IBM WebSphere Partitioning Facility for WebSphere Extended Deployment v.5.1

Management and Development Guide

October 2004

First Edition (October 2004)
This edition applies to IBM WebSphere Application Partitioning Facility for WebSphere Extended Edition Version 5.1, and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Notice

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.

Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this document at any time without notice. Information in this document could include technical inaccuracies or typographical errors. IBM Corporation does not warrant that this document is error free.

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.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available. 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.

Trademarks

- IBM, the IBM logo, WebSphere, AIX and DB2 Universal Database are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both.
- Sun, Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Linux is a registered trademark of Linus Torvalds.
- Microsoft, Windows, Windows NT and Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States or other countries or both.
- Other company, product, and service names may be trademarks or service marks of others.

Preface

This book, *IBM WebSphere Application Partitioning Facility for WebSphere Extended Edition Version 5.1 –* **Management** *& Development Guide,* provides a guide for managing the operation aspect of WPF enabled applications and provides a getting started and reference resource for developing them.

Who should read this book

System administrators who will be involved in deploying and managing J2EE applications that use WPF Partitioned Stateless Session beans.

Developers who are responsible for developing J2EE applications that will must be deployed in a high performance computing environment. Developers will create a new type of Stateless Session bean, a Partitioned Stateless Session Bean. This sort of Stateless Session bean is simply a normal EJB that implements the WPF programming framework interfaces. Developers can also enhance http and database system scalability by developing support for partitioned web applications and partitioned databases offered via the extended interfaces offered with WebSphere's Extended Deployment WPF subsystem.

This guide will describe how to manage and develop high transactional rate capable applications with resource access patterns conducive to logical partitioning. The installation guide is written for experienced J2EE developers who are familiar with WebSphere Applicator Server. Prior exposure to WebSphere Studio Application Developer is helpful.

Your Comments are Important to Us

IBM Corporation technical staff values your comments. As we write and revise, your opinions are the most important feedback we receive. Please use the reader's comment form at the back of this book to tell us what you like or dislike about this installation guide. If you prefer, FAX at 507-253-3495 or write to us at the following address:

IBM Corporation
Department XDTA
3605 Highway 52 North
Rochester, MN 55901

Related Publications

The following publications are recommended for use with this Guide:

- IBM Application Partitioning and Distributed Work Manager for WebSphere Network Deployment Version 5.1 - Installation Guide
- Overview of WebSphere Application Server for Linux
- WebSphere Application Server Enterprise, Version 5 (5630-A37)
- IBM WebSphere Studio Application Developer Version 5.1
- IBM WebSphere V5.1 Performance, Scalability And High Availability, SG24-6198[1]

Occasionally, this guide refers you to other IBM publications for system-specific information. Typically, these publications are called installation or user's guides, but their exact names vary by operating system and platform.

Conventions used in this book

Table 1: The following typographical, text formatting and key conventions are used in this guide:

Convention	Meaning
Bold	When referring to graphical user interfaces (GUIs), bold face indicates menus, menu items, labels, buttons, icons, and folders. It also can be used to emphasize command names that otherwise might be confused with the surrounding text.
`Monospace`	Indicates text you must enter at a command prompt. Monospace also indicates screen text, code examples, and file excerpts.
Italics	Indicates variable values that you must provide (for example, you supply the name of a file for *fileName*). Italics also indicate emphasis and the titles of books.
Ctrl-*x*	Where *x* is the name of a key indicates a control-character sequence. For example, Ctrl-c means hold down the Ctrl key while you press the c key.
`%`	Represents the UNIX command-shell prompt for a command that does not require root privileges.
`#`	Represents the UNIX command-shell prompt for a command that requires root privileges.
`C:\`	Represents the Windows command prompt.
Entering commands	When instructed to "enter" or "issue" a command, type the command and then press Return. For example, the instruction "Enter the **ls** command" means type **ls** at a command

[1] Redbooks are available via download in Adobe PDF format from www.redbooks.ibm.com

	prompt and then press Return.

Table of Contents

1 WebSphere Partitioning Facility Overview

WebSphere Extended Deployment offers the WebSphere Partitioning Facility (WPF). This new functionality supports the concept of partitioning for EJBs, HTTP traffic and database access. WPF is both a programming framework and a system management infrastructure. To utilize WPF, developers must implement the WPF framework within their typical J2EE applications.. WPF is intended for an organization interested in continuing to utilize J2EE Application Development tools and application sever support, but who are also willing to develop and manage their applications in a more aggressive fashion to achieve higher total transaction volumes.

In general, the goal of partitioning is to support higher volumes of transaction activity, yet scale efficiently. The following sections describe briefly the various partitioning offerings provided in WebSphere Extended Deployment.

Updates to this document, WPF white papers and other Extended Deployment information can also be found on-line at:

http://www-306.ibm.com/software/webservers/appserv/extend/support/

1.1 Partitioning Objective and Benefits

The primary advantage of partitioning is to provide the ability to more specifically control resources during cluster member execution. Requests can be routed to a specific application server that has exclusive access to some computing resource such as a dedicated server process or database server that handles a specific data set. The requests could be an HTTP, EJB or database request or update. The endpoint receiving the work is still highly available. Consequently, WPF offers functionality to route work to a particular cluster endpoint. This reduces overall system overhead while continuing to offer the safety of rapid failure recovery of each endpoint.

For example, assume that an application is being created that tracks weather system status as new weather systems dynamically occur. In any given evening, many storms are in progress nationally. If the database information is partitioned by a particular storm and a WPF partition for each storm is created, that application server hosting the partition could load the information exclusively. This allows both the storm experts updating the quickly changing information and the clients that need to render the storm information to avoid contention to acquire and present the information back to end-users to the extent possible. Clients that need to update the information will be routed to one location within the cluster, and can update the in memory copy. The in memory copy can be persisted as the opportunity allows without slowing down the consumers of the information or the updaters of the storm status. As the storm passes, it can be persisted to the database and the partition removed to make room for a new weather event.

In normally cluster applications, this is not generally possible without very extensive application design and implementation. For example, in a common J2EE application implementation each client requesting information about or updating the status of the storm would have requests randomly directed to various cluster members. Each cluster member request would result in a transaction being created. To satisfy the request, each read for example would require the data to be loaded from the database, taking additional time and causing extra database server overhead. The most intensive operation in the cluster would be a database update, locking out the readers of the application data. Each update would require exclusive access to the storm data, and would lock out other readers (depending on the isolation level chosen) across the cluster until the update is complete. In this case, the database also has to track the various requests pending on specific information and arbitrate between them. This causes additional database load, and several cluster member requests will take longer to satisfy, in effect reducing the cluster throughput for that applications.

In summary, the intent of partitioning is to direct workload to a given member, and reduce the overall cluster overhead for each request. As these individual savings are accrued, all requests for the same application information or operation should benefit in terms of lack of contention. WPF also offers the ability to make each endpoint, a partition, highly available and manageable. Thus, the general benefits of cluster failover and recovery are still present with WPF, plus the additional functional capability to reduce resource contention. Reducing the resource contention will result in higher overall cluster throughput.

1.2 J2EE Partitioning Capabilities

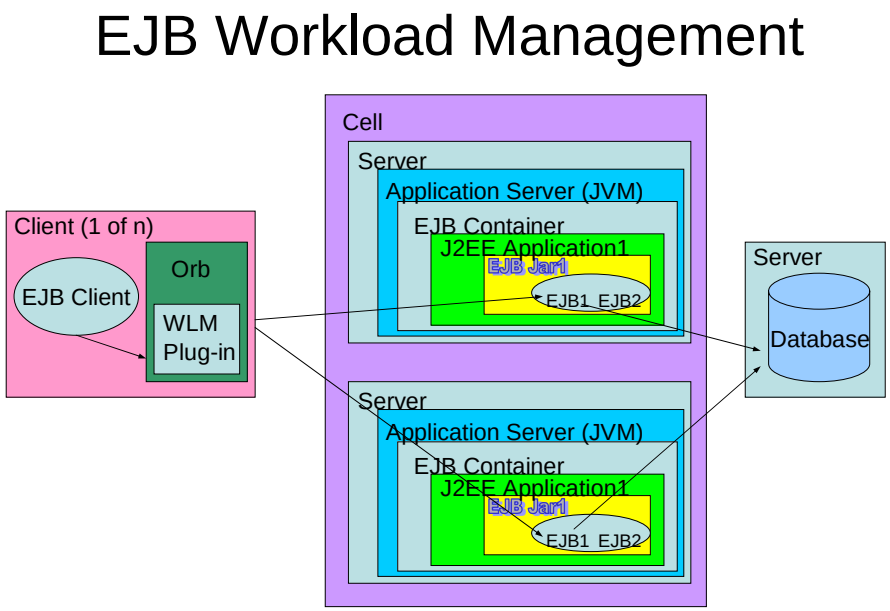
WPF is designed to augment typical J2EE technologies, including EJB, HTTP and database workloads with partitioning capabilities.

1.2.1 EJB Workload Partitioning

This section explains the basic concept of a Partitioned J2EE Application with EJBs, which is simply a normal J2EE Application with a Partition Stateless Session Bean. In addition, it describes the "cluster" view of this entity as well as how it differs from a typical EJB involved in EJB Workload Management provided in the WebSphere Application Server and Extended Deployment offerings.

1.2.1.1 Typical J2EE EJB Clustered Workload Processing

The following diagram illustrates a simple 2 node, 1 application server per node cluster configuration. The diagram depicts an EJB Workload Management[2] example a J2EE application with 2 EJBs:



In this case, each client request is routed from the EJB Client via the ORB and WLM Plug-in to EJB1 in an alternating fashion between EJB instances. Both EJB1 instances are active in the cluster, but they are not unique, work is routed between them invisibly without client intervention. While the ability to share the requests helps scalability, there are implicit limits and constraints to ensure the same data loaded in each entity during normal transaction execution is safely managed and replicated back to the database. Thus, the invisible functionality provided in the EJB Container, Adapter and other WebSphere components ensures data corruption is not possible, but does take away from the performance capabilities of the system. For some workloads this performance cost is relatively expensive and several application styles exist where this can be avoided.

[2] We suggest the reader review the first 6 chapters of the IBM Redbook IBM WebSphere V5.1 Performance, Scalability, and High Availability for background information. The redbooks can be downloaded in Adobe PDF form free of charge from www.redbooks.ibm.com.

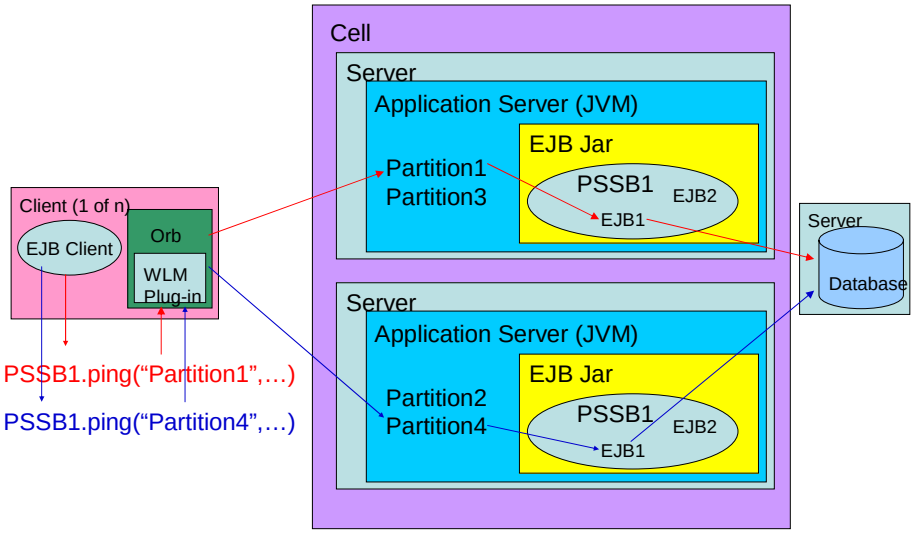
The ability to allow two or more entity bean instances to share the same data is enforced by WebSphere and the Database server. One of WPF's goal is to allow a single endpoint in the cluster to handle all data for a specific instance, and reduce the burden of the WebSphere and Database server to enforce these semantics, dramatically improving overall system scalability and throughput. Currently, WebSphere Application Server and Extended Deployment entity bean developers can use only Options B and C, because Option A exclusive access is not permitted.

1.2.1.2 WPF EJB Workload Management

The use of the WebSphere Partition Facility now supports the concept of Option A entity beans in the cluster. This extra performance does not come about magically, the developer must leverage the programming model provided with WPF, and in some cases use the Async Beans support provided in the WBI-SF.

The WPF services require one additional Stateless Session Bean to be included in the J2EE Application's EJB jar. This session bean is called a Partitioned Stateless Session Bean. When the application processes the startup sequence during application server start for the EJB module, 1 to N different "Partitions" are created. Each "Partition" is simply a uniquely addressable end point within the cluster. Thus, in summary, a Partitioned Stateless Session Bean (PSSB) is simply a typical Stateless Session that utilizes the WPF framework's PartitionManager to create individual partitions at bean startup, and implements in addition to the normal SessionBean interface, the PartitionHandlerLocal interface (described subsequently) to process Partition related life cycle events.

The WPF Partition support services allow a PSSB to create partitions. Each partition is simply an endpoint, directly accessible by the EJB Client. For this example, assume the Partitioned Stateless Session Bean, e.g. PSSB1 is included in the J2EE Application, and at startup creates partitions Partition1, Parititon2, Partition3 and Parittion4. Thus, the EJB Workload Management diagram above when the Partitioned J2EE Application is installed (simplified to focus on partitions) would look like the following.



The diagram above reflects an EJB Client executing the same PSSB1 method "ping" twice. For each invocation, the first parameter differs in a String value that is provided as method state. Later in this document, how the mapping occurs is described in more detail, but for now the value of the first parameter is a key and used by WLM to route the request to a particular partition endpoint within the cluster. At cluster started up, partitions Partition1, Partition2, Partition3 and Partition4 were created when each server's Partitioned J2EE Application was started and the PSSB1 home's instance was instantiated. The partitions created are not "servers" or JVMs, but each does have a life cycle

concept similar to a normal EJB instance when instantiated within an EJB Container. One way to think of a partition is as an addressable or routable endpoint in the WLM infrastructure.

In the first case, the EJB Client acquired the remote interface of the PSSB1 via JNDI as normal, then executed the first method ping(…) with the first parameter specifying “Partition1”. WebSphere has been augmented with Extended Deployment to allow programmers to advise the runtime how to process each method invocation’s parameters, and based upon the state in the method call route each call to a specific partition.

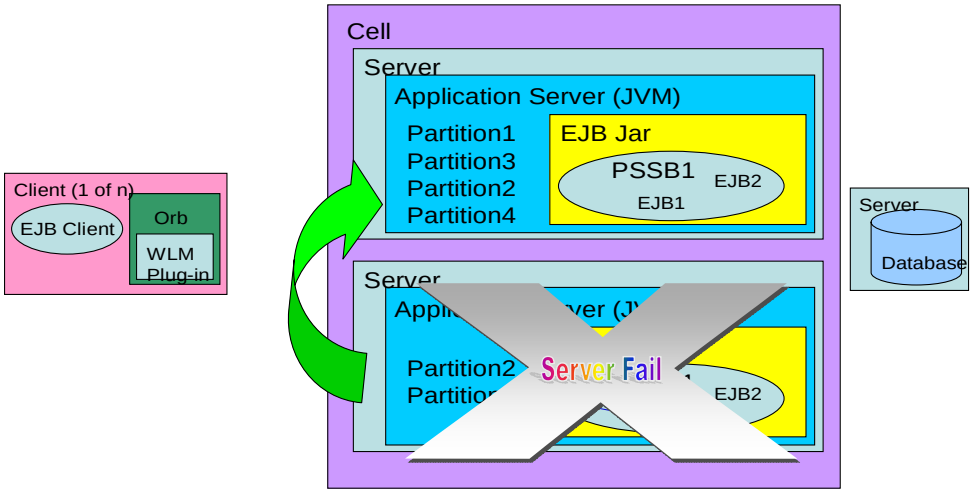
For programming staff reading this document, the challenge for their implementation of this technology is to collocate requests with certain method state to a given application server that has exclusive access to some resource. The programmer is given extra tools during partition activation and execution to be notified when and where a specific partition is loaded and unloaded. With this information, the programmer can allocate all information for a specific partition in one application. In addition, the administrator is not limited in where this processing occurs in the cluster. The partition can be moved at any time.

