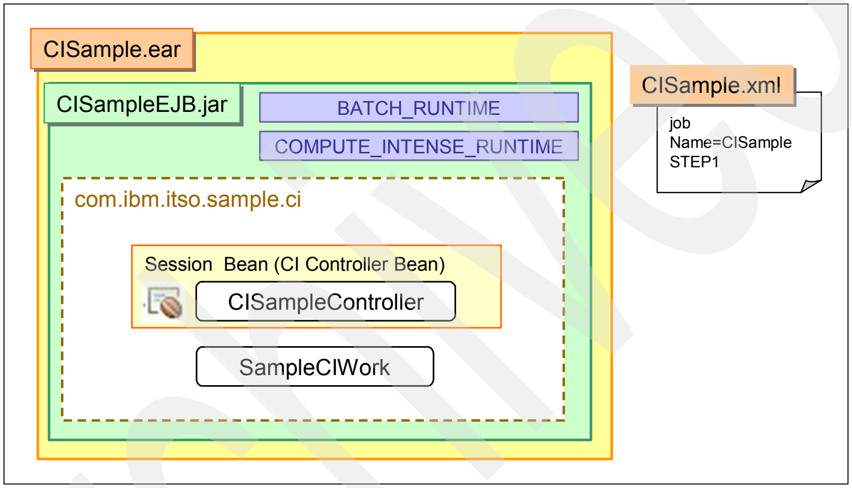


Figure 8-3 Compute-intensive application and packaging

The following is an overview of the steps needed to build and package a simple compute-intensive application using Rational Application Developer.

- Create an enterprise project and EJB project.
- Create the controller bean.
- Configure the EJB module.
- Create and implement a CIWork class.
- Create the xJCL.

8.2.1 Creating an enterprise project and EJB project

The first step is to set up the basic structure for the application:

1. Create a new EJB project and enterprise project to hold the application. In our example, we named the EJB module CISampleEJB and the EAR project was called CISample. If you are not familiar with creating projects in Rational Application Developer, see section 7.2.1, “Create an enterprise project and EJB project” on page 225 for more information.
2. Add the Extended Deployment JAR files to the Java build path of the EJB module. This is a compute-intensive application, so you need to add both *<rad_install>*\runtimes\base_v6\lib\batchruntime.jar and *<rad_install>*\runtimes\base_v6\lib\gridapis.jar.

   In section 6.6, “Configure the development environment” on page 204, you can see how to create variables to use to add the JAR files. If you use this technique, you would add the JAR files using both the BATCH_RUNTIME and the COMPUTE_INTENSE_RUNTIME variables.
3. Create a package in the EJB module to hold the classes. We specified `com.ibm.itso.sample.ci` for the package name.

8.2.2 Creating the controller bean

Now we need to declare the controller bean. Note that we do not need to implement this bean. We only have to declare it in the deployment descriptor and point to the implementation class that is provided by WebSphere Extended Deployment.

1. Expand the deployment descriptor in your EJB Project.
2. Right-click **Session beans** and select **New** → **Session Bean**.
   a. Ensure that **Session Bean** is selected.
   b. Specify a name for the bean (CISampleController).
   c. Specify the package for the bean (com.ibm.itso.sample.ci).
   d. Click **Next**.
3. Specify the following options for the new enterprise bean, as shown in Figure 8-4 on page 292.

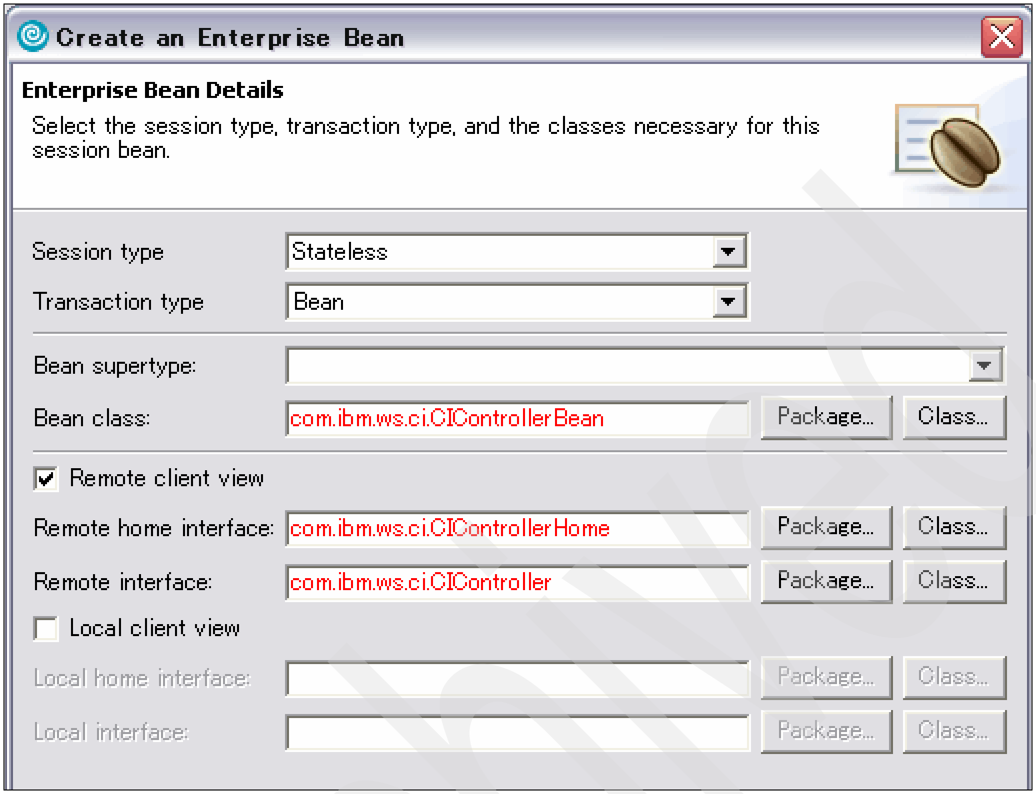


Figure 8-4 Specifying implementation classes for the new bean

4. Select **Finish** to create the bean.

8.2.3 Configuring EJB deployment descriptor

Next, there are two changes that need to be made in the deployment descriptor: configuring JNDI names and creating the WorkManager resource reference.

Configuring JNDI names

Rational Application Developer generates a default JNDI name for each EJB based on the name of the home interface. The new CIController bean would use the following name:

```
ejb/com/ibm/ws/ci/CIControllerHome
```

Multiple compute-intensive applications might be deployed in the same LREE, so it is a best practice to configure the CIController bean with its own unique JNDI name space binding to avoid a JNDI namespace conflict.

1. On the Bean tab of the deployment descriptor, select the newly created CISampleController controller bean.
2. In the right panel, change the JNDI name under WebSphere Bindings. Here we use `ejb/com/ibm/itso/sample/ci/CISampleController` as the JNDI name.
3. Save the deployment descriptor, but leave it open for the next step.

Creating the WorkManager resource reference

When the controller bean submits a job, it will use asynchronous beans to run the business logic in its own threads. The controller bean uses a work manager to accomplish this, therefore we need to configure a resource reference on the CIController bean that points to the default work manager on the LREE application servers.

1. Open the CISampleEJB deployment descriptor and select the Reference tab.
2. Select the controller bean, CISampleController.
3. Click **Add** to add a reference.
4. Choose **Resource Reference**, and click **Next**.
5. Enter the following:
   – Resource reference name: **wm/CIWorkManager**
   – Type: **commonj.work.WorkManager**
   – Authentication: **Container**
   – Sharing scope: **Shareable**
6. Click **Finish** to create the resource reference.
7. Now provide a JNDI name space binding for this resource reference
   a. Select the new resource reference **wm/CIWorkManager**.
   b. Under WebSphere Bindings, specify **wm/default** as the JNDI name.
8. Save your changes to the deployment descriptor.

8.2.4 Creating and Implementing a CIWork class

In this section, you create and implement the CIWork class.

Creating a CIWork class

We are now ready to create our CIWork class.

1. Expand the ejbModule and select the directory to hold the CIWork class, in our example, `com.ibm.itso.sample.ci`. Right-click the directory, and select **New → Class.**
2. Specify the following options for the new class as shown in Figure 8-5:
   a. Specify a name for the class, for example SampleCIWork
   b. Under Interfaces, click **Add** and add `com.ibm.websphere.ci.CIWork` as an interface.
3. Click **Finish**.

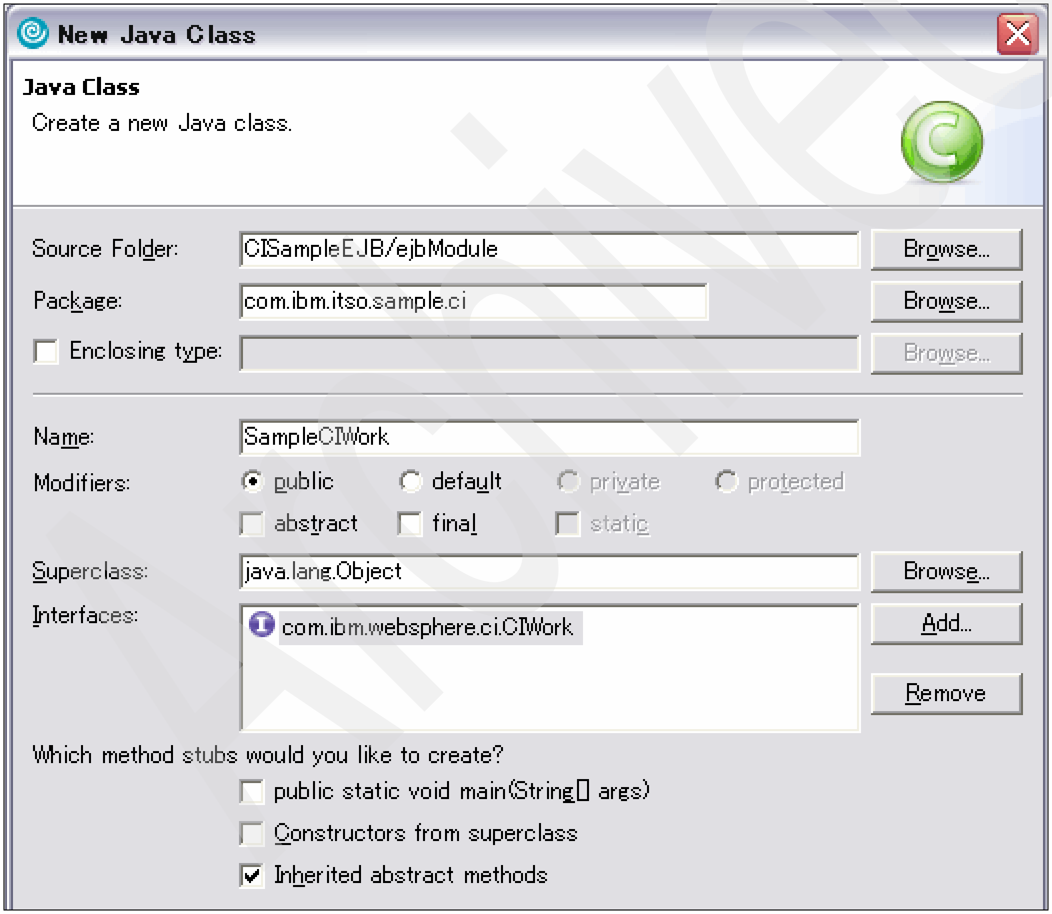


Figure 8-5 Creating the CIWork class.

Implementing a CIWork class

We now need to implement some business logic in the class **com.ibm.itso.sample.ci.SampleCIWork**. The detailed explanation of the method is in 8.3, "Compute-intensive application programming model" on page 297, but there are two things to note:

- The LREE instantiates the CIWork object using the no-argument constructor; therefore, the CIWork class *must* have a no-argument constructor.
- The isDaemon() method *must* return true, which means this Work object will work independent of the submitting component's life cycle. You are strongly encouraged to implement the release() method so that the run() method returns as soon as practical after this method is invoked.

Example 8-1 shows a simple CIWork implementation.

Example 8-1 Simple CIWork implementation sample

```
package com.ibm.itso.sample.ci;

import java.util.Map;
import com.ibm.websphere.ci.CIWork;

public class SampleCIWork implements CIWork {

   boolean _continue=true;
   Map props;

   public void setProperties(Map arg0) {
      System.out.println("SampleCIWork: setProperties");
      props = arg0;
   }
   public Map getProperties() {
      return null;
   }
   public void release() {
      System.out.println("SampleCIWork: release");
      _continue=false;
   }
   public boolean isDaemon() {
      System.out.println("SampleCIWork: isDaemon");
      return true; //Must be true
   }
   public void run() {
      System.out.println("SampleCIWork: run");

      //Prepare for the busines logic
```

```
      double timeToExecute = 60;
      try {
         if( props!=null && props.containsKey("timeToExecute")){
            timeToExecute=Double.parseDouble((String)props.get("timeToExecute"));
         }
      }catch (NumberFormatException nfe) {
         System.out.println("property specified is not a number. Using the default 
value");
      }

      while(_continue){
            double startTime = System.currentTimeMillis();
            double endTime =  startTime + timeToExecute*1000;
            int counter=0;
            double currentTime;
            while (((currentTime = System.currentTimeMillis()) < endTime) && 
_continue) {
            try{
               System.out.println("SampleCIWork: run " + counter++);
               Thread.sleep(10000);
            }catch(InterruptedException ie){
               ie.printStackTrace();
            }
            }
      }
   }
}
```

Example 8-2 shows a sample of the xJCL file that can be used to submit a job to the sample application.