1.2.1.3 Administrating Capabilities of a WPF Enabled EJB Workload

The administrator can adjust this behavior after the initial bean startup to meet operational requirements as required. For example, in the case above, if Parititon2 and Partition4 are both very heavily loaded, and Partitions 1 and 3 are not, the administrator could move Partition2 to the other application server in the cluster. In this case, the programming staff would code the implementation of the PSSB1 Stateless Session Bean to handle a partittionUnloadEvent(…) and a partititonLoadEvent(…) method. During the execution of the move partition initiated by the administrator, the partition Parition2 would receive an unload event allowing all references to the database and other j2ee application resources to be cleaned up and removed from any sort of caching implementation. Subsequently, and very quickly, Partititon2 would be reactivated on the other application server and receive a partition load event, enabling the bean developer to reinitialize the state and prepare to process transactions.

So far, this is interesting, but the high availability story has not been addressed. The interesting feature of this technology is not only that the client side can completely control the routing, and independently the server side administrator can control the final destination, but also how failures can be handled. Here the inclusion of the new HA Manager component comes to the fore. Although not directly accessible to customers, the HA Manager provides the WPF service the underpinning to detect a failed application server. The HA Manger also correlates, very quickly, the relationship between the partitions on that server and what actions should be taken to ensure they are activated elsewhere.

The diagram below depicts a very simple case of an application failover:



In the diagram above the application server with partition endpoints Partition2 and Partition4 failed. Upon failure, HA Manager detected the failure, and activated two different instances of Partittion2 and Partition4 on the other application server. This is but one application server recovery scenario, other scenarios addressed include stopping an application server down for maintenance, a network partition event and other physical or management scenarios. In this case, Partition2 and Partition4 experience an outage for a small amount of time, however, the remainders of other partitions continue as if nothing has happened. In addition, Partition2 and Partition4 can implement a level optimization at recovery time because the implementation is advised of the partition reactivation event and can check for any problems that may need to be addressed during transaction recovery.

The endpoint processing can be handled in several ways depending on the computing architecture a customer would like to employ. Many customers choose to have a few, very reliable and manageable servers in their clusters. For these customers, if Partition2 for example experiences an increased load, more LPAR resources could be allocated for that application server. For other customers that utilize more distributed clusters or blade centers, either the Partition2 could be moved to another standalone system that is not used or lightly loaded (does require an outage for Partition2 in this case) or other collocated partitions that are not in use or not heavily loaded could be moved elsewhere in the cluster. This would leave the Partition2 application server more able to handle the load. For an experienced IT staff, either option allows great deal of flexibility to meet operational requirements. The coupling of existing WebSphere functionality and capability with the WPF framework and the underlying HA Manager technology provides a new and exciting capability to make WebSphere a richer clustering solution for high performance functionality and management function to handle many typical cluster challenges.

In summary, the unique function WPF offers is a richer client request model in that the client can explicitly choose where the request should be routed. In addition, not only is the endpoint uniquely addressable and targetable, but also highly available endpoint. If the application server hosting the partition endpoint fails, the HA Manager provided with WebSphere Extended Deployment will detect this and active the Partition instance on another application server in the cluster as denoted in the following diagram. This is achieved without disabling any of the existing function that WebSphere customers have come to rely and expect from a rich clustering technology offering.

1.2.2 HTTP Partitioning

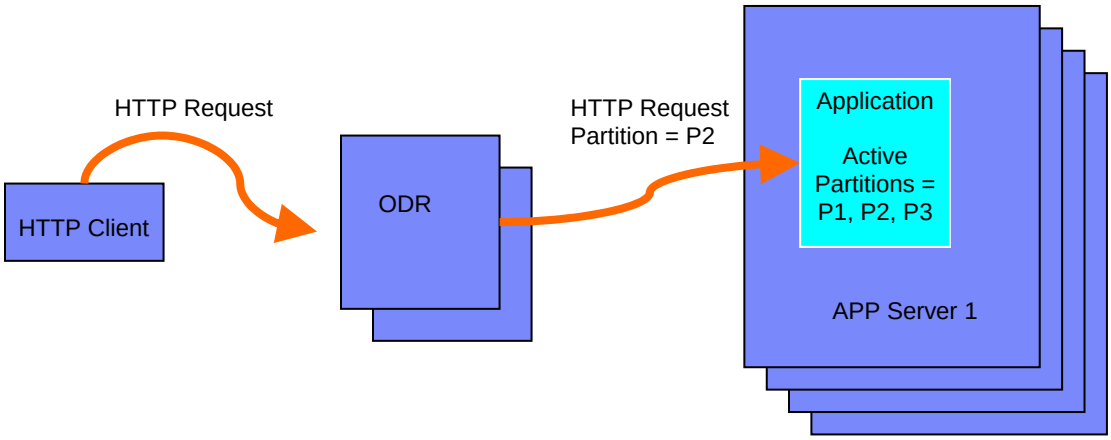
The WebSphere Partitioning Facility (WPF) provides the capability to partition HTTP requests across back-end WebSphere Application Server instances. Known as HTTP Partitioning, this capability works in conjunction with the On-Demand Router (ODR) that receives awareness of partition location and forwards HTTP requests to the appropriate target application server. This section gives an overview of the HTTP Partitioning function. Here, we describe the architecture of the solution and define key concepts required to implement HTTP WPF applications.

HTTP WPF partitions are no different than EJB Workload IIOP-based partitions: they correspond to a related collection of data elements that are commonly accessed together. They have the distinct feature that no data element belongs to more than one partition. This property enables web applications to make inferences in how these data elements may be treated. That is, no data element belongs to two partitions and a single partition is only active on at most one application instance. As a result, when an application receives a request for a particular partition, it can be certain that no other application instance is accessing this partition and the data elements associated with it. As such, the application can leverage this by being more aggressive in treating the partitioned data. When combined with techniques such as caching and batching, this ultimately improves application performance by increasing the efficacy of size-limited caches and reducing the overhead in ensuring data consistency.

HTTP Partitioning operates on the premise that incoming HTTP requests contain sufficient information to identify the partition associated with the request. This places two key restrictions on application architecture: 1) that each HTTP request access data associated with exactly one partition, and 2) that the HTTP URL contains the partition name. Restriction 1 can be typically resolved by merging overlapping partitions into a single partition. Care should be taken when designing the application's URL to ensure that restriction 2 is satisfied.

1.2.2.1 The Role of the On-Demand Router (ODR)

As shown in the figure below, the On-Demand Router (ODR) serves as a reverse proxy between the HTTP client and clustered (partitioned) application. The ODR extracts the partition name from the received HTTP request and routes it to the application server hosting the application instance that is currently serving this partition. In this case, application server 1 is hosting partition P2. The HTTP Partitioning function ensures that requests are routed to the correct server, even in circumstances when a partition is unloaded from one partition and loaded in another.



1.2.2.2 Extracting Partition Names From HTTP Requests

Partition names are extracted from HTTP requests using *request expressions*. A request expression consists of two strings: the *match expression* and the *classifying formula*. Together, they provide a mechanism for classifying HTTP requests based on Java-supported regular expressions. The match expression determines how we match on a portion of the URL and query string. The classify formula indicates the portion of the URL and query string that specifies the partition once the expression has been matched.

To determine whether an HTTP request should be partitioned, the ODR makes use of the Pattern and Matching classes of the java.util.regex package. If there is a match on any application-specified match expression, the request has an associated partition. The ODR first concatenates the URL and query string to form a single string. It invokes the Patterns built from all application-specified match expressions. Upon a match, the classify formula determines how the partition name will be built. A special character, "$", indicates the portion of the match expression to use.

Consider a case where the match expression is "(user=)(*)$" and the classify formula is "$2". The "$2" will take the portion of the match corresponding to the (.*) portion, as it is the second portion of the matching pattern. For example,

if the URL consists of www.ibm.com/something/user=fred, $1 would correspond to “user=”, while $2 would correspond to “fred”.

Consider the match expression “(user=)(*)(rodriguez)$”. Suppose a request arrived with a URL containing “user=adolforodriguez”. With a classify formula of “$2”, the resulting partition name would be “adolfo”. With a classify formula of “mypartition$2”, the request would have a partition name of “mypartitionadolfo”. Likewise, with a classify formula of “$2$3”, the resulting partition name would be “adolforodriguez”.

Also note that match expressions may overlap. For example, if we had two match expressions: “(user=)(*)$” and “(user=)(*)&” and we receive a URL with query string containing “user=adolfo&Submit=Enter”, both expressions would match. The former expression would extract “adolfo&Submit=Enter” with a $2 classify formula, while the latter would extract “adolfo”. Since only “adolfo” is a valid partition name, it would be chosen as the partition name. While it is typical that the “most specific” regular expression was the intended target, HTTP Partitioning does make an attempt to favor a particular expression over the other. Instead, all expressions are applied until a valid partition name is found or none exists.

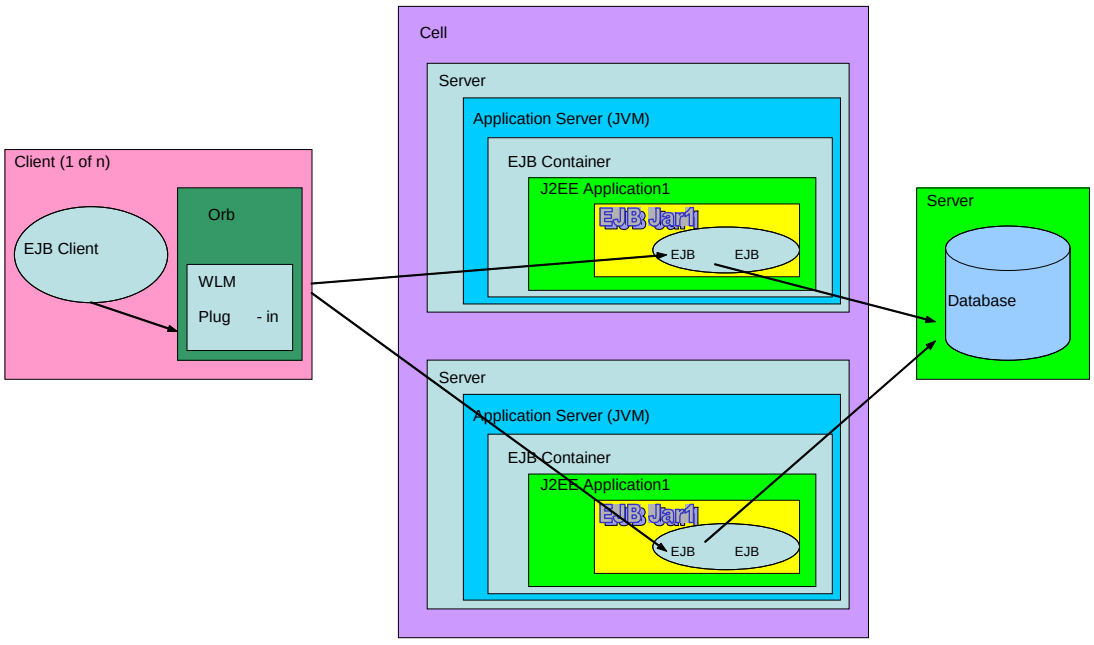
More information describing the management and implementation of HTTP Partitions is described later in this document.

1.2.3 Database Partitioning

Though we can configure a very reliable WebSphere cluster to run the application, if the WebSphere cluster uses a single database instance, then the database becomes a single point of failure as well as a vertical scaling limit. It is a single point of failure because if it becomes unavailable, no work can be done in the WebSphere cluster. This is a vertical scaling problem because most databases only scale vertically, i.e. to go faster; you need to buy a larger box.

The diagram below shows the single database server architecture.

Single Database



In this diagram, there are two WebSphere application servers, but there is only one database server. When more and more application servers are added, the database will become a performance bottleneck.

Some customers require databases that scale horizontally and that do not stop the whole application server cluster when it fails. Many applications whose non-functional requirements allow temporary failures to impact a part of the application but a complete outage is not acceptable. Such architectures use a partitioned database design.

An example of such a design would include three boxes running standalone DB2. The table schema and security system is identical for all three databases. All read only reference data is replicated to all three databases from a master reference data DB2 instance that is made highly available in the normal way. This master reference DB2 instance is not a single point of failure during normal operation, as it is not used by the application directly.

Once multiple servers are ready, the next step is to ensure the application data is partitionable. We will map the data for a particular partition to a particular DB2 instance. This can be done using a simple hashing scheme or a range mechanism, or be done by using a replicated table in the databases. The table is cached by the application server cluster and specifies the DB2 instance holding the data for a particular partition. This allows 'hot' data to be moved between databases without requiring application changes.

The partition to DB2 instance table is maintained by the master and replicated to all three database nodes. An application protocol will be needed to coordinate when a partition is moved from one database node to another. This also allows an application to add a database instance to the set of databases in use for horizontal scalability. The advantage of using three independent database instances is better availability than a normal clustered database such as Oracle RAC or DB2 EEE. The databases are independent and a failure of one of the databases just means that the set of data residing there is now unavailable but the application can continue to process transactions for the data residing in the other online database instances. This is much more preferable to a complete failure. However, the administration is now more complex as there are three databases instead of one. The application uses what is called 'Directed Transactions' to tell the application server which database instance contains the complete data for the next transaction. This pattern allows very flexible management of the database aspect of the application especially when used with the MAPPER table that tells the application which database node has the data for a particular partition.

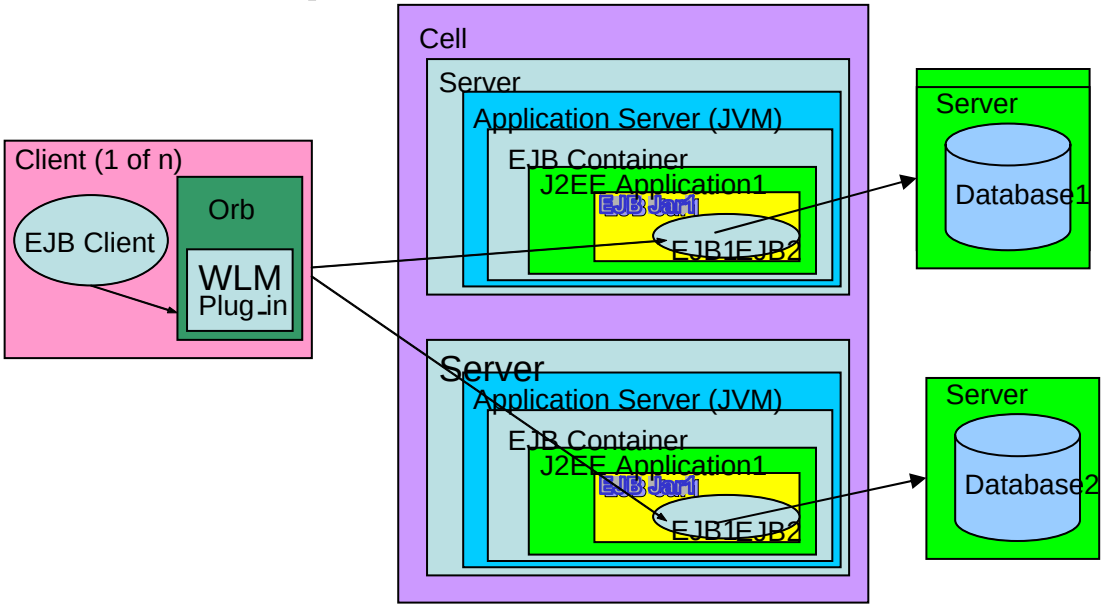
Applications that use CMP beans normally specify a single database to use with the CMP beans. This approach clearly has problems when using this pattern. You could deploy the CMP beans N times, once for each database but this is not very flexible for the following reasons:

- Requires N copies of the code with N JNDI names etc
- Requires a deploy step when a database is added.
- It is just not that easy to manage.

WebSphere Extended Deployment offers a new feature, the Proxy DataSource that allows the application to tell WebSphere which database to use before the transaction starts. This means that when a cluster member receives a request for a particular application partition then it can tell the CMP runtime to ignore the DataSource the beans were deployed with and instead use a specific DataSource for the duration of the next transaction. This allows the directed transaction pattern to be used with the application, enabling it to increase its availability, and allows the database tier to scale horizontally on blade type environments. The applications can also take advantage of the MAPPER table pattern to very flexibly manage data and move partitions around to better manage the operational aspects of an application such as how to move a very busy partition to a lightly loaded database node for performance reasons.

The following diagram shows the new architecture.

Multiple Database nodes



In this diagram, we have two databases in the system. EJB1 is deployed in both servers. In one transaction, the EJB1 in the top server access database 1. And in another transaction, EJB1 in the server below access database 2. So the database load is spread across to several database servers, instead of just one server.

2 Introduction to WPF via Example

WPF is a rich and functional offering. This section describes the steps required to install WebSphere Extended Deployment, configure the cluster, install one the samples provided with WebSphere Extended Deployment, and perform several usage and system management examples as a precursor to diving into the functional specification.

The samples are installed from the installableApps directory in the Network Deployment Manager (ND) directory tree. Prior to installing the application, WebSphere Extended Deployment must be installed on top of Application Server and ND, or Base, Extended Deployment and WBI-SF and required service levels.

2.1 WebSphere and WebSphere Extended Deployment Installation Steps

For this step, the latest and most current up to date information regarding installing Extended Deployment is located in the WebSphere Extended Deployment Installation Guide. The WBI-SF steps (in *italic*) are optional if your configuration will be limited to Base/ND.

Install the components in this manner (see the WebSphere Extended Deployment and other WebSphere install guides for more specific information):

- Pick an operating system, either Windows XP, Advanced Server, Linux , AIX, etc…
- Install WebSphere Network Deployment (ND) on primary management machine
- Install WebSphere Application Server (Base) on each cluster member
- *Install WBI-SF over the Network Deployment and each Application Server installed above*
- Upgrade to WAS Network Deployment 5.1.1 on the Network Deployment management machine
- Upgrade to WAS Application Server 5.1.1 over each of the Application Server installations
- *Upgrade to WBI-SF 5.1.1 over each location WBI-SF is installed (every Network Deployment and Application Server configuration)*
- Upgrade to WAS Network Deployment 5.1.1.1 over the Network Deployment management machine
- Upgrade to WAS Application Server 5.1.1.1 over each of the Application Server installations
- There is a JDK service level for 5.1.1.1 that should be installed.
    a) Upgrade to WAS Network Deployment JDK.
    b) Upgrade to WAS Application Server JDK.
- Install WebSphere Extended Deployment 5.1 over each Network Deployment and Application Server install (similar nodes to that installed with WBI-SF)
- You can optionally install your database server of choice on each node where non-partitioned or partitioned data will be placed

2.2 Configuration Quick Start

The steps below describe the configuration of a cluster than can host a WPF Partitioned J2EE Application. Even though a more optimized development environment can be offered for WPF developers, from an administration perspective, three application servers are used to make this example more interesting. In this case, we will describe a single node

scenario. However, the steps can be followed as a guide to get a multi-node cluster running (will point these divergent steps out as well).

In addition, we will use one example in this section. The other samples can be installed and operated similarly. See the samples overview section for more details.

2.2.1 Starting the Deployment Manager

The Deployment Manager provides the configuration, as well as visualization interface for WebSphere Extended Deployment. This server must be running prior to adding any other application server machine to the cluster. At this time, using the command line launch the Deployment Manager:

```
cd <ND Home Directory>\bin
startmanager
```

The result should be something similar to:

```
ADMU0116I: Tool information is being logged in file
        C:\was\nd51\logs\Deployment Manager\startServer.log
ADMU3100I: Reading configuration for server: Deployment Manager
ADMU3200I: Server launched. Waiting for initialization status.
ADMU3000I: Server Deployment Manager open for e-business; process id is 2568
```

2.2.2 Add each node to the Deployment Manager

In an Extended Deployment environment, each node must be added to the Deployment Manager's administrative domain. This will enable the Deployment Manager to add a particular server's Application Server to the Deployment Manager's cell. The following invocation will add the node to the current Deployment Manager:

```
cd <WAS Base>\bin
addnode <Deployment Manager hostname>
```

The result should be something similar to:

```
ADMU0116I: Tool information is being logged in file
        C:\was\base51\logs\addNode.log
ADMU0001I: Begin federation of node NodeA with Deployment Manager at
        xxxxxx:8879.

…

ADMU0305I: Install applications onto the Cell cell using wsadmin $AdminApp or the Administrative Console.
ADMU9990I:
ADMU0003I: Node NodeA has been successfully federated.
```

2.3 Cluster Configuration

Launch the administration console to manage which applications are installed within the cluster. The URL:

URL: http:://localhost:9090/admin

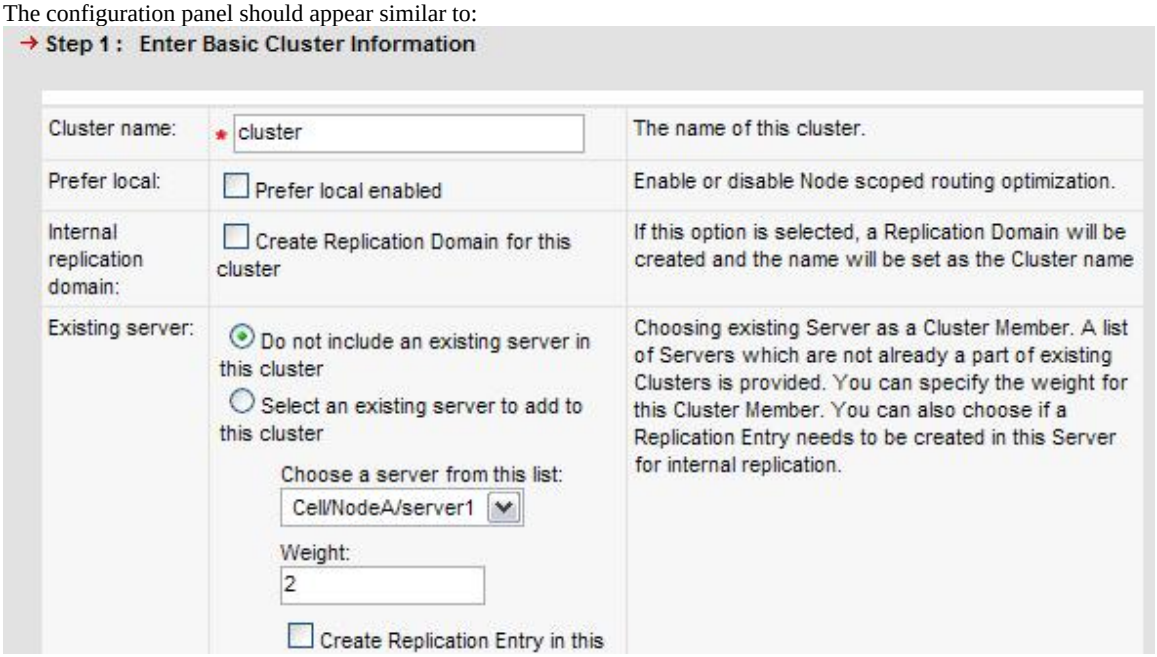
or if on a remote:

host: http://<Deployment Manager hostname>:9090/admin

To configure the cluster:
Select **Servers** in left hand column
Select Clusters in left hand column

Select the **New** option in Server Cluster pane
Enter the desired cluster name
Uncheck the **Prefer Local Enabled** option

The configuration panel should appear similar to:



Click on Next

For this example, we will create 3 application servers, all on NodeA (more than one node can be used),

Type “cluster_member_1” in the name field, select Apply.
Type “cluster_member_2” in the name field, select Apply.
Type “cluster_member_3” in the name field, select Apply.
Select Next.

Select Finish.

Select Save

Check the box for “Synchronize changes with Nodes”.

Select Save

2.3.1 Installing the WPF Example Application

For this scenario, the D_WPFKeyBasedPartition.ear will be installed on the application server. In the default case, the application creates 10 partitions, allowing the user to balance across cluster member application severs. During the WebSphere Extended Deployment installation, the WPF samples applications are installed in <Deployment Manager Home>\installableApps.

Select Applications
Select Install New Application
Select Browse on the Local option, browse to <Deployment Manager Home>\installableApps and select:
<Deployment Manager Home>\installableApps\D_WPFKeyBasedPartitionSample.ear
Select Next

Select Generate Default Bindings check box

Select Next

*The Install Applications pane should have all the correct defaults. One warning, **do not select the deploy option**. The PSSB stub will be regenerated and disable the PSSB routing. If that is the case, the default round robin functionality will result when client interacts with the partitions. If this occurs, simply reinstall the application again.*

Select Next
For the “Step 2 : Provide JNDI Names for Beans”, select Next.

For the “Step 3 : Map modules to application servers”, select the cluster option for module and Apply.

Select Next

Select Finish

Select “Save to Master Configuration”

Select Save (Synchronize with Nodes options should be selected if not already)

2.3.2 Starting the Cluster

In the Extended Deployment admin console, select the cluster that contains the Partitioned J2EE Application, and start it.

Select **Servers** in left hand column
Select Clusters in left hand column
Select the cluster created above.
Select the **Start** option in Server Cluster pane

Starting the cluster could take a minute or two.

2.4 Executing WPF Operations

This section executes basic WPF operations. The management section provides reference information for each of the following commands.

2.4.1 Verify the application is started, etc…

To verify the partitions has started, in a command window:

cd <Deployment Manager Home>\bin
wpfadmin listActive

The result should be similar to:

```
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is:
DeploymentManager
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000010: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000009: Server Cell\NodeA\cluster_member_1
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000008: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000007: Server Cell\NodeA\cluster_member_1
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000006: Server Cell\NodeA\cluster_member_1
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000005: Server Cell\NodeA\cluster_member_1
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000004: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000003: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000002: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000001: Server Cell\NodeA\cluster_member_1
```


This indicates that the application started and the partitions (10 by default) were activated correctly. The application server instance a particular partition key starts on, without providing a specific policy, depends upon which server reaches a point where it is registered in the HA Manager quorum. Whichever servers are in the view at the time quorum is detected will have activated partitions. Thus, the partitions are not guaranteed to start in a given location until the programmers or administrators provide additional configuration effort.

An example of specifying a policy is provided later in this introduction. Also, through this demonstration, each time the cluster is started, the partition specific member locations at start may vary from these instructions.

2.4.2 Launching a client application

To test the server side, launch the J2EE launchclient application.

Type the following in the command window:

launchclient c:\was\base51\installedApps\Cell\WPFKeyBasedPartitionSample.ear -CCproviderURL=corbaloc::<host>:<port>

<host> - hostname where the application server runs
<port> - RMI Connector Port (see application server logs\SystemOut.log (detail below)

The result should look similar to the following:

```
IBM WebSphere Application Server, Release 5.1
J2EE Application Client Tool
Copyright IBM Corp., 1997-2003
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class com.ibm.websphere.wpf.client.WPFKeyBasedPartitionClient
Create Partitions from PK000001 to PK000010
1st call: PK000001->partiton=PK000001,server=NodeA/cluster_member_1
2nd call: PK000001->partiton=PK000001,server=NodeA/cluster_member_1
3rd call: PK000001->partiton=PK000001,server=NodeA/cluster_member_1
1st call: PK000002->partiton=PK000002,server=NodeA/cluster_member_3
2nd call: PK000002->partiton=PK000002,server=NodeA/cluster_member_3
3rd call: PK000002->partiton=PK000002,server=NodeA/cluster_member_3
1st call: PK000003->partiton=PK000003,server=NodeA/cluster_member_3
2nd call: PK000003->partiton=PK000003,server=NodeA/cluster_member_3
3rd call: PK000003->partiton=PK000003,server=NodeA/cluster_member_3
1st call: PK000004->partiton=PK000004,server=NodeA/cluster_member_3
2nd call: PK000004->partiton=PK000004,server=NodeA/cluster_member_3
3rd call: PK000004->partiton=PK000004,server=NodeA/cluster_member_3
1st call: PK000005->partiton=PK000005,server=NodeA/cluster_member_1
2nd call: PK000005->partiton=PK000005,server=NodeA/cluster_member_1
3rd call: PK000005->partiton=PK000005,server=NodeA/cluster_member_1
1st call: PK000006->partiton=PK000006,server=NodeA/cluster_member_1
2nd call: PK000006->partiton=PK000006,server=NodeA/cluster_member_1
3rd call: PK000006->partiton=PK000006,server=NodeA/cluster_member_1
1st call: PK000007->partiton=PK000007,server=NodeA/cluster_member_1
2nd call: PK000007->partiton=PK000007,server=NodeA/cluster_member_1
3rd call: PK000007->partiton=PK000007,server=NodeA/cluster_member_1
1st call: PK000008->partiton=PK000008,server=NodeA/cluster_member_3
2nd call: PK000008->partiton=PK000008,server=NodeA/cluster_member_3
3rd call: PK000008->partiton=PK000008,server=NodeA/cluster_member_3
1st call: PK000009->partiton=PK000009,server=NodeA/cluster_member_1
2nd call: PK000009->partiton=PK000009,server=NodeA/cluster_member_1
3rd call: PK000009->partiton=PK000009,server=NodeA/cluster_member_1
1st call: PK000010->partiton=PK000010,server=NodeA/cluster_member_3
2nd call: PK000010->partiton=PK000010,server=NodeA/cluster_member_3
3rd call: PK000010->partiton=PK000010,server=NodeA/cluster_member_3
```

Compare the results to the **wpfadmin listActive** command executed in previous section, the application server where the request executed is where it had been activated.

Warning: if the particular partition targeted method call is routed to a different partition, the application was probably redeployed either during the application installation or prior to you trying this ear. See the wpfstubutil command in the programming guide for instructions to repair the example ear.

Common problems when running a launchclient command:

Launch Client cannot find the ear:

- Either cd to the <Base Application server Home>\installedApps\<CELL> directory (where the ear is deployed) and run, or
- With launch client, run from the <Base Application server>\bin and specify full path to the EAR on the command line

Cannot Resolve the JNDI Name:

- In the directory <Base Application server Home>\bin, issue a dumpnamespace –port <portnumber>
  - The port number can be found by looking in the application server SystemOut.log, e.g. the logs\SystemOut.log file.
  - find an entry similar to:
  - [9/21/04 7:48:21:231 CDT] 7191d4f0 RMIConnectorC A ADMC0026I: RMI Connector available at port **9812**
- To verify the ejb reference is in the name space and valid to reference, the dumpname space command output should appear similar to the following:
  - **dumpnamespace -port 9812**
  -
  - Getting the initial context
  - Getting the starting context
  -
  - ==
  - Name Space Dump
  - Provider URL: corbaloc:iiop:localhost:9812
  - Context factory: com.ibm.websphere.naming.WsnInitialContextFactory
  - Requested root context: cell
  - Starting context: (top)=Cell
  - Formatting rules: jndi
  - Time of dump: Tue Sep 21 09:27:33 CDT 2004
  - ==
  -
  -
  - ==
  - Beginning of Name Space Dump
  - ==
  -
  - 1 (top)
  - 2 (top)/clusters javax.naming.Context
  - 3 (top)/clusters/cluster javax.naming.Context
  - 4 (top)/clusters/cluster/ejb javax.naming.Context
  - 5 (top)/clusters/cluster/ejb/com javax.naming.Context
  - 6 (top)/clusters/cluster/ejb/com/ibm javax.naming.Context
  - 7 (top)/clusters/cluster/ejb/com/ibm/websphere javax.naming.Context
  - 8 (top)/clusters/cluster/ejb/com/ibm/websphere/wpf javax.naming.Context
  - 9 (top)/**clusters/cluster/ejb/com/ibm/websphere/wpf/WPFKeyBasedPartitionHome**
  - 9 com.ibm.websphere.wpf.ejb._WPFKeyBasedPartitionHome_Stub
  - … remainder of output…

2.4.3 Balancing the Partitions

In general a customer can implement any balancing algorithm they would like via the JMX interfaces provided. In addition, the wpfadmin command provides two basic options, a simple balancer and one that balances upon the certain performance monitoring attributes (used only when performance monitoring is enabled). For this case, we will simply balancing the partitions in a round robin format using the basic function.