Example 8-2 Sample xJCL to submit a job to the sample CI application.

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="CISample" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <jndi-name>ejb/com/ibm/itso/samples/ci/CISampleController</jndi-name>
   <job-step name="Step1">
      <classname>com.ibm.itso.sample.ci.SampleCIWork</classname>
   </job-step>
</job>
```

8.3 Compute-intensive application programming model

This section discusses the methods defined in the com.ibm.ws.ci.CIWork interface that every CIWork class must implement.

The com.ibm.ws.ci.CIWork extends the commonj.work.Work interface, an asynchronous beans interface in JSR 237. The com.ibm.ws.ci.CIWork interface also extends the java.lan.Runnable method. We will discuss the methods defined in these as well.

The structure is shown in Figure 8-6.

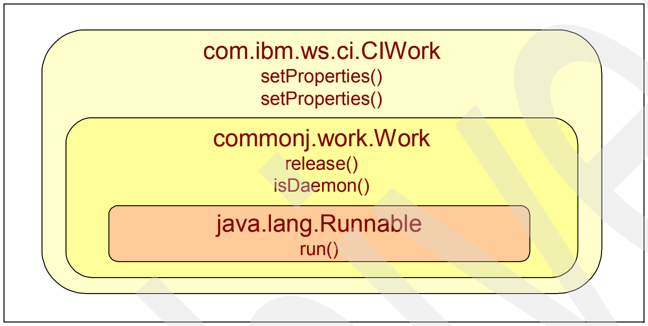


Figure 8-6 com.ibm.ws.ci.CIWork interface structure

Interface `com.ibm.ws.ci.CIWork`

This interface defines the following methods:

`public void setProperties(java.util.Map arg0)`

This method is called by LREE before run() is called, and it passes in the xJCL job properties in the java.util.Map argument. You need to implement this method so that you can use the job properties specified in xJCL.

`public Map getProperties()`

This is not called by the LREE but is provided for consistency reasons.

Interface `commonj.work.Work`

This interface defines the following methods:

`public void release()`

This method is called when the job is cancelled. It is the responsibility of the developer of the CI application to implement for the logic in this method to cause the run()

method to return promptly. The simplest way to implement this is to set the flag that is checked periodically in run().

public boolean isDaemon()

This method *must* return true. This tells the container that CIWork ran for an extended period so that the submitting container method will release the Work object and does not need to wait for the job to complete.

Interface java.lang.Runnable

This interface defines the following method:

public void run() This method is called by LREE after setProperties is called. In this method you need to implement the business logic. It is strongly encouraged to implement the logic so that this method returns as soon as practical after **release()** is invoked.

8.4 Compute-intensive application syntax in xJCL

Compute-intensive and batch applications are submitted using an xJCL file. The xJCL for compute-intensive applications is simpler than that used for batch applications.

The xJCL for a compute-intensive application consists of only one job step and does not have any checkpoint algorithms or BDS. You need to simply specify the controller bean and the CIWork class to be used and the properties to be passed to the job step.

Example 8-3 shows the structure of the xJCL for a compute-intensive application.

Example 8-3 Sample xJCL file for a compute-intensive application

```
<?xml version="1.0" encoding="UTF-8"?>
<job name="CISample" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jndi-name>ejb/com/ibm/itso/samples/ci/CISampleController</jndi-name>
  <job-step name="Step1">
    <classname>com.ibm.itso.sample.ci.SampleCIWork</classname>
      <props>
        <prop name="timeToExecute" value="60"/>
      </props>
  </job-step>
</job>
```

Top element: <job>

The top element for xJCL is `<job>`, which has the following two attributes:

- The **name** attribute is the name of the application the job is being submitted to. The LRS matches this name against the names of the enterprise applications deployed in the LREE dynamic cluster before dispatching a job. If the name does not match, the job fails.
- The **xmlns:xsi** attribute defines the name space to use for the <job> element and its sub-elements.

The <job> element has the following sub-elements:

<jndi-name>	This element is a reference to the controller bean. The JNDI name must be found in the WebSphere namespace.
<job-step>	Each compute-intensive job can consist of only a single job step.

Element:<job-step>

The <job-step> element has one attribute.

- The **name** attribute defines the name of the job step. This name can be specified arbitrarily.

The <job-step> element has the following sub-elements:

<class-name>	Specifies the fully-qualified name of the CIWork class to be executed in the job.
<props>	Additional properties that are used by the job-step implementation can be provided here. These properties can be accessed via the getProperties()method of the CIWork class.

Element: <props>

The <props> element has no attributes. It has the following sub-element:

<prop>	This element can have 0 or more <prop> sub-elements.

Element: <prop>

This element has two attributes:

- The **name** attribute defines the name of the property.
- The **value** attribute defines the value of the property.

Part 3

Data-intensive application extenders

The WebSphere Extended Deployment ObjectGrid is a high performance distributed transactional cache for storing Java objects that can greatly improve application performance and scalability. ObjectGrid can be used in the following ways:

- To improve the interaction between application services and underlying data sources for data-intensive applications.
- To reliably manage short-living session data within the application tier, removing the need to store the session data in a hardened data source.
- To accelerate SOA calls by utilizing distributed caching to cache previous SOA calls to back-ends.

- To provide a J2EE platform for ultra high-performance online transaction processing (OLTP) applications by using asymmetric clustering and application partitioning techniques.

This Part contains one chapter, Chapter 9, “HTTP session management” on page 303. This chapter shows how to use the ObjectGrid to manage short-lived HTTP session data within the application tier.

9

HTTP session management

This chapter describes how to use the ObjectGrid feature of WebSphere Extended Deployment to support HTTP session object persistency. ObjectGrid is high performance distributed transactional cache. An ObjectGrid can consist of just one JVM or any number of JVMs. These JVMs do not have to be in a WebSphere Application Server cell. WebSphere Extended Deployment provides a set of Java APIs that application programs can use to interact with the ObjectGrid. Using these APIs, application programs can store and retrieve data from the ObjectGrid.

In this chapter we discuss the use of the ObjectGrid to store HTTP session objects and work through setting up a simple example. This chapter contains the following topics:

- Session objects
- ObjectGrid session management
- Example - Single server
- XML file inter-relationship
- Example: sharing session objects among applications
- Example: ObjectGrid on separate server
- Adding a replica

9.1 Session objects

HTTP Sessions are java objects that are created by J2EE. applications to store session states across multiple HTTP requests, since HTTP is an essentially stateless protocol.

Typically when an application runs in an application server, it creates an HTTP session object. The application uses this session object as a place to store information about the status of a user within an application, for example the classic shopping cart scenario.

9.1.1 Replication of session objects

Replication in WebSphere Application Server is used to provide failover for user sessions. This creates an affinity for the user session to the application server where this session object is stored. Typically there is a process in front of the application server that ensures that requests from a specific user always return to the server where their session object exists. However, should the server fail or be stopped, the next request from the user is routed to another application server. Since the new server does not have access to the session object, any status information that was stored in the session object is unavailable, and the user has to start over.

To prevent this, WebSphere Application Server lets you configure a replication mechanism. You can configure WebSphere to replicate the session objects to a database or to another server using memory-to-memory replication. In the event that a server fails, the next request from the user flows to another server. The new server can retrieve the session object from the database or replication domain, and the user can continue from where they left off.

9.1.2 Replication with WebSphere Application Server

There are some characteristics of the basic session replication support provided by WebSphere Application Server session that you should understand in order to see how WebSphere Extended Deployment can help you expand your HTTP session replication capabilities.

- Cell boundary

  In WebSphere Application Server, session objects belong to a single cell. This means that access to the HTTP session object can only occur from servers within that cell. While technically you could configure two cells to access the same back-end database being used for replication, and load balance requests across the two cells, this is not a supported configuration.

- Replication not guaranteed

  Replication is not a guaranteed service. While the replication feature is robust, there is no retry capability.

- Applications and session objects

  The HTTP session object cannot be shared across different applications. For example if a user is accessing application A and application B, two session objects are created. Neither application can access information stored in the session object of the other application, even though it is for the same user. This is mandated by the servlet specification. WebSphere does provide an extension that allows Web applications within the same application to access a shared session object.

- Replication is done at the server level

  Replication is a usually enabled at the server level. Once enabled, every application in that server will have its session objects replicated. It is possible to disable replication at the server level and enable it at the application level; however, when using this approach where separate servers are set up as replication servers, the application must also be deployed to the replication servers, even though it is never executed there.

9.1.3 Replication with WebSphere Extended Deployment

Using ObjectGrid, you can overcome the limitations of WebSphere Application Server session replication.

- No cell boundary

  While ObjectGrid is a feature of Extended Deployment, it is not really part of the WebSphere infrastructure per se. ObjectGrid can be incorporated into use by any JVM process. It can, in effect, provide a transactional cache environment that can be accessed by any JVM process in use.

  This means that a session object stored in the grid by a WebSphere application server could be accessed by an application in another WebSphere cell or even from a non-WebSphere server.

- Guaranteed delivery

  The process the ObjectGrid uses to store session objects is more robust then the standard replication process. The ObjectGrid will retry saving a session object if the initial attempt fails.

- Cross application access

  There is a setting in the ObjectGrid process that allows different applications to access the session object for the same user. This is shown in section 9.5, “Example: sharing session objects among applications” on page 322.

- Replication set at application level

  Using ObjectGrid to store session objects is a process done on an application basis. It does not require the normal WebSphere replication process to be enabled.

9.2 ObjectGrid session management

Why use ObjectGrid for session management?

You may consider using ObjectGrid for session management for the following reasons:

- You already have an existing ObjectGrid environment and can make further use of it for managing session objects.
- You have a requirement to share session objects across applications.
- You need session failover capabilities that use a reliable (think guaranteed) session persistence mechanism.

Note: Before using ObjectGrid for HTTP session objects you should obtain fixes for the following and install on all WebSphere Application Servers or Mixed Server Environment installations that will be used:

- APAR PK35502
- APAR PK35551
- APAR PK35503

ObjectGrid can be used as a repository for HTTP session objects. Extended Deployment contains the ObjectGrid component, which consists of two parts: the ObjectGrid server and a client side set of libraries. Extended Deployment also contains an HTTP Session Manager facade. The HTTP Session Manager facade uses the ObjectGrid client libraries, which in turn communicate with the ObjectGrid. The HTTP Session Manager is not aware of the ObjectGrid setup.

The following is an overview of the steps involved in configuring an application to use the ObjectGrid for session object replication:

- Configure the ObjectGrid server-side XML files.
- Configure at least one JVM to be a member of the ObjectGrid.
- Update the application EAR file to be ObjectGrid-aware and deploy it.

- Access the application and verify the use of ObjectGrid.

9.2.1 Sample files

WebSphere Extended Deployment ships sample files for use in setting up an environment to use the ObjectGrid for session announcement. They are located in *<xd_install_root>*\optionalLibraries\ObjectGrid\session\samples, as shown in Figure 9-1.

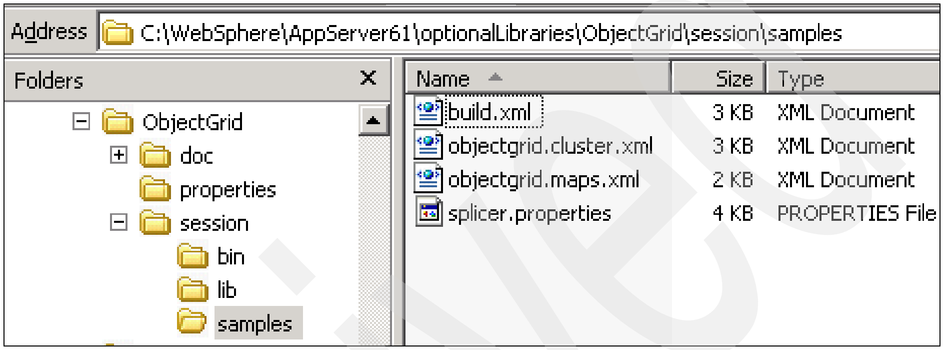


Figure 9-1 Sample files for session management with ObjectGrid

- build.xml

  Contains the customization properties in XML format, and is used if you use the ANT option to modify the web.xml file.

- objectgrid.cluster.xml

  This file is referred to as the ObjectGrid cluster file. It describes the ObjectGrid cluster configuration, for example, what servers are in the ObjectGrid.

- objectgrid.maps.xml

  This file is referred to as the ObjectGrid configuration file. The purpose of this file is usually to describe how your application will use the ObjectGrid. However in this case, it is not your application that is directly using the ObjectGrid, but rather the Extended Deployment HTTP Session Manager that is. As such, this sample file is configured to work with the Extended Deployment HTTP Session Manager. For the purposes of setting up a simple example, it does not need to be changed.

- splicer.properties

  This file contains configuration values that are stored in the web.xml file of the application.

Chapter 9 of the *ObjectGrid Programming Guide* describes the various XML elements that are used in these files. This information can be found starting with the Local ObjectGrid configuration heading. The guide can be downloaded from the following Web location:

http://www-1.ibm.com/support/docview.wss?uid=swg2700632

9.3 Example - Single server

This example illustrates how to set up session management to an ObjectGrid where the application creating the session objects and the ObjectGrid server are both running in the same application server.

> **Note:** It is unlikely that this approach would ever be used in a production environment, but the aim here is to use a very simple set up to illustrate how ObjectGrid is used for HTTP sessions. We will use this as a basis to expand upon in later sections.

The runtime environment for this example is shown in Figure 9-2.

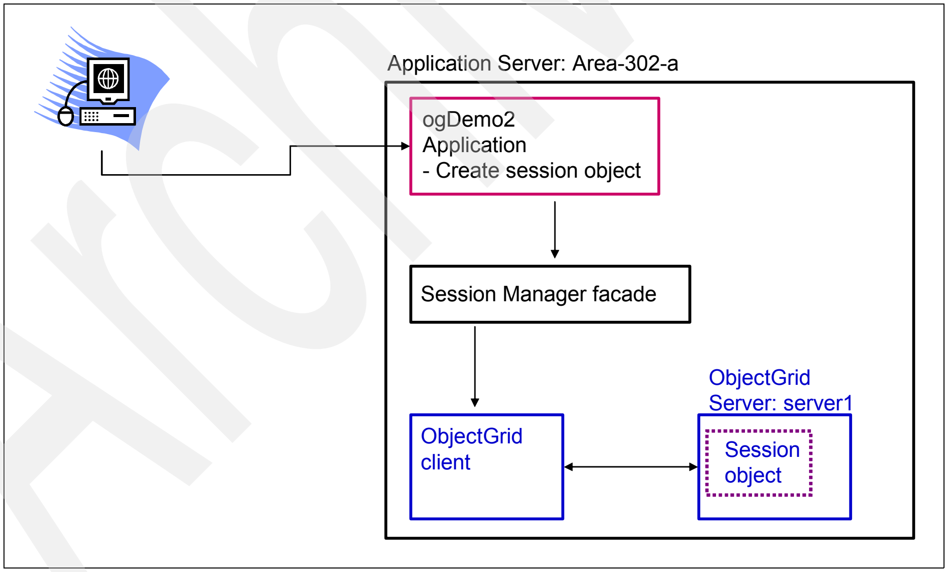


Figure 9-2 Application and ObjectGrid all in one server

For this example, we developed a small application that creates a session object with a few attributes. This application is called xd-ObjectGrid. Later, when we enable ObjectGrid for the application, it will be redeployed under the name ogDemo2.

We also created an application server called Area-302-a in an existing WebSphere Application Server V6.1 cell.

9.3.1 Install and run application without ObjectGrid

Before using ObjectGrid, we installed and tested our sample application into the application server. We then ran the application using this URL:

```
http://localhost:9088/xd-objGridWeb/ObjGridSessionDemo
```

After pressing the Submit button in the application once, we obtained the display shown in Figure 9-3.

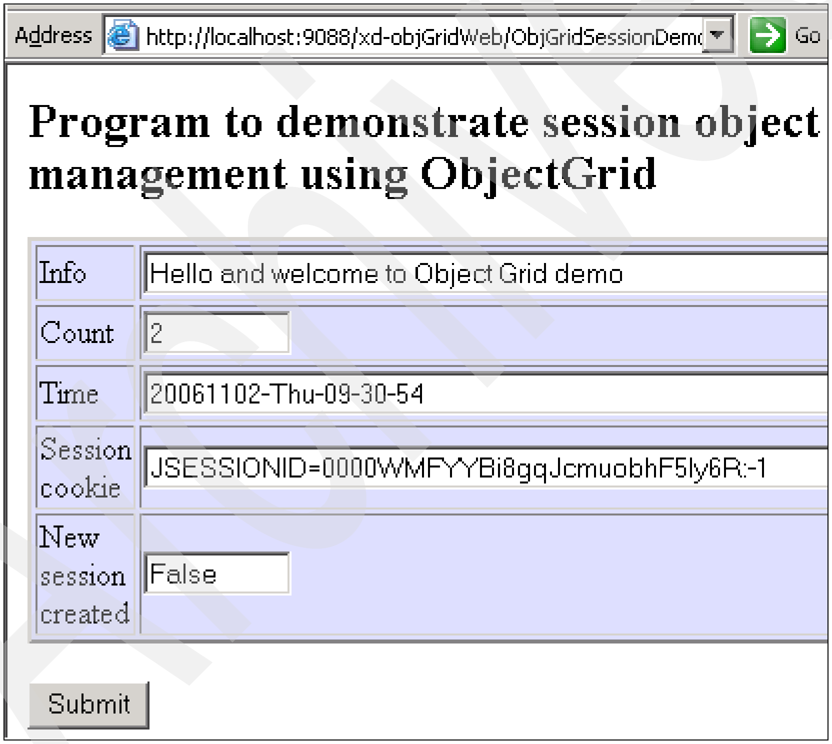


Figure 9-3 ObjectGrid sample application

Table 9-1 on page 310 describes the fields displayed by the application.

Table 9-1 Description of fields used by the servlet

Field	Description
Info	A free text field stored in the session object, any text you enter is saved in the session object.
Count	A counter, stored in the session object, increases by one each time the servlet is run.
Time	The time the servlet ran.
Session Cookie	Session ID.
New session created	If True, a new session object was created when the application ran.
Submit	Runs the servlet when clicked.

Installing the sample application as-is was done just to ensure that it worked cleanly before we started bringing ObjectGrid into the picture.

9.3.2 Set up the ObjectGrid XML configuration files

A working ObjectGrid environment requires the use of a small number of files. Extended Deployment ships sample versions of these files as described in 9.2.1, “Sample files” on page 307.

ObjectGrid cluster file

The ObjectGrid cluster file defines the ObjectGrid cluster configuration. The supplied sample file can be used as shipped for this simple example. Example 9-1 shows a portion of this file.

Example 9-1 Defining a server in ObjectGrid cluster file

```
<cluster name="cluster1" securityEnabled="false">
   <!-- call this a server -->
   <serverDefinition name="server1" host="localhost" 
clientAccessPort="12572"
      peerAccessPort="12573" workingDirectory="/websphere/temp/"
      traceSpec="ObjectGrid=all=enabled"
      systemStreamToFileEnabled="true" />
</cluster>
```

The sample file defines a cluster called **cluster1** that contains a server called **server1** that clients access on port **12572**. The server name of **server1** is a logical name given to this server in the ObjectGrid. This name can be changed to any value you prefer. Since the ObjectGrid client and server will run in the same JVM, we can leave the value of **host** as **localhost**, since the client will use this value when it tries to connect to the ObjectGrid cluster on port 12572.

We copied it from its default location to a new directory on the machine, which was:

```
C:\ogSmSample\objectgrid.cluster.xml
```

ObjectGrid configuration file

The ObjectGrid configuration file describes how your application will use the ObjectGrid. When using WebSphere Extended Deployment, it is not the application that makes direct use of the ObjectGrid, but rather the HTTP Session Manager. The sample file supplied with Extended Deployment contains settings designed to work with that component and can be used, as is. You would not change any values in the **backingMap** elements.

Example 9-2 shows a portion of this XML file.

Example 9-2 Portion of the ObjectGrid configuration sample

```
      <objectGrid name="session">
         <backingMap name="objectgrid.session.metadata" 
pluginCollectionRef="objectgrid.session.metadata" readOnly="false" 
lockStrategy="PESSIMISTIC" ttlEvictorType="LAST_ACCESS_TIME"/>
         <backingMap name="objectgrid.session.attribute" 
readOnly="false" lockStrategy="OPTIMISTIC" ttlEvictorType="NONE" 
copyMode="NO_COPY"/>
         <backingMap name="objectgrid.session.webapp.election" 
readOnly="false" lockStrategy="PESSIMISTIC" 
ttlEvictorType="LAST_ACCESS_TIME"/>
         <backingMap name="datagrid.session.global.ids" 
readOnly="false" lockStrategy="PESSIMISTIC" ttlEvictorType="NONE"/>
      </objectGrid>
```

We copied it from its default location to our directory:

```
C:\ogSmSample\objectgrid.maps.xml
```

splicer.properties

The supplied sample splicer properties file was also copied to our directory:

```
C:\ogSmSample\splicer.properties
```

This file contains a property called objectGridClusterConfigFileName that specifies the location of the ObjectGrid cluster file. Since we moved the sample file to a new location, we modified the properties file accordingly by changing the following line:

```
objectGridClusterConfigFileName = C:\\Program 
Files\\IBM\\WebSphere\\AppServer\\optionalLibraries\\ObjectGrid\\sessio
n\\samples\\objectgrid.cluster.xml
```

to

```
objectGridClusterConfigFileName = 
C:\\ogSmSample\\objectgrid.cluster.xml
```

9.3.3 Enable application for ObjectGrid

Having verified that our sample application runs correctly and prepared the ObjectGrid XML files, we now want to have the session object created by the application saved to the ObjectGrid.

To have the session objects for an application saved to the ObjectGrid requires the application EAR or WAR file to be modified. A Web application is packaged in a WAR file, which contains a file called web.xml. The web.xml file contains information about the application such as URL to servlet mappings, resources, and security.

The web.xml file must be modified so that session objects created by the application are stored in the ObjectGrid.

There are three ways to modify web.xml:

- Use the addObjectgridFilter tool.
- Use an ant task.
- Manually edit the file.

We used the supplied addObjectgridSessionFilter tool (commonly called the splicer) to modify the web.xml file. We do not recommend the approach of manually editing the XML file, as it is more likely that an error will occur.

Run addObjectgridSessionFilter

The addObjectgridSessionFilter tool takes the application EAR file and a properties file as input. We copied the application EAR file to the following:

```
C:\ogSmSample\xd-objGrid.ear
```

We also made a backup of it before running the splicer:

```
C:\ogSmSample\original-xd-objGrid.ear
```

Example 9-3 shows the command we issued to run the splicer and the output produced.

Example 9-3 Adding ObjectGrid properties to the web.xml file

```
C:\ogSmSample>C:\WebSphere\AppServer61\optionalLibraries\ObjectGrid\ses
sion\bin\addObjectgridFilter xd-objGrid.ear splicer.properties

CWWSM0023I: Reading properties file: splicer.properties
CWWSM0021I: Reading archive: xd-objGrid.ear

CWWSM0027I: Processing .war file: xd-objGridWeb
CWWSM0028I: Context parameters are:
CWWSM0029I: Context name: shareSessionsAcrossWebApps Value: false
CWWSM0029I: Context name: sessionIDLength Value: 23
CWWSM0029I: Context name: sessionTableSize Value: 1000
CWWSM0029I: Context name: replicationInterval Value: 10
CWWSM0029I: Context name: replicationType Value: asynchronous
CWWSM0029I: Context name: defaultSessionTimeout Value: 30
CWWSM0029I: Context name: affinityManager Value:
com.ibm.ws.httpsession.NoAffinityManager
CWWSM0029I: Context name: objectGridClusterConfigFileName Value:
C:\ogSmSample\objectgrid.cluster.xml
CWWSM0029I: Context name: objectGridName Value: session
CWWSM0029I: Context name: persistenceMechanism Value: ObjectGridStore

CWWSM0030I: Application splicing completed successfully.
```

Example 9-4 shows the additional XML added to the web.xml file as a result of running the splicer.

Example 9-4 web.xml after splicer has completed

```
<context-param>
        <param-name>shareSessionsAcrossWebApps</param-name>
```

```
        <param-value>false</param-value>
    </context-param>
    <context-param>
        <param-name>sessionIDLength</param-name>
        <param-value>23</param-value>
    </context-param>
    <context-param>
        <param-name>sessionTableSize</param-name>
        <param-value>1000</param-value>
    </context-param>
    <context-param>
        <param-name>replicationInterval</param-name>
        <param-value>10</param-value>
    </context-param>
    <context-param>
        <param-name>replicationType</param-name>
        <param-value>asynchronous</param-value>
    </context-param>
    <context-param>
        <param-name>defaultSessionTimeout</param-name>
        <param-value>30</param-value>
    </context-param>
    <context-param>
        <param-name>affinityManager</param-name>

<param-value>com.ibm.ws.httpsession.NoAffinityManager</param-value>
    </context-param>
    <context-param>
        <param-name>objectGridClusterConfigFileName</param-name>
        <param-value>C:\ogSmSample\objectgrid.cluster.xml</param-value>
    </context-param>
    <context-param>
        <param-name>objectGridName</param-name>
        <param-value>session</param-value>
    </context-param>
    <context-param>
        <param-name>persistenceMechanism</param-name>
        <param-value>ObjectGridStore</param-value>
    </context-param>
    <filter>
        <description>Filter that provides for an ObjectGrid based 
Session Manager.</description>
        <display-name>HttpSessionFilter</display-name>
        <filter-name>HttpSessionFilter</filter-name>
```

```
<filter-class>com.ibm.ws.httpsession.HttpSessionFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>HttpSessionFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

What the splicer did is to add every property from splicer.properties file to the web.xml file. Note that the supplied sample has all possible properties specified, most with the default values. If you are prepared to use the default values then you could remove these from the splicer.properties file.