Issue the command wpfadmin balance, and the output should appear similar to:

WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector; The type of process is: DeploymentManager

WPFC0054I: Partition PK000010 has moved from Server Cell\NodeA\cluster_member_3 to Server Cell\NodeA\cluster_member_1
WPFC0054I: Partition PK000009 has moved from Server Cell\NodeA\cluster_member_1 to Server Cell\NodeA\cluster_member_1
WPFC0054I: Partition PK000008 has moved from Server Cell\NodeA\cluster_member_3 to Server Cell\NodeA\cluster_member_1
WPFC0054I: Partition PK000007 has moved from Server Cell\NodeA\cluster_member_1 to Server Cell\NodeA\cluster_member_2
WPFC0054I: Partition PK000006 has moved from Server Cell\NodeA\cluster_member_1 to Server Cell\NodeA\cluster_member_2
WPFC0054I: Partition PK000005 has moved from Server Cell\NodeA\cluster_member_1 to Server Cell\NodeA\cluster_member_2
WPFC0054I: Partition PK000004 has moved from Server Cell\NodeA\cluster_member_3 to Server Cell\NodeA\cluster_member_3
WPFC0054I: Partition PK000003 has moved from Server Cell\NodeA\cluster_member_3 to Server Cell\NodeA\cluster_member_3
WPFC0054I: Partition PK000002 has moved from Server Cell\NodeA\cluster_member_3 to Server Cell\NodeA\cluster_member_3
WPFC0054I: Partition PK000001 has moved from Server Cell\NodeA\cluster_member_1 to Server Cell\NodeA\cluster_member_1

2.4.4 Post Balance - Launching a client application again

Then, you can run the client again, and the output should reflect the balance operation:

IBM WebSphere Application Server, Release 5.1
J2EE Application Client Tool
Copyright IBM Corp., 1997-2003
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class com.ibm.websphere.wpf.client.WPFKeyBasedPartitionClient
Create Partitions from PK000001 to PK000010
1st call: PK000001->partiton=PK000001,server=NodeA/cluster_member_1
2nd call: PK000001->partiton=PK000001,server=NodeA/cluster_member_1
3rd call: PK000001->partiton=PK000001,server=NodeA/cluster_member_1
1st call: PK000002->partiton=PK000002,server=NodeA/cluster_member_3
2nd call: PK000002->partiton=PK000002,server=NodeA/cluster_member_3
3rd call: PK000002->partiton=PK000002,server=NodeA/cluster_member_3
1st call: PK000003->partiton=PK000003,server=NodeA/cluster_member_3
2nd call: PK000003->partiton=PK000003,server=NodeA/cluster_member_3
3rd call: PK000003->partiton=PK000003,server=NodeA/cluster_member_3
1st call: PK000004->partiton=PK000004,server=NodeA/cluster_member_3
2nd call: PK000004->partiton=PK000004,server=NodeA/cluster_member_3
3rd call: PK000004->partiton=PK000004,server=NodeA/cluster_member_3
1st call: PK000005->partiton=PK000005,server=NodeA/cluster_member_2
2nd call: PK000005->partiton=PK000005,server=NodeA/cluster_member_2
3rd call: PK000005->partiton=PK000005,server=NodeA/cluster_member_2
1st call: PK000006->partiton=PK000006,server=NodeA/cluster_member_2
2nd call: PK000006->partiton=PK000006,server=NodeA/cluster_member_2
3rd call: PK000006->partiton=PK000006,server=NodeA/cluster_member_2
1st call: PK000007->partiton=PK000007,server=NodeA/cluster_member_2
2nd call: PK000007->partiton=PK000007,server=NodeA/cluster_member_2
3rd call: PK000007->partiton=PK000007,server=NodeA/cluster_member_2
1st call: PK000008->partiton=PK000008,server=NodeA/cluster_member_1
2nd call: PK000008->partiton=PK000008,server=NodeA/cluster_member_1
3rd call: PK000008->partiton=PK000008,server=NodeA/cluster_member_1
1st call: PK000009->partiton=PK000009,server=NodeA/cluster_member_1
2nd call: PK000009->partiton=PK000009,server=NodeA/cluster_member_1
3rd call: PK000009->partiton=PK000009,server=NodeA/cluster_member_1
1st call: PK000010->partiton=PK000010,server=NodeA/cluster_member_1
2nd call: PK000010->partiton=PK000010,server=NodeA/cluster_member_1
3rd call: PK000010->partiton=PK000010,server=NodeA/cluster_member_1

Notice, the client calls have been dispatched to partitions that have been balanced to application sever cluster_member2.

Another useful wpfadmin command is wpfadmin countActivePartitionsOnSevers:

wpfadmin countActivePartitionsOnSevers

The result provides a count of the active partitions per cluster member:
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector; The type of process is: DeploymentManager
WPFC0051I: Server Cell\NodeA\cluster_member_3: 3
WPFC0051I: Server Cell\NodeA\cluster_member_2: 3
WPFC0051I: Server Cell\NodeA\cluster_member_1: 4

2.4.5 Adding a Partition Dynamically

One of the more innovative features is to be able to add and remove partitions dynamically. The following command will allow a demonstration of this function:

```
C:\was\base51\bin>launchclient c:\was\base51\installedApps\Cell\WPFKeyBasedPartitionSample.ear -CCproviderURL=corbaloc::
localhost:9813 -addPartition PKMyNewOne
```

The result will be similar to:

```
IBM WebSphere Application Server, Release 5.1
J2EE Application Client Tool
Copyright IBM Corp., 1997-2003
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class com.ibm.websphere.wpf.client.WPFKeyBasedPartitionClient
Adding partition PKMyNewOne
Partitions added. Since there is a delay when a partition activates, to
run the client against the newly added partition, run launchClient again
without the -addPartition option
```

The application must be programmed to offer this function, this is not a feature wpfadmin offers in a generic sense. For programmers, they can consult the sample application to support dynamic addition and removal of partitions.

The warning above basically mentions adding and activating the partitions to the runtime view may not happen immediately (will be generally very fast), so the next command may not reflect it immediately in a very large cluster configuration.

The output of the listActive would look would appear similar to the following:

wsadmin -lang jython -f wpfadmin.pty listActive

```
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is: DeploymentMana
ger

WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PKMyNewOne: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000010: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000009: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000008: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000007: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000006: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000005: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000004: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000003: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000002: Server Cell\NodeA\cluster_member_3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000001: Server Cell\NodeA\cluster_member_3
```

In the command output, please notice that the partition “PKMyNewOne” now exists. At this point, the partition can receive transactions if you run the demonstration client without parameters.

2.4.6 Monitoring Transaction Performance Statistics

WPF provides the WPF Performance Monitoring Facility that provides a command line and graphical visualized snapshot of the current operational statistics WPF enabled applications are generating. In general, a WPF enabled application must execute the reportTransactionComplete(…) for this service to provide results. In addition, the performance monitoring is disabled by default as it does take system resources to track and publish the results. In high performance scenarios and when resources are tight, it is suggested to monitor the results in a sporadic manner versus longer term active monitoring.

The current sample application will be used to exercise this function.

2.4.6.1 Enabling Performance Monitoring

In general, the cluster should be stopped prior to starting this example. Stop the cluster via the Administrative console.

The following commands will enable the PMI performance monitoring infrastructure, allowing WPF Performance Monitoring extentions to provide transaction status.

Open a new command line interface, leaving the launchclient window used previously to exercise the server after the performance monitoring is initialized in this section.

Perform all the remaining steps in this section in the new window.

Once the new window is open, cd to the Deployment Manager's home directory. In addition, then change directory to the bin directory, as this exercise will need to execute the wpfadmin script.

This will enable PMI for the cluster. To monitor WPF, the H PMI level is required.

wpfadmin enableWPFPMI H --c cluster

```
wsadmin -lang jython -f wpfadmin.pty enableWPFPMI H --c cluster
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is: DeploymentMana
ger
WPFC0065I: Cluster set to cluster
WPFC0043I: The wpfModule of PMI is enabled for cluster cluster and set to level H.
```

This cluster should now be started, please use the admin console to restart the cluster "cluster".

At this point, you can track a particular transaction count for a specific bean instance. The following command will track the transaction count the top 10 instances every 30 seconds.

wpfadmin subscribeWPFPMI cumulative TransactionCount WPFKeyBasedPartitionSample WPFKeyBasedPartition 10 30000 --c cluster

```
wsadmin -lang jython -f wpfadmin.pty subscribeWPFPMI cumulative TransactionCount WPFKeyBasedPartitionSam
ple WPFKeyBasedPartition 10 30000 --c cluster
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is: DeploymentMana
ger
WPFC0065I: Cluster set to cluster
WPFC0040I: WPF PMI has been subscribed with options range=cumulative, type=TransactionCount, application name=WPFKeyBase
dPartitionSample, ejb name=WPFKeyBasedPartition, partition count=10, interval=30000
WPFC0041I: Your client id is 1. Use this in future wpfadmin PMI calls.
```

Note the client id, it is the reference id used to monitor this subscription. At this point, you have the ability to actually begin active monitoring.

Before you can monitor, some existing transactions have to be completed. If not, the client application will report an error. To ensure we have a valid transaction, run the launch client application as done earlier. Without ensuring some transactions are completed, you would get this error:

wpfadmin getTransactionCount --id 1 --top 15

```
C:\was\nd51\bin>wsadmin -lang jython -f wpfadmin.pty getTransactionCount --id 1 --top 15
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is: DeploymentMana
ger
WPFC0065I: Id set to 1
WPFC0065I: Top interval set to 15

PartitionName       TransactionCount  TotalResponseTime  MinimumTime  MaximumTime
WPFC0045I: No statistics are available! Please wait and try again.
```

launchclient c:\was\base51\installedApps\Cell\WPFKeyBasedPartitionSample.ear -CCproviderURL=corbaloc::<host>:<port>

Turn on the WPF Monitoring Application, use the following command.

wpfadmin getTransactionCount --id 1 --top 15

wsadmin -lang jython -f wpfadmin.pty getTransactionCount --id 1 --top 15

WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector; The type of process is: DeploymentMana
ger
WPFC0065I: Id set to 1
WPFC0065I: Top interval set to 15

PartitionName	TransactionCount	TotalResponseTime	MinimumTime	MaximumTime
PK000010	3	1418	207	639
PK000009	3	1456	32	752
PK000008	3	2254	586	907
PK000007	3	1161	149	817
PK000006	3	2130	663	760
PK000005	3	1711	77	908
PK000004	3	1040	109	784
PK000003	3	703	22	400
PK000002	3	1138	125	684
PK000001	3	926	31	670

==========

PartitionName	TransactionCount	TotalResponseTime	MinimumTime	MaximumTime
PK000010	3	1418	207	639
PK000009	3	1456	32	752
PK000008	3	2254	586	907
PK000007	3	1161	149	817
PK000006	3	2130	663	760
PK000005	3	1711	77	908
PK000004	3	1040	109	784
PK000003	3	703	22	400
PK000002	3	1138	125	684
PK000001	3	926	31	670

This option will continue tracking as long as the user does not exit with a Control-C in the command window. If a single snap shot is required rather than a continuous display:

wpfadmin getTransactionCount --id 1

wsadmin -lang jython -f wpfadmin.pty getTransactionCount --id 1
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Id set to 1

PartitionName	TransactionCount	TotalResponseTime	MinimumTime	MaximumTime
PK000010	15	7904	99	894
PK000009	15	6964	32	866
PK000008	15	6880	3	998
PK000007	15	6766	64	946
PK000006	15	9253	210	993
PK000005	15	7897	21	971
PK000004	15	7191	109	994
PK000003	15	7076	6	931
PK000002	15	7210	84	789
PK000001	15	6375	31	949

For those willing to experiment further, rather than using the cumulative option, the “active” option can be used. And the launchclient command will take a –loop xxxxx parameter. After the PMI tracking is in place, the –loop option will generate transactions against each partition endpoint in a looping manner, and the active transaction will reflect how many new transaction over a certain instance of time.

For example:

wpfadmin subscribeWPFPMI active TransactionCount WPFKeyBasedPartitionSample WPFKeyBasedPartition 10
30000 --c cluster

And the launch client command to loop:

launchclient c:\was\base51\installedApps\Cell\WPFKeyBasedPartitionSample.ear -CCproviderURL=corbaloc::<host>:<port> -loop 10000

2.4.6.2 Monitor WPF Partitions via Deployment Manager

For this section, the Microsoft Internet Explorer will be required. An Adobe SVG plug-in is used. Launch the browser if one is not already connected to the Deployment Manager to http://<hostnmae>:9090/admin.

Ensure the cluster is started, and in a separate window, launch the looping launchclient command described above. This will populate the transactions for view within the Deployment Manager.

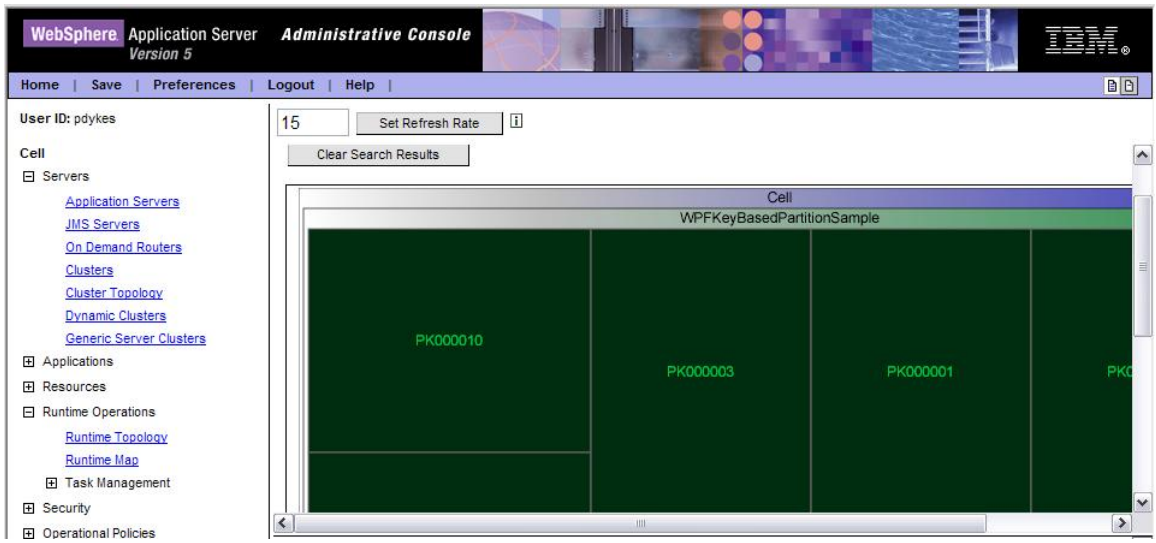
In the console, select:

- Runtime Operations
  - Runtime Map

For this example, we will monitor more aggressively:

- Preference
  - Change Set Refresh Rate to 15
  - Set Layer 1 option to Application
  - Set Layer 2 option to Partition
  - Set Layer 2 Displayed Entity field to WPFKeyBasedExample
  - Set static type to cumulative
  - Select Apply

The screen should look something like:



If you place your mouse pointer over the partition, it will show relevant facts, and as more transaction are processed, the refresh rate should update from the runtime state for the specific partition.

2.4.6.3 Managing Policies Example Overview

Policies are critical to understand when doing partitioned applications, both for programming and effective administration. As with other aspects of this step-by-step tutorial, there are detailed explanations of HA Manager policies in this guide. The goal of this section is to not only give a brief explanation of policies, but as important provide an example of setting a policy that improves the default behavior to benefit the approach to manage starting partitions.

By default, the starting of partitions is non-deterministic. The order and timeframe in which the servers start will dictate how partitions are created within HA Manager and how they are activated. A common need, especially with solutions that require many partitions, is the need to even out the partition set over the servers that make up the cluster. This example is not an optimal solution, but does provide a basic introduction to the concept of policies and creating a new one to modify the default behavior partition behavior.

To avoid one point of common confusion, it is important to distinguish the difference between a balance or move operation provided and applying a startup policy as this example offers. The balance or move operations do result in a partition outage, e.g. the partitionUnloadEvent(…) is called on the partition in these cases. While the programmer should account for this in these sorts of applications, if a policy is used to control the startup versus relying solely on the dynamic balance operations, the effort required to first activate a partition, unload it, and then reactivate on another member can be avoided.

In summary, an HA Manager policy describes a set of coregroup properties and a specific match criteria. Based upon the number of match criteria, the policy will be applied to a set of partitions. The number of "matches" will determine which policy applies. If two policies result in matches with the same number of criteria, HA Manager will generate an error, as it does not know which policy to apply to the HA Managed Group. Keep in mind, each partition, is actually an HA Manager group. In the case of a cluster scoped partition, the partition is joined on each node in the cluster where the Partition J2EE Application is installed, but "active" in only one member that will receive the requests.

For example, the default policies, located in the configuration directory's coregroup.xml file, look like:

```
 <policies xmi:type="coregroup:OneOfNPolicy" xmi:id="OneOfNPolicy_1097676485106" name="WPF Cluster Scoped Partition Policy"
description="Default WPF Cluster Scoped Partition Policy" policyFactory="com.ibm.ws.hamanager.coordinator.policy.impl.OneOfNPolicyFactory"
isAlivePeriodSec="-1" quorumEnabled="true" failback="false" preferredOnly="false">
   <MatchCriteria xmi:id="MatchCriteria_1097676485116" name="-gt" value="-p" description="Default WPF Match Criterion"/>
   <MatchCriteria xmi:id="MatchCriteria_1097676485117" name="-ps" value="-c" description="WPF Cluster Scope Match Criterion"/>
 </policies>


 <policies xmi:type="coregroup:OneOfNPolicy" xmi:id="OneOfNPolicy_1097676485126" name="WPF Node Scoped Partition Policy"
description="Default WPF Node Scoped Partition Policy" policyFactory="com.ibm.ws.hamanager.coordinator.policy.impl.OneOfNPolicyFactory"
isAlivePeriodSec="-1" quorumEnabled="false" failback="false" preferredOnly="false">
   <MatchCriteria xmi:id="MatchCriteria_1097676485166" name="-gt" value="-p" description="Default WPF MatchCriterion"/>
   <MatchCriteria xmi:id="MatchCriteria_1097676485167" name="-ps" value="-n" description="WPF Node Scope Match Criterion"/>
 </policies>
```

In the two stanzas above, one describes WPF Cluster Scoped Partitions policy and the other is the WPF Node Scoped Partitions policy. These policies support the default partition types WPF provides. The particular details are described in more detail in the HA Manager policies section. The key attributes to note for this example are the basic properties for each policy (quorumEnabled, isAlivePeriodSec, etc…) and the match criteria.

For the match criteria (in bold), the –gt represents the group type, which is –p (for partition). The partition scope attribute (-ps) differs for each type, e.g. cluster scoped attribute is –c and nodescoped is –n). These name and value types are reserved for WPF. However, the user can extend these and for extend policy management (other WPF attributes that are IBM controlled are listed in the policy section). When a partition is created, WPF provides some default properties that can be used by programmers and administrators to control. The programming guide describes the IBM set attributes. In addition, users can add their own depending on the manner in which partitions are created.

In this example, we will use an existing property, the partition name specifically, to match yet a third match criteria, and thus override the two default polices that match two match criteria. A new policy will be created below to accomplish this. A policy consists of properties that control the behavior of the HA Managed Group, and the match criteria. For this example, the WPF default attribute name "-pn" is set and the value is the partition name. In the samples above, the partition name would be PK000001 for example. If a policy is created, that not only specifies the match criteria for HA Manager Group type (in our case for partitions, that is "-gt"), cluster partition scope related to activation ("-ps"), and the partition name ("-pn") is defined, the policy will override the default.

Thus, for this example, we will create a new policy with 3 match criteria, effectively overriding the default attributes to facilitate a common requirement, e.g. to balance the partitions at startup. To accomplish this, we will extend the default cluster scoped partition policy, and define a preferred server startup choice that results in the HA Manager assigning a partition to one of two servers. Servers that are for preferred startup are an example of a coregroup property as compared to a match criteria. Once the maximum match criteria is established during runtime, the policy attributes will be applied.

In this particular case, a primary and backup server will be denoted. If primary does not start, or starts later than the secondary, the partition will start on the secondary. The goal in this case is to at least start the partition one 1 of 2 servers, and these two servers will be alternated for each partition to achieve a balance startup.

In addition, as we are not trying to change the Application Server functionality, just the startup location, the other coregroup properties are set the same as already denoted in the coregroup.xml above.

With that background in place, it is time to implement the examples.

First, either acquire the example policies from the WPF web site, or type the following in to a text file. For this example, name the text file “policyPK1_startup.properties” and place in the deployment manager \bin directory.

CoreGroupName = DefaultCoreGroup
PolicyType = OneOfNPolicy
PolicyName = PK1StartupPolicy
PolicyDescription = WPF Cluster Scoped Partition Policy Extended PK000001 Start
IsAlivePeriodSec = -1
QuorumEnabled = true
NumOfMatchCriteria = 3
Name_0 = -gt
Value_0 = -p
Name_1 = -ps
Value_1 = -c
Name_2 = -pn
Value_2 = PK000001
Failback = true
PreferredOnly = true
NumOfPolicyServers = 2
NodeName_0 = NodeA
ServerName_0 = cluster_member_1
NodeName_1 = NodeA
ServerName_1 = cluster_member_2

Note, if you opted to use different server names, node names, etc.. make the appropriate changes to suit your configuration. Notice the 3rd match criteria in bold, this will be loaded prior to the startup of the cluster.

2.4.7 Managing Policies Example

If the cluster is running, stop it at this point. Also ensure the Deployment Manger and NodeA’s node agent is running. Open a command shell, and change directory to the Deployment Manager \bin directory. Ensure the policyPK1_startup.properties created above is also in the Deployment Manager \bin directory.

Execute the following command to load the policy created above.

wpfadmin createpolicy policyPK1_startup.properties

The generated output should look something this, if not, please verify the properties file:

```
wsadmin -lang jython -f wpfadmin.pty createPolicy policyPK1_startup.properties
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is: DeploymentMana
ger
The policy PK1StartupPolicy has been created
```

The coregroup.xml in for the Deployment Manager now has the following policy:

```
  <policies xmi:type="coregroup:OneOfNPolicy" xmi:id="OneOfNPolicy_1097944892103" name="PK1StartupPolicy" description="WPF Cluster
Scoped Partition Policy Extended PK000001 Start" policyFactory="com.ibm.ws.hamanager.coordinator.policy.impl.OneOfNPolicyFactory"
isAlivePeriodSec="-1" quorumEnabled="true" failback="true" preferredOnly="true" preferredServers="CoreGroupServer_1097678779756
CoreGroupServer_1097678774418">
    <MatchCriteria xmi:id="MatchCriteria_1097944898452" name="-gt" value="-p" description=",None"/>
    <MatchCriteria xmi:id="MatchCriteria_1097944898532" name="-ps" value="-c" description=",None"/>
    <MatchCriteria xmi:id="MatchCriteria_1097944898582" name="-pn" value="PK000001" description=",None"/>
  </policies>
```

Normally, if the node agent is running, the coregroup file will be propagated (can verify by looking at the coregroup.xml file and finding the entry above). Otherwise, the node should be synchronized so the policy update in the coregroup can be propagated to the nodes in the cluster. To do so, with the nodeagent disabled, the syncNode command can be used.

Use the administrative console, and start the cluster.

In this case, the partitions startup typically on cluster_member_2, thus PK000001 should be the only one on cluster_member_1, Your configuration may differ. To verify after cluster startup:

wpfadmin listActive

```
wsadmin -lang jython -f wpfadmin.pty listActive
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is: DeploymentMana
ger
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000010: Server Cell\NodeA\cluster_member_2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000009: Server Cell\NodeA\cluster_member_2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000008: Server Cell\NodeA\cluster_member_2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000007: Server Cell\NodeA\cluster_member_2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000006: Server Cell\NodeA\cluster_member_2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000005: Server Cell\NodeA\cluster_member_2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000004: Server Cell\NodeA\cluster_member_2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000003: Server Cell\NodeA\cluster_member_2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000002: Server Cell\NodeA\cluster_member_2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000001: Server Cell\NodeA\cluster_member_1
```

Notice, all the partitions started on cluster_member_2 rather than cluster_member_1.

To update the policy, we could simply update the –pn option, and adjust so applies to PK000002 instead or a second policy could be created. An additional operation could be simply stop cluster member 1, e.g. “stopserver cluster_member_1”. In this case, since “Failback” is set, HA Manager will try to activate on cluster_member_2.

The policy can be updated using the following command:

```
wpfadmin updatePolicy “PK1StartupPolicy” -failback true -preferredOnly false -preferredServers NodeA/cluster_member_2,NodeA/cluster_member_1
```

The command output will appear similar to:

```
C:\was\nd51\bin>wsadmin -lang jython -f wpfadmin.pty updatePolicy PK1StartupPolicy -failback true -preferredOnly false -preferredServers
NodeA/cluster_member_2,NodeA/cluster_member_1
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is: DeploymentManager
The policy PK1StartupPolicy has been updated
```

To delete the policy:

```
wpfadmin deletePolicy PK1StartupPolicy
```

```
wsadmin -lang jython -f wpfadmin.pty deletePolicy PK1StartupPolicy
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is: DeploymentMana
ger
The policy PK1StartupPolicy has been deleted
```

If the cluster is restarted, you noticed you have reverted back to the default startup partition placement algorithms.

This concludes this example sequence. Keep in mind this is not the optimal way to modify the selected startup servers. Please refer to the HA Manager Policy section for more information.

2.5 Example Summary

In summary, this sequence of steps is used to install a Partitioned J2EE Application. The other example applications can be used in a similar fashion. Note the readme.txt included within each J2EE ear file for additional details. The following sections provide more overview on some the topics illustrated in this example sequence.

3 Partitioning Introduction

This section explains the terminology used in the remainder of this text, and provides basic information programmers and administrators should be aware of. More specific information for each audience is provided in the sections specific to each discipline.

3.1 What is a Partition?

A “Partition” is simply a uniquely addressable end point within the cluster. A partition is not a server (JVM). A partition has a life cycle, and is managed by the High Availability Manager (HA Manager). A partition is created dynamically during J2EE application initialization at startup and then available for client applications to use as a target end point when in the “Active” state. To become “Active”, HA Manager will move the partition from a “Idle” state to an “Active” state via a management transition. The state transitions can result from JMX management commands or system events such as an application server starting or stopping. The ***wpfadmin*** command provides the administrator a list of active partitions.

A partition can be activated on any cluster member in the cluster. The HA Manager guarantees there is a single instance of an active partition in the cluster at a given time within the cluster for Cluster scoped partitions (there is another type called Node Scoped discussed later in this document). The HA Manager allows a partition to be moved from one member in the cluster to another via a ***wpfadmin*** command. When moving a partition, the partition will change states on each cluster member. For example it will be deactivated on the original cluster member, and it will be activated on the new target cluster member.

3.1.1 Partition Life Cycle

Partitions are by default highly available. A partition will only be hosted on a single cluster member at a time. They are made highly available using the HA Manager component. If a cluster member fails either because of a JVM shutdown, a JVM panic or the box hosting the cluster member fails/gets powered down then the HA Manager moves all partitions which were running on the failed cluster members to the surviving cluster members.

The following diagram illustrates the state transitions a partition will encounter as member of an HA Group.

- IDLE:
Partition is currently deactivated and waiting for an activation command, a Partition is in this state at the Bean start during server startup after the createPartitionDefinition(…) until the PartitionLoadEvent(…) API is called by the HA Manager Coordinator.
- ACTIVATING:
HA Manager is attempting to activate the partition but has not been acknowledged yet, this in effect during the PSSB PartitionLoadEvent(…) method execution.
- ACTIVATED:
This means the partition current activate and working. The PartitionLoadEvent(…) event in the application has completed. This state is implies that the HA Manager has picked a target application server for this partition, it is active and ready to process methods, and the Work Load Management layer has enabled for client to route requests to the particular application server associated with the specific partition.
- DISABLED:
This means the partition whilst still part of the group cannot become an active member. This is a failure state.
- DEACTIVATING:
The partition received a deactivate signal and is still deactivating. This occurs while the PartitionUnloadEvent(…) method executes in the PSSB for this partition. The Partition then returns to IDLE and the HA Manager signals the activate coordinator managing that group and provide it a copy of the current local state.

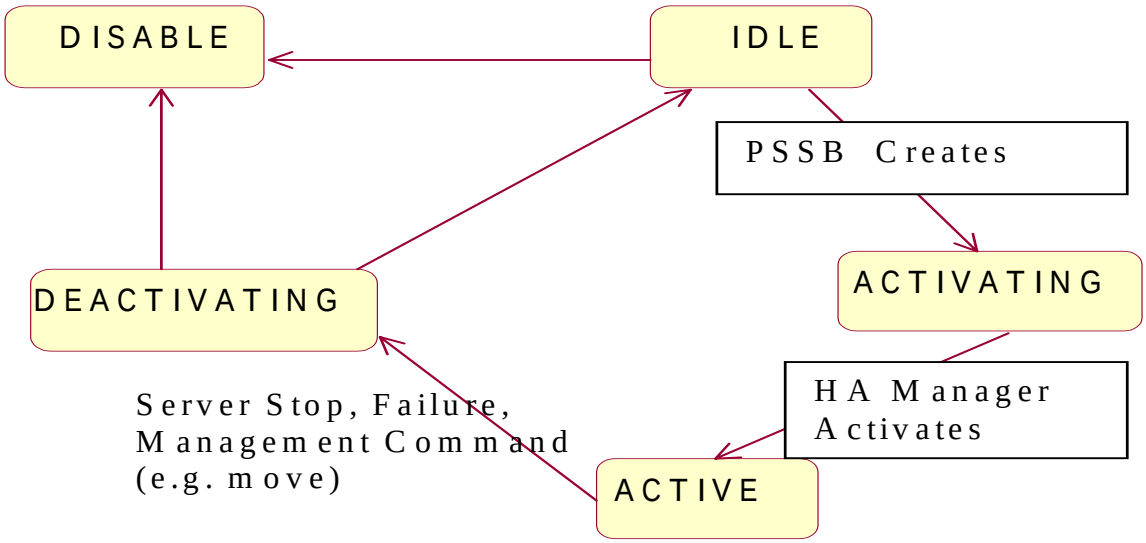


Figure 1 Group member valid state transitions

3.1.2 Partition Creation

A partition is created when the J2EE Application starts or subsequently after the start is complete via the PartitionManager interface. During the application startup, via the PartitionManager interface in the WPF framework, the application programmer implements an API that dynamically creates the partitions by name. The partitions created can be either hard coded strings, e.g. “P001” or could be strings from the result of an external input such as a database. All the same partitions should be created on application servers in the cluster, otherwise that partition will not be activated.