The key property that must be present is **`objectGridClusterConfigFileName`.** When the ObjectGrid client code is called by the HTTP Session Manager, it looks for this property in the web.xml file to locate the file name.

9.3.4 Deploy the ObjectGrid enabled application

We then deployed the updated application EAR file to the same application server, calling the application ogDemo2. We also changed the context root of the application to ogDemo2 by selecting **Enterprise Applications** → **ogDemo2** → **Context Root For Web Modules** in the administrative console and typing in ogDemo2 as the new context root, and then saving the change.

Because this application accesses no resources, it can be started without having to recycle the server.

9.3.5 Configure the server to be a member of the ObjectGrid

In this example where everything is running in a single JVM, the next step is to configure the application server to have an ObjectGrid server running inside it. Note that ObjectGrid servers can be thought of as logical servers that reside in actual JVMs.

Table 9-2 on page 316 lists the Java properties to be added to the WebSphere server definition. The settings were added by selecting **Application servers** → **area302-a** → **Process Definition** → **Java Virtual Machine** in the administrative console and entering the values in the field labeled **Generic JVM arguments**. A space serves as the delimiter between arguments.

Table 9-2 Java process properties for ObjectGrid

Java property	Description
-Dobjectgrid.server.name=server1	Tells the ObjectGrid which ObjectGrid server (server name) logically exists in this JVM.
-Dobjectgrid.xml.url=file:///c:\\ogSmSample\\objectgrid.maps.xml	Location of the ObjectGrid configuration file.
-Dobjectgrid.cluster.xml.url=file:///c:\\ogSmSample\\objectgrid.cluster.xml	Location of the ObjectGrid cluster file.

Tip: Instead of adding generic arguments, you could instead add Custom Properties on this same panel by selecting **Custom Properties → New**. The property name is the Java property name and the value you want to set it to.

We then recycled the server. Example 9-5 shows the messages written to the SystemOut.log showing the startup of the ObjectGrid server.

Example 9-5 ObjectGrid startup messages in the WebSphere SystemOut.log

```
[11/2/06 15:03:20:687 EST] 0000000a ServerRuntime I   CWOBJ1720I:
HAManager Controller detected that ObjectGrid server is in the
WebSphere environment, using WebSphere HAManager instead of
initializing and starting standalone HAManager.
[11/2/06 15:03:20:765 EST] 0000000a Launcher      I   CWOBJ2501I:
Launching ObjectGrid server server1.
[11/2/06 15:03:20:781 EST] 0000000a Launcher      I   CWOBJ2502I:
Starting ObjectGrid server using ObjectGrid XML file URL
"file:///c:\\ogSmSample\\objectgrid.maps.xml" and Cluster XML file URL
"file:///c:\\ogSmSample\\objectgrid.cluster.xml".
[11/2/06 15:03:22:187 EST] 0000000a Launcher      I   CWOBJ2514I:
Waiting for ObjectGrid server activation to complete.
[11/2/06 15:03:22:359 EST] 0000000a ObjectGridImp I   CWOBJ1308I:
Security of the ObjectGrid instance session is disabled.
[11/2/06 15:03:22:531 EST] 0000000a ServerRuntime I   CWOBJ1118I:
ObjectGrid Server Initializing [Cluster: cluster1 Server: server1].
[11/2/06 15:03:22:546 EST] 0000000a ServerRuntime W   CWOBJ9001W: This
message is an English-only Warning message: LOCALHOST is used in the
configuration that may lose server identity in multiple machine
environment.
```

```
[11/2/06 15:03:22:546 EST] 0000000a ServerRuntime I   CWOBJ9000I: This
message is an English-only Informational message: Client Thread Pool
created and size established. Minimum size: 5 Maximum size: 50.
[11/2/06 15:03:22:593 EST] 0000000a TCPChannel    I   TCPC0001I: TCP
Channel TCP is listening on host *  (IPv4) port 12572.
[11/2/06 15:03:22:593 EST] 0000000a WSChannelFram A   CHFW0019I: The
Transport Channel Service has started chain ObjectGridServiceChain.
[11/2/06 15:03:22:656 EST] 0000000a ServerRuntime I   CWOBJ1511I:
Replication Group Member {0} (Server[server1] ObjectGrid[session]
Mapset[myset] ReplicationGroup[r1] Partition[0]) (Primary only) is open
for business.
[11/2/06 15:03:22:718 EST] 0000000a ServerRuntime I   CWOBJ1001I:
ObjectGrid Server server1 is ready to process requests.
[11/2/06 15:03:22:734 EST] 0000000a ObjectGridCom I   CWOBJ0900I: The
ObjectGrid runtime component is started for server server1.
```

To verify the ObjectGrid server was listening on the port that we specified, we issued this a netstat -an command and saw the following results:

```
TCP    0.0.0.0:12572          0.0.0.0:0              LISTENING
```

9.3.6 Run the ObjectGrid enabled application

We then accessed our new ObjectGrid enabled application with the following URL:

```
http://itsonode1:9088/ogDemo2/ObjGridSessionDemo
```

Session trace

We then wanted to verify that the session object for the application was indeed being written to the ObjectGrid server. To do this, we added these trace settings to the server runtime configuration:

```
session=all: session.http=all
```

We then re-ran the application and looked in the trace.log.

Example 9-6 shows a small extract of trace from running one request.

Example 9-6 Trace of XD HTTP Session manager facade

```
HttpSessionRequestWrapper.constructor.
****************************************************

HttpSessionRequestWrapper.constructor. req.getContextPath()=/ogDemo2
HttpSessionRequestWrapper.constructor. req.getMethod()=POST
```

```
HttpSessionRequestWrapper.constructor. req.getPathInfo()=null
HttpSessionRequestWrapper.constructor. req.getPathTranslated()=null
HttpSessionRequestWrapper.constructor. req.getProtocol()=HTTP/1.1
HttpSessionRequestWrapper.constructor. req.getQueryString()=null
HttpSessionRequestWrapper.constructor. req.getRemoteAddr()=127.0.0.1
HttpSessionRequestWrapper.constructor. req.getRemoteHost()=127.0.0.1
HttpSessionRequestWrapper.constructor.
req.getRequestedSessionId()=zgFI6z-CziGqg4tp5zU5eh-
HttpSessionRequestWrapper.constructor.
req.getRequestURI()=/ogDemo2/ObjGridSessionDemo
HttpSessionRequestWrapper.constructor. req.getServerName()=localhost
HttpSessionRequestWrapper.constructor. req.getServerPort()=9088
HttpSessionRequestWrapper.constructor.
req.getServletPath()=/ObjGridSessionDemo
HttpSessionRequestWrapper.constructor.
****************************************************
```

As can be seen in the trace, the HttpSessionRequestWrapper class is in use, which is part of the HTTP Session Manager facade that interacts with the ObjectGrid.

We then ran the original application which was not modified to use the ObjectGrid, in the same server, and no trace messages were produced, indicating it was not interacting with the ObjectGrid.

9.3.7 ObjectGrid and PMI

When ObjectGrid is present in a WebSphere application server, Performance Monitoring Infrastructure (PMI) information is available. You can use the Tivoli Performance Viewer in the administrative console to view this information.

Note: We found that initially the ObjectGrid PMI data was not viewable via the Tivoli Performance Viewer. This was because we were using WebSphere Application Server V6.1, which did not support this correctly. Pending a permanent fix for APAR # PK35551, we were able to obtain the ObjectGrid PMI data by manually editing the pmi-config.xml file.

1. Edit the following file on the deployment manager for the application server with the ObjectGrid server:

   C:\WebSphere\AppServer61\profiles\AppSrv01\config\cells\ITSOCell01\nodes\ITSOnode1\servers\area302-a\pmi-config.xml

   Change the following lines:

```
<pmimodules xmi:id="PMIModule_1161630711578"
moduleName="objectGridModule"
type="com.ibm.ws.objectgrid.pmi.objectGridModule" enable=""/>
  <pmimodules xmi:id="PMIModule_1161630711593"
moduleName="mapModule" type="com.ibm.ws.objectgrid.pmi.mapModule"
enable=""/>
```

   to

```
<pmimodules xmi:id="PMIModule_1161630711578"
moduleName="objectGridModule"
type="com.ibm.ws.objectgrid.pmi.objectGridModule" enable="*"/>
  <pmimodules xmi:id="PMIModule_1161630711593"
moduleName="mapModule" type="com.ibm.ws.objectgrid.pmi.mapModule"
enable="*"/>
```

   The change is the addition of an * towards the end of each line.

2. Then using the WebSphere Administration GUI, set the PMI level on the server to Custom.

   Monitoring and Tuning → **Performance Monitoring Infrastructure (PMI)** → *<server>*

3. Save the change, and restart the server.

1. Select **Monitoring and Tuning** → **Performance Viewer** → **Current Activity**.
2. Check the box to the left of the server, and click **Start Monitoring**.
3. Click the server name to see the data.

In the Tivoli Performance Viewer, there are two entries under Performance Modules related to ObjectGrid:

- **objectGridModule**

  The objectGridModule data shows the average transaction time.

- **mapModule**

  The mapModule shows you information about the maps in the ObjectGrid server. In our case we wanted to find data that would show us how many objects there were stored in the server. We found that one of the values shown under **mapModule** → **Session** → **datagrid.session.globals.ids** was a count of the objects in the server. This value increased by one every time we ran the application and it created a new session object.

  Note that due to the bug we encountered (see Note box on page 319), the values did not have descriptions associated with them.

Figure 9-4 shows a portion of the Tivoli Performance Viewer display, showing the datagrid.session.global.ids value.

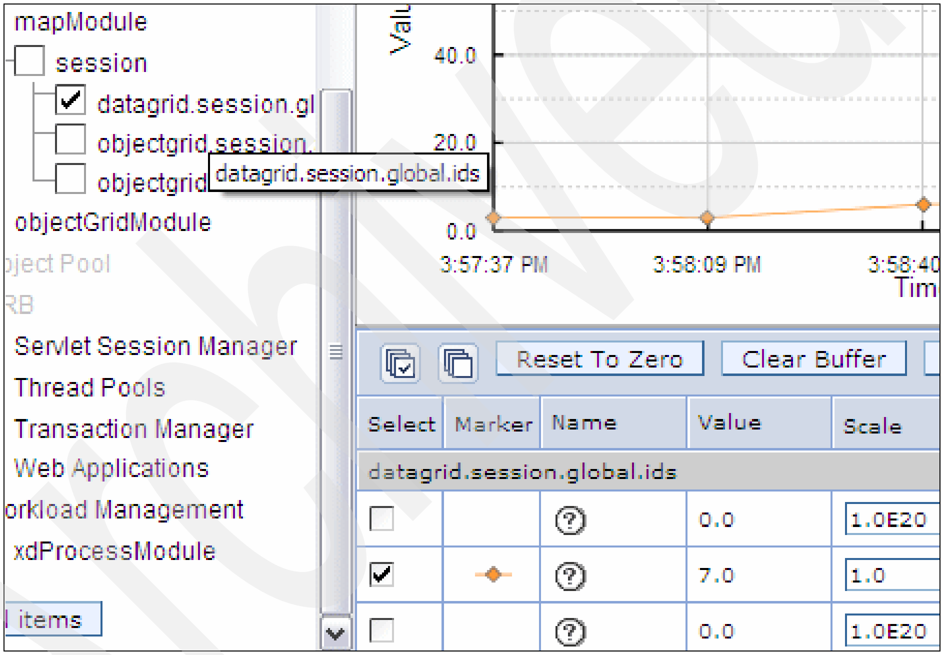


Figure 9-4 Displaying ObjectGrid performance indicators in the Tivoli Performance Viewer

9.4 XML file inter-relationship

We provided a simple ObjectGrid setup in section 9.3, “Example - Single server” on page 308. In this section we outline the inter-relationship between the XML files involved.

9.4.1 The XMLFiles

There are three XML files involved in the simple example:

- The web.xml file in the application ear file
- The ObjectGrid configuration file, objectgrid.maps.xml
- The ObjectGrid cluster file, objectgrid.cluster.xml

The diagram in Figure 9-5 depicts the inter-relationship that exists in an ObjectGrid environment used for session management.