A more complete example will be placed in the programming guide, but creating a partition in the most straightforward manner is implemented with the following API in the programmer’s PSSB’s `PartitionDefinition[] getPartitions()` method:

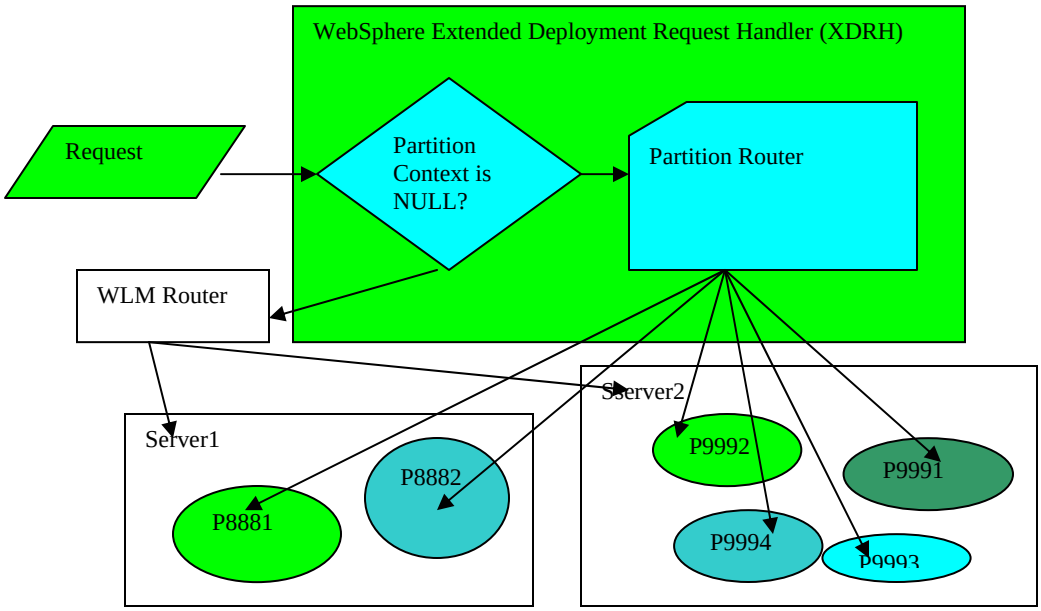
PartitionDefinition p[x] = createPartitionDefinition(String partitionName)

The programmer will typically implement a loop and create partitions using a string. The name itself can be any typical character string. For example, 2 partitions could be created with the above API and named “P001” and “P002”. Each partition will be subsequently activated on a single member in the cluster when the HA Manager detects and “runs” the policy. The partition name, e.g. “P001”, must be unique within the entire cluster.

If this API does not execute consistently on each application server in the cluster, the partition cannot be moved to or made active on a given application server. In terms of debugging this problem, please refer to the Debugging Tips section in the management chapter.

3.1.3 IIOP Routing to a Partition

Client requests submitted using the Orb (via IIOP) can be routed to a particular partition in a cluster. When the client makes a remote request to the routable session bean, the client stubs determine which partition the request is intended for. The current location of the partition is determined using the WebSphere workload management (WLM) framework. The request is then sent directly to the cluster member currently hosting the partition. If more than one cluster member is hosting the partition then the requests are dispatched in round robin fashion over the set of candidates.



The above diagram shows how the request flow is processed in WPF-enabled WebSphere application servers. With Extended Deployment is installed, new router called WPFWLM router or partition router is created. All requests will go to WPFWLM router or partition router first no matter the requests need partition routing or not. If partition router decides that the request doesn't need to do partition routing, partition router will forward this request to normal WLM router. If partition router decides that this request requires partition routing, partition router will use partition routing mechanism to route this request directly to right partition as denoted as small box inside big oven. There may have thousands of partition per server.

Application writers can control their application's routing behaviors by writing <EJBName>_PartitionKey.java class, this class is used to signal whether partition router or normal non-partition WLM router is used to do routing. There is <EJBName>_PartitionKey.java for each EJB remote interface you want to make it partition routable. Inside <EJBName>_PartitionKey.java, you need to have a static method for each remote method that you want to make it partition routable. The static method returns a not null string to signal partition routable to partition router. Without such static method or static method returning null, partition router will handle this request as non-partition routable, and forward this request to normal non-partition WLM router. More detail related to the <EJBName>_PartitionKey class is provided in the programming section of this text.

3.2 What is a Partitioned Stateless Session Bean?

A Partitioned Stateless Session Bean is a stateless session bean that implements the PartitionHandlerLocal interface, and utilizes the PartitionManager to create partitions from the WPF framework. Thus, the runtime characteristics are the same as a normal stateless session bean from a runtime perspective.

The special features of this bean type include the following:

- While the application started in the EJB Container, the bean is analyzed, and if a PSSB the PartitionHandlerLocal interface methods are called, which when implemented with the PartitionManager API will submit requests to the HA Manager Coordinator to create and activate partitions based upon the current policy.
- During the execution of methods from the client to the server implementation of the PSSB, each client method invocation will be intercepted and processed to determine which partition this method should be driven against on the server infrastructure. For example, assume a PSSB named TestBean is being demonstrated, and this bean has a remote interface method called ping(String partition). When the client executes the TestBean.ping(partition) method, the TestBean_PartitionKey.ping(partition) (a class the programmer will create) will process all the method attributes and return a single string to advice the WLM subsystem which partition endpoint the method should be directed towards.

- The PSSB has several methods to handle the state changes the partition instances may encounter during normal execution. For example, there is PartitionLoadEvent(Partition) and PartitionUnLoadEvent(Partition) method that will be executed if the administrator uses JMX or wpfadmin command to move a partition endpoint from server to another server. In this case, if a Partition P001 is associated with application server appsvr1, and the user execute a wpfadmin move operation that changes the P001 partition from appsvr1 to appsvrN in the cluster these bean call back methods are invoked. The PSSB on appsvr1 would receive a deactivate callback from HA Manager, and in turn the WPF runtime would call the PartitionUnLoadEvent(…) method. After this method successfully completes, on appsvrN HA Manager will drive an activate related call back method, and the WPF runtime will execute a PartitionLoadEvent(…) on the PSSB.

In summary, a PSSB is a typical session bean that implements and calls methods from the WPF framework. These methods in many ways are similar to existing EJB call back method handling, but have function that is related to partitioning rather than normal EJB call back methods.

3.3 What is a Partitioned J2EE Application?

A partitioned J2EE application is a typical J2EE application with a single Partitioned Stateless Session Bean (PSSB). There are no limitations related to the J2EE application that includes the PSSB. For Extended Deployment 5.1, a single application ear file, can include several EJB modules as normal. However, only one EJB module can contain the single PSSB. In addition, the partition names chosen during the PSSB Partition Manager initialization sequence must be unique within the cluster. Also, the application should be installed on nodes in the cluster.

For customers using web content, e.g. JSP, HTML, Servlets, they can also reference the PSSB normally via JNDI, and the EJB module will contain the PSSB. The servlet typically will execute a remote method of the EJB PSSB remote interface, which result in a routed call to the server with the appropriate target endpoint partition.

A single PSSB can have multiple partition sets created during the partition creation process. For example, if your organization has a solution that needs to map partitions to customer id to a given cluster member to take advantage of a partition database solution. In addition to mapping by the customer id, and acquiring the customer information, assume the customer request need to be mapped to a partition for further processing. To satisfy this need, the application will map the request to another partition scheme, in this, case two separate partitions schemes are required. The WPF framework provides both the programmatic support and management functionality to support more than one type of partition scheme in the same application.

The various types of programming strategies for PSSB include calling the bean directly, using a Partition Routable Bean and a Façade Session interface to the PSSB. Both approaches allow the PSSB to provide the underlying routable support, but allow the business interfaces to be externalized in another bean within the same application, same EJB Jar specifically.

3.4 What is a Partitioned HTTP Application?

As with a Partitioned J2EE Application with WPF, each HTTP WPF application must also contain a Partitioned Stateless Session Bean (PSSB). This bean interacts with a HttpPartitionManager interface make requests of HTTP Partitioning. An application may optionally contain one or more web modules that may also use the HttpPartitionManager. A Servlet (within a web module) may register for WPF events such as the loading and unloading of partitions by the WPF subsystem. . HTTP Partitioning, a single partition should only be active on a single application server within the cluster. Finally, in addition to the application.xml contained in the META-INF directory of an enterprise application archive (EAR), an application may also specify partition information in a partitions.xml file.

This application is particularly useful if the EJB Container and Servlet are collocated on the same application. Note that the EJB client and Web client will be directed to the same target endpoint partition on the application server. This implies that partitions used jointly by IIOP and HTTP may only be active on a single application server within the cluster. Please refer to the programming guide for more information.

3.5 Samples Overview

WebSphere Extended Deployment installation process installs 8 example Partitioned J2EE Applications. These are located in the <Deployment Manager Home>\installableApps directory. Most examples have two versions, a non-deployed example with the source included and a fully deployed and updated stub version (WPF requires a post ejbdeploy stub update step).

For those wishing to view the code, we encourage you to import each Partitioned J2EE Application into WSAD, and look over the Session Bean implementation. The Partitioned J2EE Applications with the same name but prefixed with a D_* are ready for installation. When installing:

- do not select to execute the deploy step during the installation
- the basic steps in the tutorial should be followed

As a reminder, every WPF Partitioned J2EE Application must be deployed normally after written. However, these application require an additional step. Each application must have follow-on processing to update the generated stubs for this version of the product. The utility to perform this task is “wpfstubutil”, and documented in the programming guide. If this tool is not used, the normal round robin session bean method execution across all available will occur.

3.5.1 Partition Examples

Sample	Description
WPFKeyBasedPartitionSample.ear	Uses a partition schemed based upon specific database keys. This example Partitioned J2EE application is used earlier in this document. Note: this client cannot be installed in the same cluster as the WPFFacadePartitionSample.ear, due to the current restriction that partitions names must be unique in the cluster.
WPFHashBasedPartitionSample.ear	Uses a mapping approach to map many incoming items to a set of partitions via hashing.
WPFHybridBasedPartitionSample.ear	Combines an integrated key based and hashed based example
WPFFacadePartitionSample.ear	This example uses a session bean façade, with the intent that the PSSB routing is all done on the server side. This is the most optimal way in terms of performance to use partition based routing, as all the routing work is done within the server infrastructure versus the client. Note: this client cannot be installed in the same cluster as the WPFKeyBasedPartitionSample.ear, due to the current restriction that partitions names must be unique in the cluster.
ProxyDSAccountSample.ear	Combines the WPF partition functionality with Datasource Proxy functionality.
httpwpfsample.ear	HTTP Partitioning example includes a sample Servlet that leverages the Servlet API.

The deployed versions have a D_ prefix, and are the ones that should be used by administrators to experiment. The programming staff can use those without the D_ prefix, as they have the source code included. In addition, both versions have a "readme" document that can be used to experiment with the sample.

4 Managing a WPF Environment

This section describes the management capabilities of the WPF framework within WebSphere Extended Deployment. The section will assume the user has some experience with the WPF example scenario above. This section begins with an HA Manager overview, provides general guidelines and then moves to more detailed explanation of the individual management functions

4.1 HA Manager

The high availability manager (HA Manager) is a new component for WebSphere 5.1. Extended Deployment provides the capability to manage HA groups of resources in a clustered environment. The HA Manager is configured using a policy mechanism allowing for precise control of its runtime behavior.

4.1.1 HA Manager Overview

For HTTP and EJB partitioning, the High Availability Manager (HA Manager) generally manages one or more highly available groups, specifically for WPF this correlates partitions to cluster members. The HA Manager manages highly available groups of application servers and partitions. As cluster members are stopped, started or fail, the HA Manager will monitor the current state, and based upon a given set of policy attributes adjust the state of the partitions as required. Consequently, the HA Manager provides the fundamental functionality to manage partitions. The HA Manager allows the creation of policies that allow the programmer and administrator to replace or augment the defaults provided. The HA Manager is a collection of technologies to manage distributed resources. The Core Group, Coordinator and Policy functions enable the key functions the HA Manager service provides.

Core Group

A core group is a set of servers (JVMs) that can be divided up into various high availability groups. In a run-time server environment each core group functions as an independent unit. A Java Virtual Machine (JVM) that is contained within a cell can be a member of one core group only. This JVM can be a node agent, an application server or a deployment manager. However, even though the deployment manager can belong to one core group only, it is still responsible for configuring all of the application servers within a cell, even if multiple core groups are defined for that cell. The core group configuration is used by the HA Manager to establish the members where WPF partitions will be allocated at activation.

Coordinator

The coordinator is the elected, or default, high availability manager from a runtime server perspective. The coordinator is responsible for tracking all of the members of a core group when members leave, join, or fail. In addition, the coordinator is not a single point of failure. In the event of a failure involving the coordinator, the preferred coordinator, or a default, picks up the high availability manager work, including the management of the core group. The default coordinator is sometimes referred to as the active coordinator. Additional coordinators can be used. These serve as backup coordinators and under heavy load additional coordinators are required to spread the workload.

Policy

A policy is used to designate core group members as part of a specific high availability group. A Coordinator relies on the currently active policy as each HA Managed event is detected for each core group member. Even though a policy is defined at the core group level, it does not apply to the core group.

A policy is basically a set of attributes describing how the HA Managed Group should behave, and MatchsetCriteria to define which groups should have a particular policy applied. In the example above, the PK1StartupPolicy was an example. As a reminder, this is how the PK1StartupPolicy appears in the coregroup.xml file:

```
<policies xmi:type="coregroup:OneOfNPolicy"
        xmi:id="OneOfNPolicy_1097944892103"
        name="PK1StartupPolicy"
```



```
        description="WPF Cluster Scoped Partition Policy Extended PK000001 Start"
        policyFactory="com.ibm.ws.hamanager.coordinator.policy.impl.OneOfNPolicyFactory"
        isAlivePeriodSec="-1"
        quorumEnabled="true"
        failback="true"
        preferredOnly="true"
        preferredServers="CoreGroupServer_1097678779756 CoreGroupServer_1097678774418">

    <MatchCriteria xmi:id="MatchCriteria_1097944898452" name="-gt" value="-p" description=",None"/>
    <MatchCriteria xmi:id="MatchCriteria_1097944898532" name="-ps" value="-c" description=",None"/>
    <MatchCriteria xmi:id="MatchCriteria_1097944898582" name="-pn" value="PK000001" description=",None"/>

 </policies>
```

In this case, several attributes are defined, and HA groups that apply must match three different match criteria. Each group is created with a set of group properties. When policies are applied, the policy with the most number of matching attributes will be applied to a given group.

Several different high availability groups can use the same policy, but all of the high availability groups to which it applies must be part of the same core group. A policy is established for a high availability group when that group is created. The following policies can be specified for a high availability group:

- All active policy: Under this policy, all of the group members are activated.
- M of N policy: Under this policy, *M* group members in the core group are activated. The number represented by *M* is defined as part of the policy details.
- No operation policy: Under this policy no group members are activated.
- One of N policy: Under this policy, only one group member in the core group is activated.
- Static policy: Under this policy, group members on all servers in the list are activated.

Thus, each set of partitions can have slightly different policies applied to them. For example, different partition groups can be treated differently for such attributes as preferred and failback server within the same cluster. Additional details regarding other policy attributes are described later in the management section.

The attributes particular to a policy describe how HA Manager should machine a set of partitions that map to a policy via the respective match criteria. In the example above, the attributes set include:

IsAlivePeriodSec – if the isAlive method should be called for an active HA Managed member, and if so, how many seconds between requests.
quorumEnabled - a boolean value describing if the HA Group should be managed with quorum detection or not
failback – a boolean describing if when a HA Group active member fails, and recovers, should the member be moved back to the server that is now recovered.
preferredOnly – should the HA Group active member be started on a specific cluster member, or any available at the time the partition is ready to be activated
preferredServers – is preferredOnly is enabled, then which servers should HA Manager target to active the member when the servers start and become operational.

The following sections describe more precisely the WPF partitioning implementation with respect to HA Manager concepts and techniques.

4.1.2 HA Managed Policy Applied to Partitioning

Partitions use a “One of N” policy with quorum enabled by default, which means partitions can only be activated when the majority of the possible cluster members are online or as considered in the case, in a state of “quorum”. Each Partitioned J2EE application creating a set of partitions (more detail for this in next section) creates an entire set of HA managed groups. Each “Partition” is actually an official HA Manager group, and can be managed separately from other groups. Thus, if the customer has a need to mix and match partition policies differently for each application, or individual partitions the capability exists. Additionally, when the specifics are discussed, each Partitioned J2EE application can actually sub classify portions of the overall partition set into subgroups, and manage them uniquely.

For example, assume a user is creating a stock trading application. They wish to have one Partitioned J2EE application that handles all stock types, but they wish to treat S&P 500 stocks differently due to the trade volume (load) characteristics. When the application servers start all partitions will be activated on the set of servers available at the

time quorum is established. At this time, the administrator can set a new policy and balance the load more effectively using the HA Manager policy infrastructure.

If this example, the administrator wishes to balance the partitions across the entire cluster fairly based on expected transaction volume. They can create a policy that will balance the S&P 500 partitions uniformly over all the existing cluster members, and then balance all the other stocks across the same set of cluster members similarly. This approach guarantees that the stocks transactions against the S&P500 companies will be balanced across the entire cluster, versus randomly balancing all stocks over the cluster members. If the partitions were managed as one grouping, as in this case, the result could be that some cluster members may have an inordinate number of high volume S&P 500 related partitions and thus many more transactions than other servers. In addition, some servers may have a large number of stocks that receive little to none in terms of transaction volume in a single day and are under utilized.

Other examples which utilize the HA Manager Policy support are to set preferred servers for specific partitions, predefine servers to be used for failover scenarios, define whether a partitions should be sent back to the original server once the server is back on-line. Many other options are available and described in the subsequent sections.

4.1.3 HA Manager Quorum Attribute

The default partition policy is quorum enabled, which is a policy attribute supported by the HA Manager. If there is an odd number of cluster members in the cluster then quorum state is achieved when the number of members (application servers hosting partitions) that have successfully started is (N/2) + 1. For a specific example, assume there are five cluster members, then at least three must be online for any application partitions to be activated in that cluster. Three servers constitute achieving a quorum state.

If the number of cluster members is an even number then the same general rule applies, albeit one member will be given two votes versus each getting one in the case of cluster with an odd number of members. If there are four members then three of the four must be online. Each cluster member has a vote, and this vote is normally one; however, when the number of possible cluster members is even, then the “first cluster member” gets two votes. The “first cluster member” is not the application server that manages to start first and is a member of a cluster. Rather, the “first cluster member” is determined by a lexically sorted list of the cluster member names. This includes the entire <Cell>\\<Node>\\<member name> name identifier, not only the member name.

In general, quorum is reached when the sum of the votes from the online cluster members equals or exceeds the majority of the possible votes for the cluster. Once the current cluster membership achieves quorum, then the HA Manager will start activating cluster scoped partitions in round-robin fashion over the set of online members each partition is activated only once across the members of the cluster. The activation process normally results in an unbalanced cluster topology. The included WPF management functions offer the ability to rebalance the partitions across all available cluster members operational at the time the balance is issued.

A partitioned application will continue running until the hosting cluster loses quorum. If the set number of votes falls to less than the majority then the application is stopped. This situation is uncommon and typically only happens when there is a network partition that causes the cluster machines to split into two independent clusters.

For an example, assume a cluster is created, and a core group defined for that cluster. In the core group, if a particular partition is given a preferred server to be activated within in a policy file, the HA Manager will monitor and enforce the appropriate policy for the life cycle of the partition and the cluster member. If the cluster member (a core group member) is brought down for maintenance or simply fails, HA Manager will reactivate the partition on another cluster member. Additionally, if the partition in this example were defined to fail back, HA Manager would move the partition back to the previously failed cluster member when it is available for use in the cluster.

An important fact to keep in mind, to ensure cluster reliability, if quorum is lost all remaining cluster members will be terminated. This is to ensure that the cluster handling the workload begins to enter a running state that is not safe or reliable. Administrators must plan for this case, and provision the cluster as required to account for this. Additionally, quorum is an attribute that can be turned off or on.

4.1.4 WPF Partition HA Manager Implementation

WPF partitions are simply a single implementation of an HA Manager service. The default HA Group properties already predefined for WPF partitions are defined in the table below.

Description	Property Name	Property Value	Cluster Scoped	Node Scoped	Common
Cluster Name	IBM_hc	<cluster name>	X	X	X
HA Group Type	-gt	-p	X	X	X
Partition Name	-pn	<partition name>	X	X	X
Partition Class Name	-pc	<PSSB ejb class name>	X	X	X
Partition Scope	-ps	Cluster: -c Node: -n	X	X	X
Node for group	-pnn	<node>		X	

These properties can be used to define a custom policy that applies to a specific set of partition HA Groups. For example, a policy could be defined to apply to a particular partition's group by included the partition name in the MatchCriteria as was done in the example previously. Alternatively, an administrator or programmer could create a policy that applies to all partitions for the same application by using the Partition Class name in a MatchsetCriteria.

4.1.5 HA Manager Policy Explanation

The HA Manager requires a matching policy for all WPF partitions. Examples of the possible behaviors using the built-in policies are shown in the table below.

Built-in Policy	**HA Manager activation behavior**
One of N	Exactly one of the possible cluster members that can host the partition will be activated at a time. This is 'classic' HA behavior. Keep the partition running on one server at all times. This is the default policy for WPF partitions.
M of N	The partition runs on at most M of the N available cluster members. If less than M members are currently online, then it runs on all of the currently running members.
All active	The service is activated on every available cluster member.
NOOP	The service is not activated on any member. (Not applicable to WPF specific applications).
Static	The HA Manager only activates the service on a specific server. If that server is not available then the service is down. Typically, if this is used then the WebSphere node should be made highly available using conventional clustering solutions such as HACMP and other solutions.

The default policy is the One of N for WPF partitions. Cluster scoped partitions default to have quorum enabled, and Node Scoped partitions do not.

HA Manager policies have the following attributes:

Name : String	This is the unique name of the policy.
MatchsetCriteria : Map<String,String>	This is the set of name/value string properties used to determine which services a policy can manage. Typically, a policy manages more than one HA Group.
Type	This is the policy type. (One of N, M of N, All Active, NOOP and Static).
Type-specific attributes	Each policy type has specific attributes that can be set.
Quorum : boolean	If true then quorum logic is applied to all services managed by the policy.

The match set is used to determine which services are managed by a policy. A service must be managed by exactly one policy. A policy matches a service if its match set is a complete subset of the service name. If multiple policies match a service then the one with the largest match set is the one chosen to manage the service. If there is more than one policy matching the service at the end of this process, then the HA Manager reports an error (`HMGR0302E`) in the log of the coordinator's JVM, indicating that the service is offline. The wpfadmin command has several policy related commands,

one is useful for the cases when many policies may match a given group (and neither will be applied). See the wpfadmin resolvePolicyForGroup invocation.

For example, here is an example, WPF's Default Cluster Scoped policy:

```
<policies xmi:type="coregroup:OneOfNPolicy"
          xmi:id="OneOfNPolicy_1097968497415"
          name="WPF Cluster Scoped Partition Policy"
          description="Default WPF Cluster Scoped Partition Policy"
          policyFactory="com.ibm.ws.hamanager.coordinator.policy.impl.OneOfNPolicyFactory"
          isAlivePeriodSec="-1"
          quorumEnabled="true"
          failback="false"
          preferredOnly="false">
    <MatchCriteria xmi:id="MatchCriteria_1097968497415" name="-gt" value="-p" description="Default WPF Match Criterion"/>
    <MatchCriteria xmi:id="MatchCriteria_1097968497425" name="-ps" value="-c" description="WPF Cluster Scope Match Criterion"/>
</policies>
```

WPF's Default Node Scoped Policy is defined as:

```
<policies xmi:type="coregroup:OneOfNPolicy"
          xmi:id="OneOfNPolicy_1097968497435"
          name="WPF Node Scoped Partition Policy"
          description="Default WPF Node Scoped Partition Policy"
          policyFactory="com.ibm.ws.hamanager.coordinator.policy.impl.OneOfNPolicyFactory"
          isAlivePeriodSec="-1"
          quorumEnabled="false"
          failback="false"
          preferredOnly="false">
      <MatchCriteria xmi:id="MatchCriteria_1097968497465" name="-gt" value="-p" description="Default WPF MatchCriterion"/>
      <MatchCriteria xmi:id="MatchCriteria_1097968497475" name="-ps" value="-n" description="WPF Node Scope Match Criterion"/>
</policies>
```

For example, here is an example of the WPFKeyBasedSample example Partitioned J2EE Application group properties:

4.1.6 Policy Administration

WebSphere 5.1 Extended Deployment does not have the GUI panels in the admin console to manage the HA Manager policies. There are two options for administering the policies.

- wpfadmin (both a script and examples for users to create their own custom scripts)
- wsadmin scripts.

Policies can be updated at any time and take effect immediately (do not require a JVM restart). Refer to the wpfadmin command for examples that explain managing existing, creating new, and deleting HA Manager CoreGroup policies.

4.2 How does a "WPF Partition" relate to an HA Group

With the previous section as a background, it is important to understand a single WPF "partition" is actually an HA Group of members. This may seems counter intuitive. For example, when PK000001 is created, it is created on each cluster member in the specific cluster the application is installed and started within at cluster start.

Each cluster member creates a "Partition" and joins with a group of common attributes as described above. The attributes defined are all the same, but the value of one is unique to the partition name (-pn). For example, each PK000001 partition is created with a property "-pn" having the value of "PK000001". Thus the HA Group representing the partition's HA Group for PK000001 has a member on each available cluster member, and only one.

Depending on the policy chosen, e.g. cluster scoped for example, HA Manager will determine on which cluster member PK000001 will be "activated" upon. The activated instance of the group in WPF has a special significance. For WPF, the active member will receive all work requests for the HA Group that represents PK000001. How many members of the HA Group for a particular partition is activated over the set of active cluster members is called "Partition Scope" and is discussed in more detail below.

The key concept here is to denote that each cluster member creates a member in the HA Group, and the values of the properties are the same. When they are created on the separate cluster members, HA Manager recognizes they are of the same group cluster wide by comparing the group properties providing during application startup, and treats each individually created cluster member created as part of the PK000001 HA Group when the properties are equal.

4.2.1 Partition Scope

Partitions have two possible scopes: cluster and node scope. The application specifies the scope when it creates the partition's PartitionDefinition. The partition scope influences the HA Group properties for the partition at creation time and how the partition's HA Group is handled once the cluster starts.

4.2.1.1 Cluster Scope Partitions

Cluster scoped WPF Partitions are the default type. In this case, when each member of the cluster creates and joins a Partition HA Group, only one instance will have a Partition endpoint activated. This allows clients to be guaranteed that a routing request will be sent to a unique cluster member. Cluster scoped partitions are IIOP routable and only activate when the cluster reaches quorum.

4.2.1.2 Node Scoped Partitions

A node-scoped partition includes a node attribute in the group properties called "-pnn", which has a value of <node name>. Node scoped partitions are named as follows:

If there are five nodes then there will be five active partitions, as the extra attribute –pnn distinguishes each partition on a separate node from another. For example, on NodeA, serverA, the –pnn attribute would have the value "NodeA". All partitions of that type on that logical node (could be N application server cluster members) with that partition's set of similar HA group properties, each set on a specific node establishing their own cluster wide group. Consequently, one these partitions on NodeA will be activated. For the same partitions created on NodeB, all the partitions would be the same for that application, exception the –pnn value would be set to "NodeB", uniquely distinguishing them from the Node Scoped partitions on NodeA.

If a one of N policy matches a node-scoped partition then the partition is activated once on every node with a cluster member started. If there are four nodes with cluster members running, then the partition will be activated once on every one of the four nodes.

Node scoped partitions do NOT wait for cluster quorum (see 10) before activating. Requests to these partition types will be round-robbined across members. Thus, for solutions where more then one endpoint can suffice and would beneficial, node scoped partitions are available.

4.2.2 How many policies are too many?

The number of policies available in the Default Core Group of the HA Manager does affect performance. Each time an action must be applied to a HA Manager Group (a single partition is a HA Group). The default number of WPF policies is two, Cluster and Node Scoped policies. Users can create an enumerable number of more unique policies that meet their requirements. However, keep in mind, when HA Manager attempts to apply the policies it must search through all policies available and determine which one best applies to a specific HA Group.

In general, the suggested approach to policies is to use what you need, but only that many. For example, in the case of the demonstration key based sample, a unique policy was created per partition to control the startup of the application to a given application server in the cluster. If the application had 10,0000 partitions, that would require 10,000 policies. This is obviously not the approach a programmer and administrator should take.

Few policies can be used using additional member properties. One of the createPartitionDefinition(…) API signatures can have a "Map" of additional attributes provided. These properties could help reduce the number of policies. For example, if the user created a new attribute called "startOn", and the value could be a string that represents "server#" where # is a number representing 1 of the cluster members. The policy could then specify the default already provided for in the Cluster Scoped policy (-gt=-p, -ps=-c), but also (startOn=server4). This would provide three match criteria,

and enable # of policies vs. 10,000 partitions. If there were 30 servers for example to start the partitions on, there could be 30 policies, each with a third match set criteria and the preferred server to be server4. See the wpfadmin policy commands for an example, or the example above if you have not stepped through the tutorial.

4.2.3 How many partitions are too many?

As a general rule, applications should use as few partitions possible. WPF can scale to a very large number of partitions; however, more partitions equates to more memory and additional management overhead. The JVMs hosting the HA Manager coordinators must have adequate memory to manage their partitions. It is possible to tell the HA Manager to use more than one coordinator to manage its partitions. If the applications in a core group use many partitions (>1,000), then the HA Manager should be configured to use more coordinators. The actual number that a single coordinator can manage depends on the amount of available memory in the JVM and the number of partitions/coordinator. Partitions are uniformly distributed over the available coordinator JVMs using a hash scheme (See 17 for more information on scaling WPF).

4.3 Advanced HA Manager Concepts

4.3.1 HA Managed "Network partitions"

The danger of a network partition occurring is termed "Split Brain Syndrome". Imagine a seven-node cluster. The network partition results in two clusters of four and three machines respectively. The four-node cluster will start to fail-over the partitions it believes are down, i.e. the partitions that are running on the three nodes. The three node cluster will behave likewise. This results in the same partitions activating in both clusters which is obviously a potentially a catastrophic problem (fatal in the case of how the WPF runtime reacts). We will classify the cluster with four nodes as a majority quorum and the cluster with three nodes as the minority quorum. Majority quorum means the sum of the votes among the cluster members is at least the majority of the total votes available (Seven in this example). The HA Manager will kill all minority cluster members which have active quorum enabled partitions. This kill process happens as soon as the minority cluster is detected and before any partitions are failed over.

4.3.2 Critical time window for network partitions

There still exists a small time window of danger when a network partition occurs. Assume there are seven partitions in seven nodes, with a single partition running on each cluster member. The network partition occurs. We now have partitions 1-4 running in the majority network and partitions 5-7 running on the minority network partition. The HA Manager running in the majority network partition detects what it thinks is an event showing nodes 5-7 failed, and starts to failover the application partitions 5-7 on machines 1-4. A problem arises because the HA Manager running in the minority network partition may not have realized the network partition has occurred before this failover takes place.

This could result in a situation where application partitions 5-7 are running twice, once in the majority network partition and again in the minority network partition. The HA Manager running in the minority network partition will tell its nodes to self exit. It is during this critical window that it is possible for an application partition to be active for a short time period in both network partitions. The application must be designed to 'tolerate' this rare but possible occurrence.

4.3.3 Tolerating the critical time window

If the partitioned application uses a database, then the application can use the following logic to tolerate such occurrences. We make an additional table in the database called "partition_owner". It has two columns:

- P_KEY (string).
  This is the partition name.
- P_OWNER (string).
  The server name that was last activated for the partition.