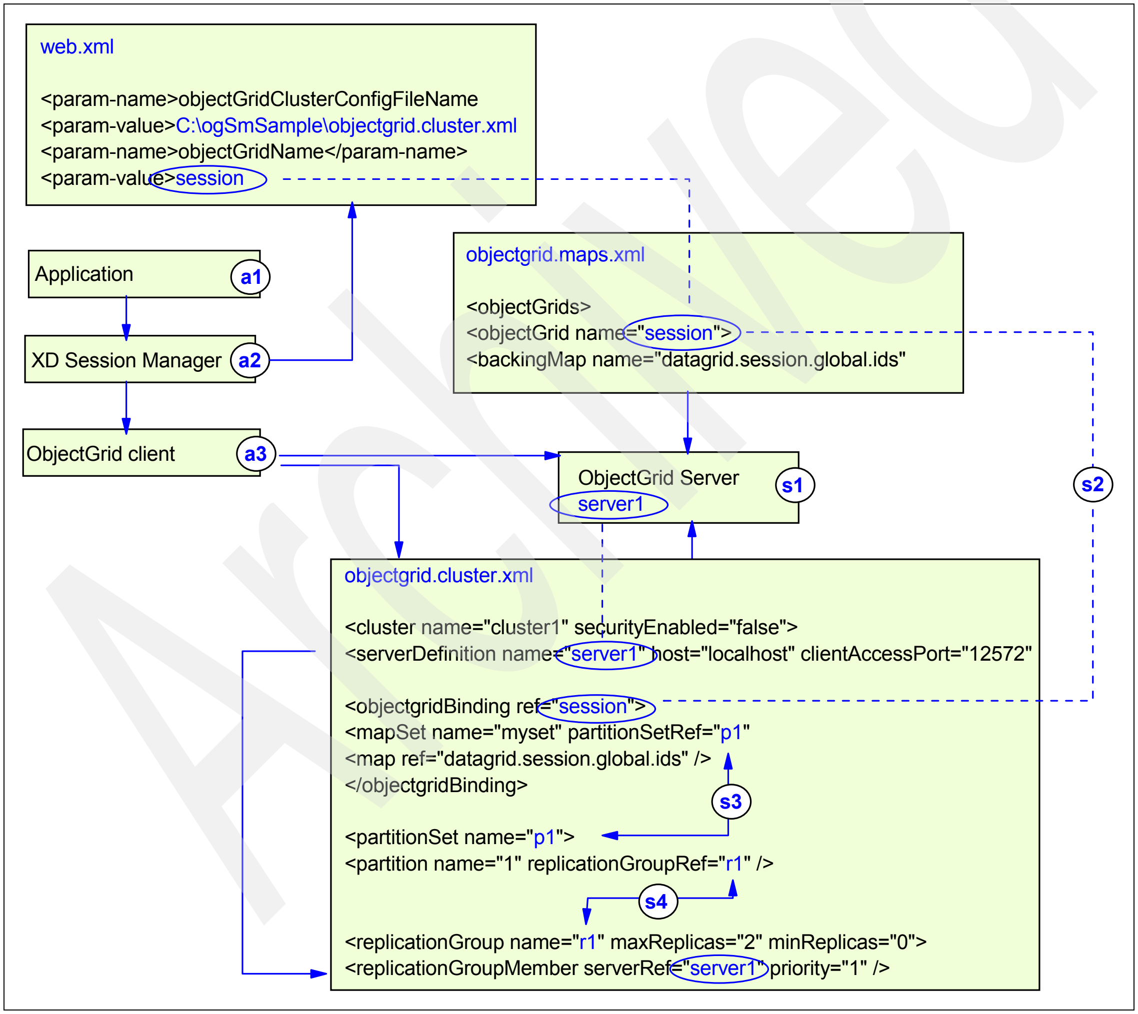


Figure 9-5 ObjectGrid - Session Object - inter-relationships

Description of server side relationship

S1 When started, the ObjectGrid server is passed the name of the logical server it is to be in the grid, in this case, server1. It is also passed the location of the ObjectGrid configuration file and ObjectGrid cluster file. The ObjectGrid cluster file defines the port the server is to listen on.

S2 The ObjectGrid configuration file has an objectGrid element called session. This relates to the objectgridBinding element with the same name in the ObjectGrid cluster file.

S3 In the ObjectGrid cluster file, the objectgridBinding element contains a mapSet element that identifies the partition map that will back the map set, in this case, p1.

S4 The partitionSet element identifies the primary server that is used to store the partition, in this case, server1.

Description of client side relationship

A1 The application is invoked, and creates or accesses a session object.

A2 This request to access the session object is intercepted by the session manager facade. It reads the web.xml file of the application EAR file to see if it contains entries indicating that the session manager facade should intercept the session object calls. If it does not, the call will pass through to the normal WebSphere session classes. If it does, then the Extended Deployment session manager extracts the ObjectGrid cluster file name and the objectGridName and passes these to the ObjectGrid client code.

a3 The ObjectGrid client code reads the ObjectGrid cluster file and uses the objectGridName value to work out to which ObjectGrid server it should connect to. It then connects to that server and performs the appropriate session object action.

9.5 Example: sharing session objects among applications

In this section we show how to share a session object between more than one application.

9.5.1 Why share?

As mentioned in “Applications and session objects” on page 305 you cannot share the session object between different applications in WebSphere Application Server Network Deployment. Although the default behavior of ObjectGrid is to enforce the same restriction, you can use a parameter to allow a session object to be shared between applications. This applies not only for applications in the same application server, but applications in the same cell, applications in different cells, and even applications running in different products.

Why would you want to share a user’s session object across multiple applications? If you have customers using your Web site accessing multiple applications, allowing the session objects to be shared can create a better user experience.

9.5.2 Sharing considerations

The mechanism used by Extended Deployment to provide this capability relies on the use of the JSESSIONID cookie. When an user starts using an application, a cookie called JSESSIONID is returned to the end users browser. Each subsequent request from the user includes this cookie. The application uses this cookie to retrieve the session object it is using to manage the end users session.

Cookies are tied to domain names. For example, if a user receives a cookie from www.test.co, if they then access a site at www.sample.co, the cookie received from www.test.co is not sent to www.sample.co.

Since Extended Deployment relies on the JSESSIONID cookie to manage the session object keep in mind that all the applications will have to be accessed via the same domain ID to allow the session object to be shared.

The way the end users use their browser is also important. Take, for example, an user who is accessing application A and application B from a Microsoft Internet browser. The user clicks the Microsoft Internet browser icon to open a new browser window and accesses application A, creating a new session object. The user now wants to access application B. If the user opens a new window by clicking the Microsoft Internet browser icon to access application B, a new session object is created. This is because the two browser windows are not sharing session objects that they create. The second browser window does not know there is a cookie in the first browser window since the JSESSIONID cookie is stored in memory only. If you plan to share session objects across applications, then you need to train the end users not to use this approach. If they want to access application B at the same time that they are accessing application A, then in the current browser window they should select **File** → **New Window** to start a second browser window. We recommend that you thoroughly test this behavior in whatever version of browser you use.

9.5.3 Topology

For this example we use the same topology we used in 9.3, “Example - Single server” on page 308.

9.5.4 Modify web.xml

To enable session object sharing across applications, modify the web.xml file for each application. You must change the value of the shareSessionsAcrossWebApps attribute to `true` (the default is `false`) and redeploy the applications.

```
<context-param>
        <param-name>shareSessionsAcrossWebApps</param-name>
        <param-value>true</param-value>
    </context-param>
```

The recommendation is to use the splicer tool (see “Run addObjectgridSessionFilter” on page 313); however, since it is a minor editing change, you could also make the modification using an application development tool.

9.5.5 Testing

We first modified the web.xml file fro ogDemo2 and re-installed it to the application. We then installed a copy of ogDemo2 on the area302-a server, calling it ogDemo4 and setting the context root to ogDemo4. We stopped and started the server.

To test the session object sharing, we first accessed ogDemo2 using the following URL:

`http://itsonode1:9088/ogDemo2/ObjGridSessionDemo`

We clicked the submit button a few times, obtaining the display shown in Figure 9-6 on page 325.

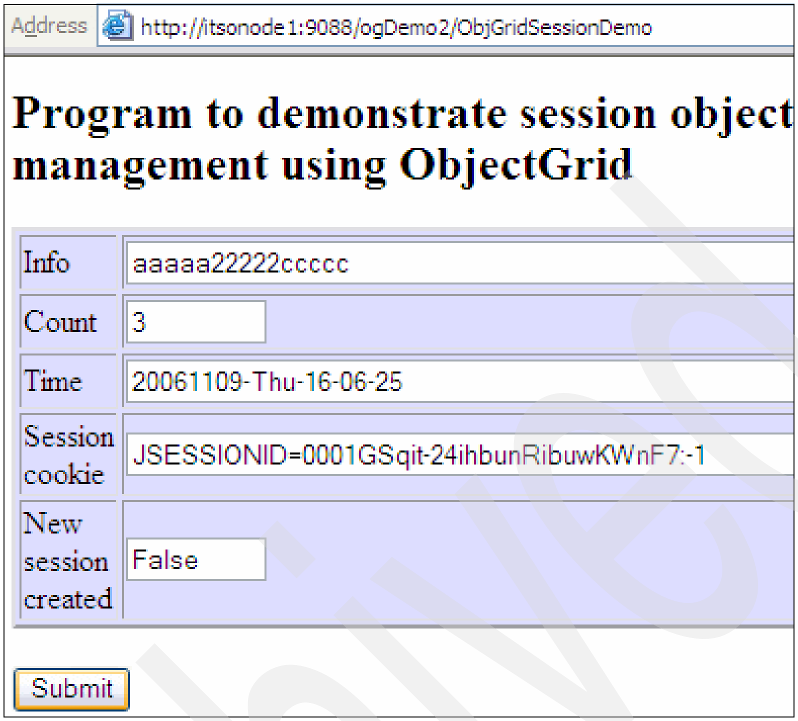


Figure 9-6 Accessing the first application

We then selected **File** → **New Window** and accessed ogDemo4 using the following URL:

```
http://itsonode1:9088/ogDemo4/ObjGridSessionDemo
```

This produced the display shown in Figure 9-7 on page 326.

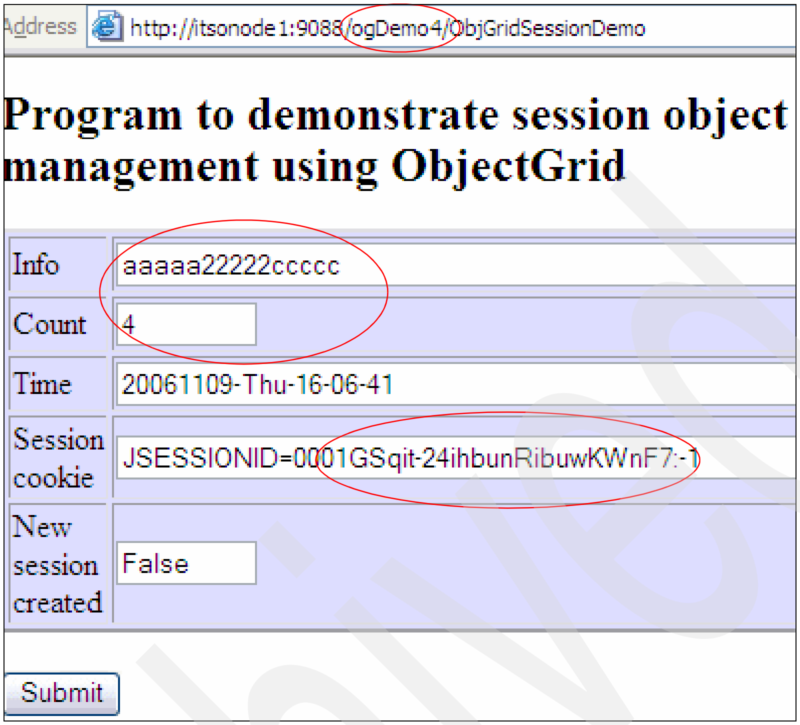


Figure 9-7 After invoking the second copy of the same application

Note that the value for New session created is False, indicating that when the second copy of our sample application was run, it used the existing session object created by running the first copy of our sample application. Also the value for count increased by 1 and not reset to 1, as would have been the case if the session object was not shared.

When we continued to run both applications by alternating between the two windows, the main part of cookie value did not change, indicating the same session object was being used. The four characters at the start of the ID are used by WebSphere for management of session objects and may change.

9.6 Example: ObjectGrid on separate server

We now build on the example setup in 9.3, “Example - Single server” on page 308 by moving the ObjectGrid server into its own JVM on another machine.

The topology we plan to set up is shown in Figure 9-8 on page 327.

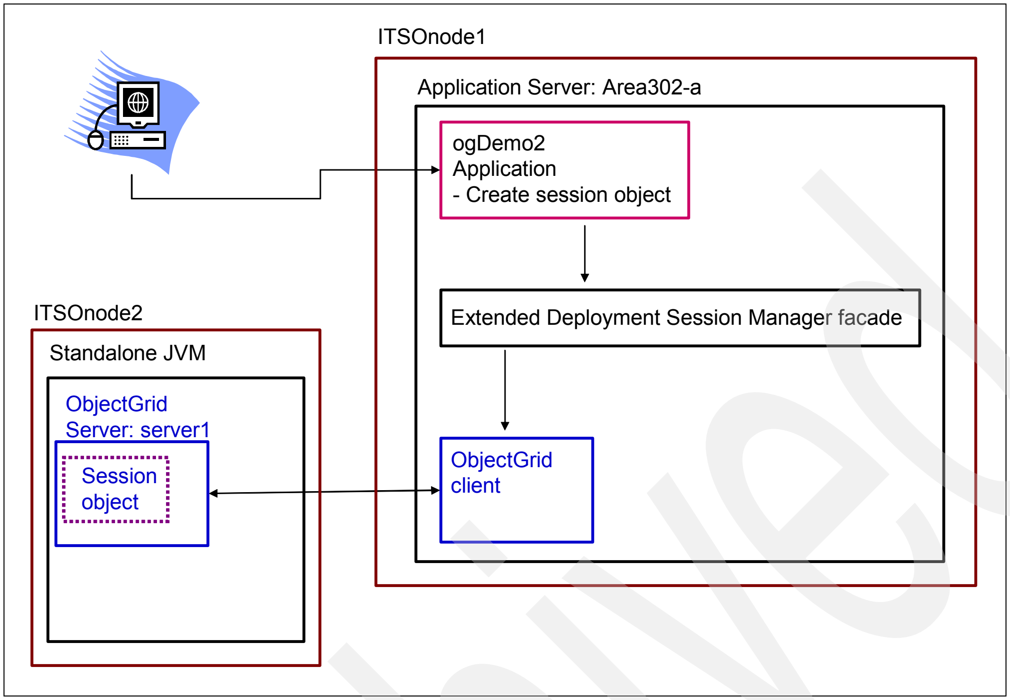


Figure 9-8 ObjectGrid with two servers

9.6.1 Set up a new ObjectGrid JVM

To demonstrate the flexibility of ObjectGrid, rather then setting up the ObjectGrid server in a new server in the existing WebSphere cell, we set up the ObjectGrid server in a standalone JVM.

To do this all that was required was to start a Java process, passing in the ObjectGrid related properties, and making available the ObjectGrid product companionate.

Installation

When running the ObjectGrid outside of a WebSphere cell, you need to install the ObjectGrid feature that is shipped with the WebSphere Extended Deployment Mixed Server Environment. You need version 6.0.1 or better of this component as the ObjectGrid feature is not present in version 6.0.0.

The install process is straight forward. Be sure to install any applicable fixes (see the Note box on page 306).

We installed the feature to a directory called c:\zProducts\MixedServer on the second node of our existing WebSphere cell called ITSOnode2.

ObjectGrid XML files

The ObjectGrid XML files that we used in the single-server example (see section 9.3.2, “Set up the ObjectGrid XML configuration files” on page 310), were copied to the same location on ITSOnode2.

Start the ObjectGrid server

Example 9-7 shows the command we issued to start the ObjectGrid server and the resulting output.

Example 9-7 Starting ObjectGrid server in standalone JVM

```
C:\>set JAVA_HOME=C:\IBMJava142

C:\>C:\zProducts\MixedServer\ObjectGrid\bin\startOgServer.bat server1
-objectgridFile C:\ogSmSample\objectgrid.maps.xml -clusterFile
C:\ogSmSample\objectgrid.cluster.xml
************ Start Display Current Environment ************
Host Operating System is Windows 2003, version 5.2
Java version = J2RE 1.4.2 IBM Windows 32 build cn142-20050609 (JIT
enabled: jitc), Java Compiler = jitc, Java VM name = Classic VM
was.install.root = null
user.install.root = null
Java Home = C:\IBMJava142\jre
ws.ext.dirs = null
Classpath =
C:\zProducts\MixedServer\ObjectGrid\lib\objectgrid.jar;C:\zProducts\
MixedServer\ObjectGrid\session\lib\sessionobjectgrid.jar;C:\zProducts\M
ixedServer\ObjectGrid\lib\asm.jar;C:\zProducts\MixedServer\ObjectGrid\l
ib\cglib.jar
Java Library path =
C:\IBMJava142\bin;.;C:\WINDOWS\system32;C:\WINDOWS;C:\Progra
m Files\Windows
ResourceKits\Tools\;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\
Wbem;C:\Program Files\ObjREXX;C:\Program
Files\ObjREXX\OODIALOG;C:\Tools;C:\IBMJava142\bin
Current trace specification = *=all=disabled
************* End Display Current Environment *************
[11/3/06 16:52:08:062 EST] 26f0003d ManagerAdmin  I TRAS0017I: The
startup trace state is *=all=disabled.
[11/3/06 16:52:09:094 EST] 26f4003e Launcher      I CWOBJ2501I:
Launching ObjectGrid server server1.
```

```
[11/3/06 16:52:09:141 EST] 26f4003e Launcher      I CWOBJ2502I:
Starting ObjectGrid server using ObjectGrid XML file URL
"C:\ogSmSample\objectgrid.maps.xml" and
 Cluster XML file URL "C:\ogSmSample\objectgrid.cluster.xml".
[11/3/06 16:52:10:766 EST] 26f4003e Launcher      I CWOBJ2514I: Waiting
for ObjectGrid server activation to complete.
[11/3/06 16:52:11:172 EST] 26f4003e ManagerAdmin  I TRAS0017I: The
startup trace state is ObjectGrid=all=enabled.
[11/3/06 16:52:11:203 EST] 26f4003e ObjectGridImp I CWOBJ9000I: This
message is an English-only Informational message: WebSphere Application
Server PMI is not found.
[11/3/06 16:52:11:234 EST] 26f4003e BaseMap       I CWOBJ9000I: This
message is an English-only Informational message: WebSphere Application
Server PMI is not found.
[11/3/06 16:52:11:438 EST] 26f4003e ObjectGridImp I CWOBJ1308I:
Security of the ObjectGrid instance session is disabled.
[11/3/06 16:52:11:484 EST] 26f4003e ServerRuntime I CWOBJ1118I:
ObjectGrid Server Initializing [Cluster: cluster1 Server: server1].
[11/3/06 16:52:11:500 EST] 26f4003e ServerRuntime W CWOBJ9001W: This
message is an English-only Warning message: LOCALHOST is used in the
configuration that may lose server identity in multiple machine
environment.
[11/3/06 16:52:11:594 EST] 26f4003e ServerRuntime I CWOBJ9000I: This
message is an English-only Informational message: Client Thread Pool
size established. Minimum size: 5 Maximum size: 50.
[11/3/06 16:52:12:875 EST] 26f4003e ServerRuntime I CWOBJ1511I:
Replication Group Member {0} (Server[server1] ObjectGrid[session]
Mapset[myset] ReplicationGroup
[r1] Partition[0]) (Primary only) is open for business.
[11/3/06 16:52:12:906 EST] 26f4003e ServerRuntime I CWOBJ1001I:
ObjectGrid Server server1 is ready to process requests.
```

Note that when you want to stop this server, you use the following command:

```
C:\>c:\zProducts\MixedServer\ObjectGrid\bin\stopOgServer.bat server1
-bootstrap localhost:12572
```

ObjectGrid trace and log files

In the ObjectGrid cluster files we currently have the following settings:

```
workingDirectory="/websphere/temp/"
traceSpec="ObjectGrid=all=enabled"
```

The value specified for workingDirectory was treated as an absolute directory. When the server was started, directories and logs where created as shown in Figure 9-9.

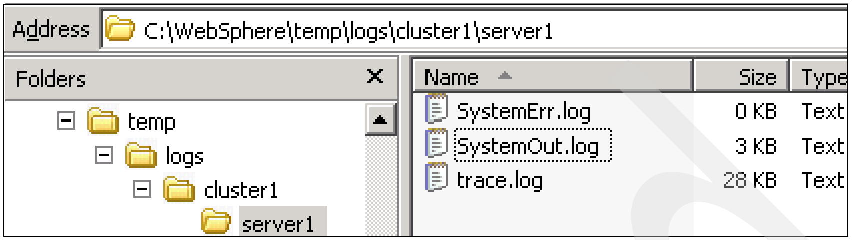


Figure 9-9 ObjectGrid server logs

9.6.2 Set up the ObjectGrid client

Two changes need to be made to our existing environment to use the new ObjectGrid server.

Update the ObjectGrid cluster file

The sample application is running on the Area302-a application server on node ITSOnode1. We now need to modify the ObjectGrid cluster file to reference the ObjectGrid server now running on ITSOnode2.

In the file `C:\ogSmSample\objectgrid.cluster.xml` `we changed this line:`

```
   <serverDefinition name="server1" host="localhost"
clientAccessPort="12572"
```

to

```
<serverDefinition name="server1" host="ITSOnode2"
clientAccessPort="12572"
```

Remove the server JVM arguments

As we will not be running the ObjectGrid server in the Area302-a server, we removed the JVM ObjectGrid arguments we added in “Remove the server JVM arguments” on page 330 then saved that change.

Recycle the server

We then stopped and started the Area302-a server to pick up the changes to the ObjectGrid cluster file.

9.6.3 Run the application

We then ran the ogDemo2 application, which created a session object. When the object is created the XD HTTP Session Manager calls the ObjectGrid client code, which in turn connects to the ObjectGrid server running in the JVM on ITSOnode2.

Example 9-8 is an extract from SystemOut.log.

Example 9-8 Messages showing ObjectGrid client starting

```
[11/6/06 10:24:27:531 EST] 000000a2 ServletWrappe I   SRVE0242I:
[ogDemo2] [/ogDemo2] [ObjGridSessionDemo]: Initialization successful.
[11/6/06 10:24:28:546 EST] 000000a2 ConfigNetwork I   CWOBJ1903I:
Configuration network service is initialized.
[11/6/06 10:24:28:578 EST] 000000a6 ConfigNetwork I   CWOBJ1904I:
Configuration network service is started.
[11/6/06 10:24:28:609 EST] 000000a6 ConfigNetwork I   CWOBJ1905I:
Configuration handler is started.
[11/6/06 10:24:29:625 EST] 000000a2 ObjectGridImp I   CWOBJ1308I:
Security of the ObjectGrid instance session is disabled.
[11/6/06 10:24:29:718 EST] 000000a2 ClientRuntime I   CWOBJ1118I:
ObjectGrid Server Initializing [Cluster: cluster1 Server: 0].
[11/6/06 10:24:29:718 EST] 000000a2 ClientNetwork I   CWOBJ1900I:
Client server remote procedure call service is initialized.
[11/6/06 10:24:29:734 EST] 000000a2 SysAdminServi I   CWOBJ1913I:
System administration network service is initialized.
[11/6/06 10:24:29:750 EST] 000000a2 ObjectGridMan I   CWOBJ1120I:
ObjectGrid Client connected successfully to host: ITSOnode2 port:
12572.
[11/6/06 10:24:29:796 EST] 000000ae ClientNetwork I   CWOBJ1901I:
Client server remote procedure call service is started.
[11/6/06 10:24:29:796 EST] 000000ae ClientNetwork I   CWOBJ1902I:
Client server remote procedure call handler threads are started.
[11/6/06 10:24:29:796 EST] 000000b0 SysAdminServi I   CWOBJ1914I:
System administration network service is started.
[11/6/06 10:24:29:812 EST] 000000b0 SysAdminServi I   CWOBJ1915I:
System administration handler is started.
```

We ran the application a few times to verify that the session object stored in the remote ObjectGrid server was being correctly updated.

9.7 Adding a replica

In this section we describe the use of replica servers in an ObjectGrid and show an example of how to set one up.

9.7.1 What is a replica?

The purpose of a replica, as the name suggests, is to provide a server that contains the same data as the primary and can become the primary should the primary server fail or stop.

In the ObjectGrid cluster file, partition sets are defined which are backed by the servers in the replication group element. Any number of servers can be defined in the replication group element, but only one runs as the primary server. Any other servers run as replicas or standby servers. Updates that need to be done to objects in the partition set backed by the replication group are applied to the primary server.

More then one replica can be defined. For example, you may want to define more then one replica if you have an application that makes a large number of read-only calls for objects, as these calls can be sent to any server in the replication group.

9.7.2 When does the replica get updated from the primary?

When the primary server is updated, this update needs to flow to all replicas. Which leads to the question, when will this updating of the replicas occur? This depends on whether you specified synchronous or asynchronous for the replication setting.

If the object update is being done as a part of a synchronous call from the ObjectGrid client, then control does not return to the caller until the update occurs on the primary server and all replica servers in the replication group.

If the object update is occurring asynchronously from the ObjectGrid client, control returns to the client once the update to the primary server is complete. The updates to the replica occur when the primary server has the capacity to perform this task.

With regard to session objects, the setting that controls when the session objects are written to the ObjectGrid servers is set in the web.xml using the following setting:

```
<context-param>
        <param-name>replicationType</param-name>
```

```
        <param-value>asynchronous</param-value>
</context-param>
```

If set to synchronous, then a servlet waits for the session object to be written to the primary server and any replica servers before completing.

If set to asynchronous, then servlet completes without waiting, and the ObjectGrid client updates the ObjectGrid servers asynchronously per additional settings set in the web.xml that specify the interval between when updates should be done.

9.7.3 Example: Using a replica

Figure 9-10 on page 334 shows how we created a topology that used a primary ObjectGrid server and a replica.

> **Note:** When using replicas the ObjectGrid servers must be of the same type, meaning they must be all WebSphere application servers or all stand-alone servers in the Mixed Server Environment (MSE).

In this configuration, the application remains on Area302-a. The stand-alone JVM used in the previous scenario was stopped and will not be used in this scenario. In its place, a new ObjectGrid server (server1) was set up on ITSOnode2 in the Area302-1P application server. The replica ObjectGrid server (server1-R) resides on ITSOnode4 in the Area302-1R application server.

> **Important:** The primary server and at least one replica server must be active before the ObjectGrid servers consider themselves to be completely initialized.

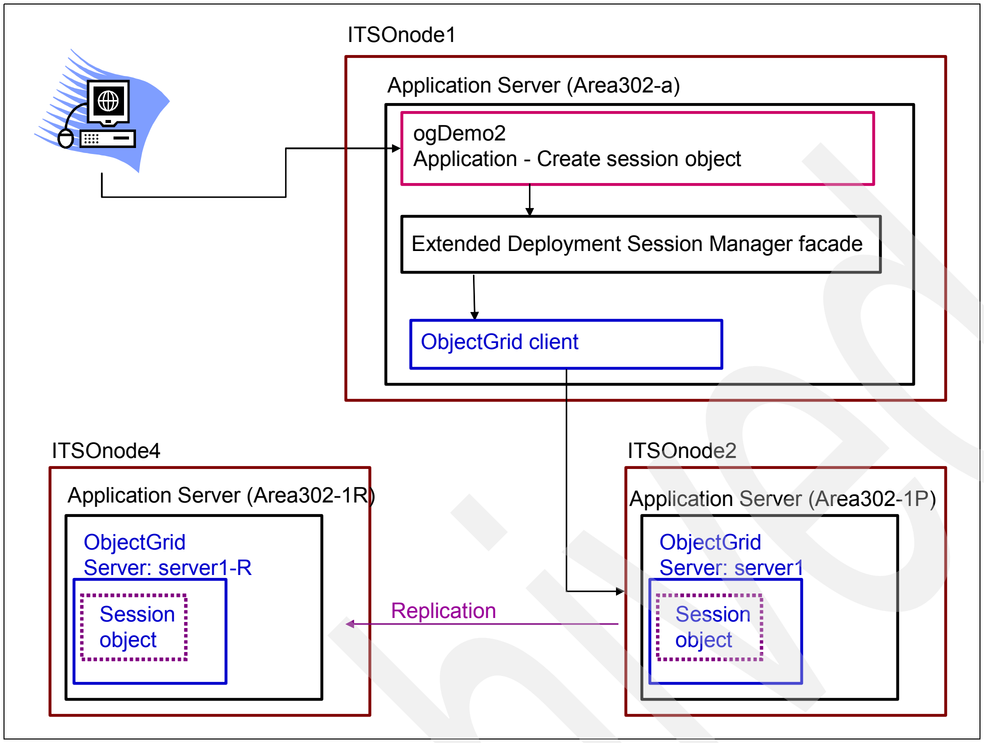


Figure 9-10 Adding a replica server

To configure this environment requires the following steps.

Modify the ObjectGrid cluster file

We defined two ObjectGrid servers in the cluster file, server1 and server1-R. Example 9-9 shows an excerpt of this file showing the relevant definitions.

Example 9-9 ObjectGrid cluster file with a replica server

```
<serverDefinition name="server1" host="ITSOnode2" clientAccessPort="12572"
peerAccessPort="12573" workingDirectory="/websphere/temp/"
traceSpec="ObjectGrid=all=enabled" systemStreamToFileEnabled="true" />

<serverDefinition name="server1-R" host="ITSOnode4" clientAccessPort="12578"
peerAccessPort="12579" workingDirectory="/websphere/temp/"
traceSpec="ObjectGrid=all=enabled" systemStreamToFileEnabled="true" />

<partitionSet name="p1">
<partition name="1" replicationGroupRef="r1" />
```

```
</partitionSet>

<replicationGroup name="r1" maxReplicas="2" minReplicas="0">
<replicationGroupMember serverRef="server1" priority="1" />
<replicationGroupMember serverRef="server1-R" priority="2" />
</replicationGroup>
```

This ObjectGrid cluster file is copied to the appropriate location on each application server in the scenario.

Define the ObjectGrid servers

We defined a new application server on ITSOnode2 called Area302-1P. The JVM arguments were updated with the following:

```
-Dobjectgrid.server.name=server1
-Dobjectgrid.xml.url=file:///c:\\ogSmSample\\objectgrid.maps.xml
-Dobjectgrid.cluster.xml.url=file:///c:\\ogSmSample\\objectgrid.cluster
.xml
```

We then defined a new server on ITSOnode4 called Area302-1R. The JVM arguments were updated with the following:

```
-Dobjectgrid.server.name=server1-R
-Dobjectgrid.xml.url=file:///c:\\ogSmSample\\objectgrid.maps.xml
-Dobjectgrid.cluster.xml.url=file:///c:\\ogSmSample\\objectgrid.cluster
.xml
```

Recycle the servers

The Area302-1P and Area302-1R servers were started and a quick check of the logs verified that the ObjectGrid servers had started and were listening on their respective ports.

We then stopped and started the WebSphere server that runs the application, namely Area302-a that runs on ITSOnode1 to pick up the change to the ObjectGrid cluster file.

Test and verify

We then used a load tool to run our test application so that a large number of session objects were created.

We used the WebSphere Tivoli Performance Viewer as described in 9.3.7, “ObjectGrid and PMI” on page 318 to view how many session objects were created in the Area302-1R server on ITSOnode4.

Part 4

Appendix

A

Appendix - Visualization logs

This appendix contains details of the contents of the visualization logs. All of these log files are produced in comma-separated-value form. The first record of the file is a list of field names.

Following are the visualization logs available in WebSphere Extended Deployment V6.0.2:

- BusinessGridStatsCache
- DeploymentTargetStatsHistoricCache
- NodeStatsHistoricCache
- ServerStatsCache
- TCModuleStatsCache, TCModuleInstanceStatsCache

Two more will be added as part of an interim fix to WebSphere Extended Deployment V6.0.2:

- FineGrainedPowerConsumptionStatsCache
- ServerPowerConsumptionStatsCache

A sample Perl script is included to illustrate how to process logs using scripts:

- Perl script to calculate CPU utilizations

BusinessGridStatsCache

This log provides information about batch work. It provides counts of jobs run, both successfully and unsuccessfully with average execution times. It creates a record for each application / deployment target combination. Although it is populated with data from both online and batch work, it is really only of value for batch work.

BusinessGridStatsCache	
Field name	**Description**
timeStamp	Time in milliseconds since January 1, 1970, 00:00:00 GMT
node	Node name
server	Server name
tcname	Transaction class name
appname	Application name
j2eemodname	J2EE module name
version	Node version
dtname	Deployment target name
scname	Service policy name
nodegroup	Node group name
cell	Cell name
updateTime	Time of update
num_requested	Number of jobs that arrive at the execution environment (endpoint application) for processing
num_completed	Number of jobs that run to completion at the execution environment
exec_time	Average time in milliseconds that jobs spend executing
max_concurrency	Maximum concurrency level that is attained
num_queued	Number of jobs that are queued at the scheduler
num_dispatched	Number of jobs that are dispatched at the scheduler

BusinessGridStatsCache	
Field name	**Description**
num_failed	Number of jobs that failed in the execution environment
num_errors	Number of dispatch errors that occurred for jobs
queue_time	Average time in milliseconds that a job spent in the queue
dispatch_time	Average time in milliseconds that a job spent being dispatched
dispatch_error_time	Average time in milliseconds for jobs spent being dispatched when a dispatch error occurred

DeploymentTargetStatsHistoricCache

This file is of little use for chargeback purposes, but for completeness we have included its record layout.

DeploymentTargetStatsHistoricCache	
Field name	**Description**
timeStamp	Time in milliseconds since January 1, 1970, 00:00:00 GMT
deploymentTargetName	Deployment target name
nodeName	Node name
deploymentTargetType	Deployment target type
speedReq	Speed req
highMemMark	High memory mark

NodeStatsHistoricCache

This log file contains historic information about the node statistics cache. At the specified time interval it creates a record for the deployment manager and every node agent in the cell.

Table 9-3 Contents of the NodeStatsHistoricCache log file

NodeStatsHistoricCache	
Field name	**Description**
timeStamp	Time in milliseconds since January 1, 1970, 00:00:00 GMT
nodeName	Node name
nodeCPU	Percentage of CPU utilization
nodeFreeMemory	Free memory in kilobytes
usedMemory	Memory being used
version	Node version
nodeSpeed	Node speed
background	Background CPU. This non-WebSphere background is due to things other than application servers. WebSphere JMS servers are included in the background.
wasBackground	WebSphere Application Server background CPU utilization on that node. The background includes application server instances that are starting and stopping.
isDBNode	True, if this is a database node.

ServerStatsCache

This log file contains data from the server statistics cache. At the specified time interval it creates a record for every server in the cell including stopped servers and Web servers. (The deployment manager and node agents are reported in the NodeStatsHistoricCacheLog.) It contains the following fields.

Table 9-4 Contents of the ServerStatsCache log file

ServerStatsCache	
Field name	**Description**
timeStamp	Time in milliseconds since January 1, 1970, 00:00:00 GMT
name	Server name
node	Name of the node the server is running on
version	Node version, for example XD 6.0.2.0
weight	dWLM weight of server
cpu	Percentage of CPU utilization
usedMemory	Used memory in kilobytes
uptime	Server up time
totalRequests	Total requests for server
updateTime	Time of statistics
highMemMark	The high memory mark comes from the Memory Profiler that runs in the cell and provides input to the application placement controller. The Memory Profiler has a five minute interval: over that interval - it evaluates the totalMemory stat for the server. The HighMemoryMark is the largest value it observes over the interval.
residentMemory	Resident memory
totalMemory	The total memory for a server is its resident + swapped memory for the server process.
db_averageResponseTime	Average response time for database server
db_throughput	Throughput for database server
totalMethodCalls	Total number of bean module method calls

TCModuleStatsCache, TCModuleInstanceStatsCache

These log files record performance statistics for transaction class modules (TCMs) at two levels of aggregation. The TCModuleStatsCache logs at the cluster level, while the TCModuleInstanceStatsCache records data for each server instance. A TCM is an application module executing under a transaction class (middleware application/module/transaction class).

The contents of these files are similar but not identical. The fields are ordered differently and some fields appear only in one of the logs. Confusingly, some fields in TCModuleInstanceStatsCache contain invalid values.

The TCModuleStatsCache.log contains records keyed on the following:

- Transaction class module name

  Where Transaction class module name is a concatenation of the following:

  – Transaction class
  – Application name
  – Module name

- Gateway ID

  Where Gateway ID is a concatenation of the following:

  – Cluster name
  – Fully qualified ODR name

The TCModuleInstanceStatsCache.log contains records keyed on the following:

– Transaction class module name
– Gateway ID
– Server name

Table 9-5 TCModuleStatsCache

TCModuleStatsCache	
Field name	**Description**
timeStamp	Time in milliseconds since January 1, 1970, 00:00:00 GMT
tcmodname	Transaction class module name, a concatenation of the transaction class, the application name and the module name.
gwid	Gateway ID, a concatenation of the cluster name, cell name, node name, and ODR name.
dtname	Deployment target name

<table>
<tr><th colspan="2">TCModuleStatsCache</th></tr>
<tr><th>Field name</th><th>Description</th></tr>
<tr><td>j2eemodname</td><td>J2EE module name</td></tr>
<tr><td>appname</td><td>Application name</td></tr>
<tr><td>tcname</td><td>Transaction class name</td></tr>
<tr><td>scname</td><td>Service policy name</td></tr>
<tr><td>nodegroup</td><td>Node group name</td></tr>
<tr><td>cell</td><td>Cell name</td></tr>
<tr><td>proxy</td><td>Proxy or on demand router name</td></tr>
<tr><td>arrivals</td><td>Number of requests that arrived during the reported interval</td></tr>
<tr><td>executingInt</td><td>Integral over each millisecond in the reported interval, of the number of requests that were executing at the start of the interval, in units of milliseconds * requests.</td></tr>
<tr><td>lengthInt</td><td>Integral over each millisecond in the reported interval, of the number of requests in the queue at the start of the interval, in units of milliseconds * requests.</td></tr>
<tr><td>currentLen</td><td>Length of the queue at the end of the reported interval departs: number of requests dispatched to server.</td></tr>
<tr><td>departs</td><td>Number of requests in the interval that were dispatched from the ODR to a server for a specific <app>|<mod>|<tranClass>|<servicePolicy>|<odr>. If the request fails and gets re-routed, it is still considered to be dispatched just once.</td></tr>
<tr><td>dropped</td><td>Number of requests dropped due to queue overflow.</td></tr>
<tr><td>waittm</td><td>Total wait time in queue of all requests in interval.</td></tr>
<tr><td>resptm</td><td>Total response time for requests (includes wait time and service time).</td></tr>
<tr><td>servicetm</td><td>Total service time for requests serviced: number of requests serviced.</td></tr>
</table>

TCModuleStatsCache	
Field name	**Description**
serviced	This is the number of requests in the interval that were serviced by the server for a specific <app>|<mod>|<tranClass>|<servicePolicy>|<odr>.
begintm	Start time for statistics interval, in milliseconds.
endtm	End time for statistics interval, in milliseconds.
qlen	Total queue length over in the interval.
abvgoal	Number of requests that both returned during the reported interval and had a response time above their service class threshold.
workFactors	The amount of work consumed per request by a TCM. Described in "Calculating compute power consumed" on page 67.

Table 9-6 TCModuleInstanceStatsCache

TCModuleInstanceStatsCache	
Field name	**Description**
timeStamp	Time in milliseconds since January 1, 1970, 00:00:00 GMT
tcmodname	Transaction class module instance name
gwid	Gateway ID
j2eemodname	J2EE module name
dtname	Deployment target name
scname	Service policy name
appname	Application name
tcname	Transaction class name
server	Server name
node	Node name
nodegroup	Node group name cell: cell name

TCModuleInstanceStatsCache	
Field name	**Description**
proxy	Proxy or on demand router name
arrivals	No meaningful value
currentLen	No meaningful value
lengthInt	No meaningful value
executingInt	Integral over each millisecond in the reported interval of the number of requests that were executing at the start of the interval
departs	Number of requests dispatched to server
dropped	No meaningful value
waittm	Total wait time in queue of all requests in interval
resptm	Total response time for all requests in interval
servicetm	Total service time for all requests in interval serviced
serviced	Number of requests serviced
begintm	Start time of interval
endtm	End time of interval
qlen	Total queue length over in the interval
abvgoal	Number of requests that both returned during the reported interval had a response time above their service class threshold
workFactors	Work coefficient

FineGrainedPowerConsumptionStatsCache

This log file contains fine grained power and work consumption data. A record is written for every TCM/server instance. This gives a record for every middleware application, module, transaction class, and server instance that has had work routed through an ODR. There are additional fields that hold relationship information such as the cluster that the server belongs to, the nodegroup the cluster is associated with, and the service policy with which the transaction class is associated.

.

FineGrainedPowerConsumptionStatsCache	
Field name	**Description**
timeStamp	Timestamp of when the data is logged. Time in milliseconds since January 1, 1970, 00:00:00 GMT.
tcmodname	Transaction class module name, a concatenation of the transaction class, the application name, and the module name.
gwid	Gateway ID, a concatenation of the cluster name, cell name, node name, and ODR name.
cell	Cell name
appname	Application name
modulename	Module name
servicepolicy	Service policy
tcname	Transaction class name
server	Server name
node	Node name
odr	ODR name
cluster	Cluster name
nodegroup	Node group name
begintm	The begin time of the interval
endtm	The end time of the interval

FineGrainedPowerConsumptionStatsCache	
Field name	**Description**
workfactor	The estimated work factor from the work profiler for the request type. See “Calculating compute power consumed” on page 67.
numserviced	The number of requests serviced of this type
workcompleted	The amount of work completed in the interval, calculated as numserviced*workfactor
powerconsumed	The power consumption, calculated as numserviced*workfactor/(endtm-begintm)
nodepower	This is the total power available for the node
nodeworkpotential	The total amount of work that the node could accommodate over the interval (totalnodepower*(endtime-begintime))
cellpower	The total amount of power available for consumption for the cell—this is a sum of the nodepower over all nodes in the cell
cellworkpotential	The total amount of work that the cell could accommodate over the interval (totalcellpower*(endtime-begintime))

ServerPowerConsumptionStatsCache

This file is a consolidation of the FineGrainedPowerConsumptionStatsCache at the server level with some additional server data.

ServerPowerConsumptionStatsCache	
Field name	**Description**
timeStamp	Time in milliseconds since January 1, 1970, 00:00:00 GMT
cell	Cell name
name	Server name
node	Node name
cluster	Cluster name
nodegroup	Node group name
begintimestamp	The begin time of the interval
endtimestamp	The end time of the interval
cpu	The average server % CPU utilization over the cputimeinterval. See "Calculating compute power consumed" on page 67.
workcompleted	The amount of work completed by the server in the interval (cpu*nodepower)*(endtime-begintim in seconds)/100.
powerconsumed	The power consumption by the server (cpu*nodepower)/100.
nodepower	This is the total power available for the node
nodeworkpotential	The total amount of work that the node could accommodate over the interval (totalnodepower*(endtime-begintime))
cellpower	The total amount of power available for consumption for the cell—this is a sum of the nodepower over all nodes in the cell
cellworkpotential	The total amount of work that the cell could accommodate over the interval (totalcellpower*(endtime-begintime))

Perl script to calculate CPU utilization

The following script can be used to calculate CPU utilization by cluster.

Important: This code sample is provided "as is" to illustrate a technique you might choose to use. It has not been thoroughly tested. It assumes that there are only dynamic clusters in the environment, and so it will not provide the compute power metric for static clusters or singleton servers. Also, it assumes that the time stamp is a single field. If you choose to use some of or all of this sample, you are responsible for ensuring that it meets your requirements and for testing.

Example: A-1 Sample Perl script for calculating CPU utilization by clusters

```
#!/usr/bin/perl

$|=1;

#print "size = $#my_array\n\n";

$ss_file = "ServerStatsCache.log";
$ns_file = "NodeStatsHistoricCache.log";
$ds_file = "DCStatsCache.log";

open (SS, $ss_file) or die "can't open $ss_file";
open (NS, $ns_file) or die "can't open $ns_file";
open (DS, ">$ds_file") or die "can't open $ds_file";
print DS "timestamp, name, numinstances, utilization\n";

# get header line from each file
$ss_header = <SS>;
$ns_header = <NS>;

# get header indices of desired fields
$ss_timestamp_idx = get_header_index("timestamp", $ss_header);
$ss_name_idx      = get_header_index("name",      $ss_header);
$ss_node_idx      = get_header_index("node",      $ss_header);
$ss_weight_idx    = get_header_index("weight",    $ss_header);
$ss_cpu_idx       = get_header_index("cpu",       $ss_header);
$ns_timestamp_idx = get_header_index("timestamp", $ns_header);
$ns_nodename_idx  = get_header_index("nodename",  $ns_header);
$ns_nodespeed_idx = get_header_index("nodespeed", $ns_header);
```

```
# sequentially process each record in server stats file and node stats
file
# record = all lines with a particular time stamp

$done = 0;
$record_index = 0;
while (!$done)
{
    undef @ss_record;
    undef @ns_record;

    $ss_timestamp = get_record("ss");
    $junk         = get_record("ns");

    if (($junk==-1) || ($ss_timestamp==-1)){last;} #break out of while
loop

    # parse ss lines
    undef %data_cluster;

    for ($i=0; $i<=$#ss_record; $i++)
    {

        @fields = split(/,/, $ss_record[$i]);
        $cluster_name = $fields[$ss_name_idx];
        @fields_test = split("_", $cluster_name);
        if ($#fields_test >0)
        {
            $cluster_name = $fields_test[0];

            $size = $#{$data_cluster{$cluster_name}};

            $data_cluster{$cluster_name}[$size+1] = $ss_record[$i];
        }
        else
        {

        }
    }



    # parse ns record and store data
```

```
for ($i=0; $i<=$#ns_record; $i++)
{
    @fields = split(/,/, $ns_record[$i]);
    $node_name = $fields[$ns_nodename_idx];
    $data_node{$node_name} = $fields[$ns_nodespeed_idx];
}



# calculate numinstances and utilization
for $cluster (keys %data_cluster)
{
    # calculate numinstances
    $count{$cluster} = 0;

    for $i (0..$#{$data_cluster{$cluster}})
    {

        @fields = split(/,/, $data_cluster{$cluster}[$i]);

        $weight = $fields[$ss_weight_idx];

        if (($weight != 0) && ($weight ne ""))
        {
            $count{$cluster}++;

        }
    }

    # calculate utilization
    $sum_numer = 0;
    $sum_denom = 0;
    undef %count_nodenames;
    for $i (0..$#{$data_cluster{$cluster}})
    {
        @fields = split(/,/, $data_cluster{$cluster}[$i]);
        $node = $fields[$ss_node_idx];
        $count_nodenames{$node}++;
        $cpu  = $fields[$ss_cpu_idx];
        $nodespeed = $data_node{$node};
        $sum_numer += ($nodespeed * $cpu);
        # if haven't already encountered this node, add it to the
sum
        if ($count_nodenames{$node}<=1)
        {
```

```
                $sum_denom += ($nodespeed);
            }
        }

        $utilization = $sum_numer / $sum_denom;

        print DS "$ss_timestamp, $cluster, $count{$cluster},
$utilization\n";
    }



    $record_index++;
}





close (SS);
close (NS);






sub get_record
{
    my ($flag) = @_;
    my $done = 0;
    my $j=0;
    my $ss_timestamp=0;

    if ($flag eq "ss")
    {
        $j=0;
        if (defined($ss_record[$j] = <SS>))
        {
            @fields = split(/,/, $ss_record[$j]);
            $ss_timestamp = $fields[$ss_timestamp_idx];
            $done=0;
            while(!$done)
            {
                $temp = <SS>;
```

```
            @fields = split(/,/, $temp);
            $ss_timestamp_temp = $fields[$ss_timestamp_idx];
            if ($ss_timestamp_temp eq $ss_timestamp)
            {
                $j++;
                $ss_record[$j] = $temp;
            }
            else
            {
                $done=1;
            }

        }
    }
    else {return -1;}
}
elsif ($flag eq "ns")
{
    $j=0;
    if (defined($ns_record[$j] = <NS>))
    {
        @fields = split(/,/, $ns_record[$j]);
        $ns_timestamp = $fields[$ns_timestamp_idx];
        $done=0;
        while(!$done)
        {
            $temp = <NS>;
            @fields = split(/,/, $temp);
            $ns_timestamp_temp = $fields[$ns_timestamp_idx];
            if ($ns_timestamp_temp eq $ns_timestamp)
            {
                $j++;
                $ns_record[$j] = $temp;
            }
            else
            {
                $done=1;
            }

        }
    }
    else {return -1;}
}
else
{
```

```
        print "\nerror\n\n;"
    }

    return $ss_timestamp;
}


sub get_header_index
{
    my ($label, $header) = @_;
    my $index = -1;
    my $i = 0;

    @fields = split(/,/, $header);
    for($i=0; $i<$#fields; $i++)
    {
        if ($fields[$i] =~ /$label/i)
        {
            $index = $i;
            return $index;
        }
    }
    return -1;

}
```

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 360. Note that some of the documents referenced here may be available in softcopy only.

- *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153
- *WebSphere Application Server V6 System Management & Configuration Handbook*, SG24-6451
- *WebSphere Application Server V6.1: System Management and Configuration*, SG24-7304

Online resources

The following Web sites are also relevant as further information sources:

- *ObjectGrid Programming Guide*

  http://www-1.ibm.com/support/docview.wss?uid=swg2700632
- WebSphere Extended Deployment Information Center

  http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp
- WebSphere Application Server Information Center V6.1

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp
- WebSphere Application Server Information Center V6.0

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp
- *Solving Business Problems with WebSphere Extended Deployment*

  http://www-128.ibm.com/developerworks/websphere/library/techarticles/0606_antani/0606_antani.html
- IBM Tivoli Usage and Accounting Manager

  http://www-306.ibm.com/software/tivoli/products/usage-accounting/

- Tivoli Software Information Center

  http://publib.boulder.ibm.com/tividd/td/IBMTivoliUsageandAccountingManager6.1.html

- *Tivoli Usage and Accounting Manager Administrator's Guide*

  http://publib.boulder.ibm.com/tividd/td/IBMTivoliUsageandAccountingManager6.1.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

E

F

G

W

X

IBM

Redbooks

Best Practices for Implementing WebSphere Extended

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Best Practices for Implementing WebSphere Extended

Optimize for service levels, health management, and application hosting

Deploy long-running applications in WebSphere

Enhance session data sharing capabilities

This IBM Redbook looks at scenarios for using WebSphere® Extended Deployment and outlines procedures and best practices for these scenarios. Scenarios for operations optimization, long-running application extenders, and data-intensive application extenders are included.

This book focuses on process, design, and usage guidelines, complimenting the "how to" information found in *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153. In addition, the business grid (batch and compute-intensive) capabilities are covered extensively including *how to* and *best practice* information.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7343-00 ISBN 0738489638