The application logic should be modified to update the owner column for the partition being activated to the server name. If the record does not exist then the record should be inserted.
Every transaction that results in the application state changing should then verify that the partition is still owned by this server. This check should be the last statement to execute in the transaction. If this check fails then the transaction should be rolled back and any existing exception returned to the client. The application can treat it in a similar manner as a database failure. This logic will prevent the unlikely occurrence of the critical time window from causing any problems for your partitioned applications. Based upon the scenario outlined above, when the majority network partition starts to activate the partitions in the minor network partition, they will update the owner columns. This will

cause the partitions running in the minority network partition to fail, and shortly thereafter the HA Manager will suicide the cluster members ensuring a safe outcome to the network partition. This does incur small performance penalty upon the partitioned application in the form of a single SQL statement per update transaction; however, the cost can be minimized if the SQL request can be combined with the last update/delete statement issued to the database.
The WPF is capable of managing thousands of partitions but this requires planning in order to configure the cluster to manage this. Normal J2EE applications that are clustered have a single route table. This route table is named after the application name and contains the endpoints for every cluster member that is currently running the application. If the cluster has ten members that are currently running then the route table for the application has ten endpoints. When a cluster application uses partitions then there is one additional route table for each partition. This means that if the application has 20,000 partitions then there are 20,001 route tables for the applications. The route table for a partition has an end point for each cluster member in which the HA Manager has activated the partition.

4.3.4 Cluster member memory usage for active partitions

The route table for a partition using a one of N policy consumes about 4KB of memory on the cluster member on which it is active. Cluster members that are not activated for a partition do not have such a route table. The HA Manager coordinators also have a copy of this route table. Therefore, if a cluster member has Y active partitions then there needs to be 4Y KB of memory. If there are N partitions for the application, then the coordinator needs 4N KB of memory.

For example, assume we have an application that has 20,000 partitions. There are 10 machines Node 1 to Node 10. These machines have 4GB of memory and have two CPUs. Each machine has two cluster members and a node agent. Assuming the partitions are spread evenly among the cluster members then each machine has 2000 active partitions, 1000 per cluster member. Thus, each cluster member uses about 4MB (1000 * 4KB) of memory for the route tables. A single coordinator would require about 20000 * 4KB or roughly 80MB of memory; however, the coordinator function can be spread across multiple servers. An example configuration for the HA Manager might be:

- Number of coordinators = 4 (All references to 4 below mean the value of this setting
- Preferred Servers = NodeA_NodeAgent, NodeB_NodeAgent, NodeC_NodeAgent, NodeD_NodeAgent

Based upon this configuration, the HA Manager will run up to four coordinators in the cell. If there are less than 4 JVMs running in the cell, then the HA Manager will use all running JVMs for the coordinator function. The management of route tables will be uniformly distributed using a hashing algorithm of the set of available coordinators (Preferred servers above). The HA Manager normally uses the lexically lowest servers as coordinators. It simply sorts the JVMs using the server name (cell/node/serverName) and picks the lower M for the coordinator function where M is the number of desired coordinators from the cell configuration. Based upon the current example, each coordinator will need 20MB of memory when all four coordinators are running (80MB / 4 coordinators). The preferred coordinator list lets a customer specify which JVMs the customer would like to use as coordinators. The coordinator function is fault tolerant; therefore, if a JVM currently hosting a coordinator fails then another JVM will replace it. In the case where there are less than four JVMs available (this examples preferred number), the coordinators will be redistributed equally over the survivors.

If all machines were running and then NodeB crashed, the coordinator would continue to run on four JVMs, the three remaining preferred servers and the lexically lowest of the rest of the JVMs in the cell, thus keeping the number at four. The JVMs on the machines that are coordinator candidates (i.e. the preferred servers) should have their heap sizes configured to accommodate the possible coordinator function. You can have more preferred servers than coordinators. The HA Manager will just choose the four most preferred servers that are currently running.

4.3.5 Why define more than one coordinator?

We recommend the use of more than one coordinator with a large number of partitions (a thousand or more) to reduce partition activation time and distribute the memory cost associated with the coordinator function.

4.3.6 Partition Activation reaction times.

A coordinator manages the route tables of all the partitions that it is designated to host (based upon hashing algorithm). Let us assume there is a single coordinator for 20,000 partitions. When the set of running JVMs changes, the lone coordinator will evaluate the policy for all 20,000 partitions. It will then send activation/deactivation messages to the surviving set of JVMs in the cell depending on the policy, which with a single coordinator could take an inordinate amount of time More coordinators allows for this work to be performed by multiple machines in parallel and thus improve the cell's reaction time in the event of partition failures.

4.3.7 Memory usage

Recall that 20,000 partitions translate to a charge of about 80MB of memory for a single coordinator server process. Garbage collection could become a factor depending on what else the JVM is hosting. Multiple coordinators allows for the memory burden to be distributed among multiple JVMs.

4.3.8 Coordinator Configuration

When WebSphere Extended Deployment is first installed, it is configured as follows:

- Number of coordinators = 1
- No preferred servers
- Two threads for the HA Manager to activate/deactivate partitions.

This is not a recommended configuration for production, especially if you are using a partitioned application. We recommend the following work be completed prior to production:

- Figure out how many partitions you plan on having.
- Figure out the memory requirements based upon the number of partitions (see **Error! Reference source not found.**).
- Determine how many coordinators (N) you want.
- Assign at least N + 2 preferred servers (so JVM heap sizes can be sized appropriately).
- Assign an appropriate number of threads to the HA Manager thread pool.
    - This depends on the number of processors available on a box as well as the nature of the work performed by the application during partition activation.

4.3.9 Recommendations for preferred server locations

The preferred servers should be on separate physical boxes ideally for isolation reasons. The coordinator adds almost no load on its JVM host during steady state operation with the exception of its memory requirements. We recommend the preferred servers be on 'stable' servers. A stable server is a JVM that is running continually and it should not be stopped normally during production. The reason for this is that when a JVM which is currently a coordinator fails or is stopped then the route tables are redistributed over the surviving JVMs (preferred servers) in the cell. It is better to reduce this churn to a minimum from an operation perspective and this is the reasoning for the recommendations. The additional memory load of a coordinator will increase the frequency of GC and it will reduce the memory available for caches, etc… to any applications running in that JVM. The CPU usage will very slightly influence the JVM when the cluster membership changes. These issues can be avoided by running this function on separate machines if possible.

4.3.10 Reaction times

When a partition is activated on a cluster member then the route table for the partition must be updated. A cluster member can update the route table at a rate of around 250/sec on a uniprocessor 2Ghz P4 system. If we have 10,000 partitions running on ten boxes with 2 cluster members each then this means under normal conditions that there will be 500 active partitions per cluster member. If a single cluster member JVM fails then 500 partitions must be recovered. These will be spread[3] over the surviving 19 cluster members and thus each cluster member will have an additional 25 partitions that will take around 0.1 seconds to have the route tables updated. If a box failed then there would be 1000 partitions to recover. The 1000 partitions are spread over the surviving 9 machines or 18 cluster members. Each cluster member gets an additional 50 partitions that should take around 0.25 second per cluster member for the route tables to activate. The recovery time of the application must be added to the above times also.
We will now show a scenario to highlight potential problems. The customer is using 40,000 partitions on a couple of 4 way systems with two cluster members apiece. This means that normally there are 10,000 partitions on a cluster member. If a single cluster member failed then 3300 partitions would need to be activated on the surviving 3 cluster members. This will take 13 seconds each if it on a uniprocessor and 4 seconds on a 4 way box as activation is multi-threaded[4].

[3] This assumes that all cluster members are equally eligible to host the partition. This isn't true if the customer has policies with preferred servers for certain partitions.
[4] The HA Manager's default configuration uses 2 threads for activation purposes ((HAManagerservice.xml).

4.3.11 HA Manager Event Callback Thread Pool

The HA Manager uses a thread pool to deliver events, such as the partition load and unload to the application. The HA Manager guarantees that sequential events for a particular partition will be delivered serially. So, if the sequence was load and then unload for a partition then HA Manager guarantees that the unload will not be dispatched until the load completes, i.e. the application processes it. This thread pool is defined in the hamanagerserver.xml file that is specific to a particular server. Different servers can have different thread pool sizes.

4.3.12 Number of HA Manager Coordinators

It is possible to change the number of coordinators and the list of preferred servers without restarting any JVMs in the cell, but should be done infrequently to reduce change. See wpfadmin to perform these types of changes.

4.3.13 HA Manager TCP/IP Tuning

HA Manager relies on RMM, a high performant java messaging transport protocol to transport state information between all cluster members. Tuning RMM is critical to high transaction rate WPF transaction processing applications.

4.3.13.1 Overview

Each network socket is allocated a send buffer for outbound packets and a receive socket for inbound packets. These buffers are assigned a default size that depends on parameters of the operating system. The operating system also determines the maximum size of the socket buffers. To support high data rates at the receiver it is imperative to increase the receiver socket buffer size (a value of over 1 Megabyte is recommended). The RMM code attempts to increase socket buffer sizes; in order to succeed the maximum allowed size may have to be configured.
This document describes the procedures of increasing the maximum socket buffer size in different operating systems.

4.3.13.2 AIX

The command to use is '`no`' (i.e., network options).
The parameters are '`sb_max`' , '`udp_sendspace`' and '`udp_recvspace`'.
The format and recommended values are:

```
no –o sb_max=<value> where value= 1048576, 4194304 or 8388608
no –o udp_recvspace=<value> where value= 1048576, 4194304 or 8388608
no –o udp_sendspace=65536

no –a  (to view the current value of all options)
```

Note: the value for sb_max should be greater than the others because it is the max for all socket buffers combined.

The above settings last until the next reboot. To make the changes last across reboots the above command should be added to the end of the file '`/etc/rc.net`' and the full path of the command ('`/usr/sbin/no`') should be used.

4.3.13.3 Linux

The command to use is '`sysctl`' (i.e., system control).
The parameters are '`net.core.rmem_default`' , '`net.core.rmem_max`', '`net.core.wmem_default`' and '`net.core.wmem_max`'
The format and recommended values are

```
sysctl –w net.core.rmem_default=65536
sysctl –w net.core.wmem_default=65536
```



```
sysctl –w net.core.rmem_max=8388608
sysctl –w net.core.wmem_max=8388608

sysctl –a   (to view the current value of all options)
```

The above settings last until the next reboot. To make the changes last across reboots the following lines should be added to the end of the file '`/etc/sysctl.conf`'

```
net.core.rmem_default=65536
net.core.wmem_default=65536
net.core.rmem_max=8388608
net.core.wmem_max=8388608
```

4.3.13.4 Sun

The command to use is '`ndd`' (i.e., network device driver ?).
The parameters are '`udp_xmit_hiwat`' , '`udp_recv_hiwat`' and '`udp_max_buf`' in '`/dev/udp`' and '`icmp_xmit_hiwat`' , '`icmp_recv_hiwat`' and '`icmp_max_buf`' in '`/dev/rawip`'
The format and recommended values are

```
ndd –set /dev/udp udp_xmit_hiwat 65536
ndd –set /dev/udp udp_recv_hiwat 65536
ndd –set /dev/udp udp_max_buf 8088608

ndd –set /dev/rawip icmp_xmit_hiwat 65536
ndd –set /dev/rawip icmp_recv_hiwat 65536
ndd –set /dev/rawip icmp_max_buf 8088608

ndd /dev/udp \?   To view the available parameters
ndd /dev/udp udp_max_buf   To view the current parameter value
```

4.3.13.5 Windows

Under heavy load, one parameter modification that was beneficial was changing the TcpTimedWaitDelay in the registry with regedit. The specific location is:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TCPIP\Parameters

30 is the minimum setting, you will have to create a Dword to add.

4.4 General Cluster and WPF Management Considerations

This section covers general management concerns to be aware of, command line utilities to manage the WPF infrastructure and more specialized information such as default HA Group properties partitions have, and how extra ones created can ease the management of a WPF Partitioned J2EE Application.

4.4.1 Scalability Considerations

For customers who plan to have high workload environments, it is important to plan how the cluster will operate in normal conditions and under failure conditions. This section serves as a reminder to administrators and programmers to avoid common scenarios that degrade cluster performance. Each application solution is different, but these guidelines should provide some general advice to ensure your application can achieve and sustain a performant implementation.

The HA Manager monitors many cluster wide resources. In general, this takes a certain amount of performance. If the cluster members are paging or otherwise engaged such that the HA Manager functionality cannot operate effectively, HA Managed events will begin to occur to account for perceived cluster member anomalies. It is recommended the application servers not be under a large load in normal cases to better handle spikes at times when challenges arise. In addition, reducing virtual memory paging as much as possible will result in a more reliable cluster operational environment.

4.4.2 Conservative Partition Design

The number of partitions in a specific solution should be managed carefully. When possible, fewer partitions are generally simpler and more efficient. Each partition takes system resources to implement within Workload Management, additional administrator effort from a system management standpoint and will take away cluster performance when tracking from a performance monitoring perspective.

As a solution requires more partitions, each of these begins to scale and require either additional resource and/or additional performance to maintain. Some solutions are complicated, and may need larger number of partitions, or even creating two or three different types of partitions to be more efficient (see hybrid partition vs. a key Application Server solution), in these cases carefully manage the solution. Possibly dynamically create partitions as they are needed, and when not required remove them.

From an administrative standpoint for example, often one of the most costly long term aspects of an implementation, managing several thousand partitions and the load placed against them in the cluster is more challenging when compared to a solution requiring the management of hundreds of partitions. However, several solutions might require in the order of thousands of partitions, if so, the computing resources, developers and administrative resources to address the problem will be more extensive.

Internally to IBM, WPF has been tested with 10,000 partitions under load across several machines successfully. A key concern finding during those tests was that the number of active coordinators must be 4 at least in this case. Additionally, using the HA Manager Policy to set these coordinators to specific physical machines in the cluster and setting preferred servers for the partitions to avoid these machines (or even application servers if have a reduced number of machines) proved to be beneficial.

More details are provided in the management section, but in general we think users should manage resources conservatively. This approach ensures that when performance spikes occur and failure conditions arise, the operational integrity of the cluster is not compromised.

4.4.3 Physical Machines CPU and Paging Utilization

In general, machines should be provisioned within the cluster to utilize 10-20% CPU under load. This is to ensure that under heavy load scenarios, failure and failover cases the remaining cluster members can handle the load. In addition, this reduces the need for the operating system to use virtual memory paging. For OnDemand, high throughput capable systems that need to handle critical workloads, avoiding paging is important.

In addition, be mindful that many factors affect performance. The following sections are ideas and reminders programmers and administrators should consider when implementing a high throughput solution (several outside the scope of WPF itself).

4.4.4 Application Thread Pools (Async Beans)

For applications with computational expensive operations which process incoming requests from a non-transactional message transport, use a separate 'worker' thread to perform the computation. After receiving the request, create an Async Bean Work instance and submit the work to a WorkManager to be performed on a separate thread.

When the Work completes, carry on processing the request. Let us assume that your application is running on a four-way SMP machine. The WorkManager used for the application should be limited to two threads and should not be growable, limiting the impact of these long running tasks to two of the four CPUs. The remaining two CPUs can be used to schedule the short running tasks.

Please refer to WBI-SF documentation for more details, or subsequent to the general availability the Extended Deployment Library.

4.4.5 Carefully control what is running on each node and application server in the cluster

It may be obvious, but to achieve the highest performance levels do not run anything else on the cluster members with your critical application, especially when it has 2 or less CPUs. Do not run any computational commands on the machine such as tar, gzip, or similar applications. These commands may negatively impact thread scheduling for the applications.

Additionally, service functions such as the WPF PMI Performance Monitor and the HA Manager coordinator(s) should be carefully managed. The HA Manager policy mechanism covered above allows the administrator to be very specific about which runtime components should run where.

For example, if the performance monitoring function will be used often for your environment, it is suggested the PMI Aggregator be configured to run on an application server exclusively. This will then prohibit the chance that performance monitoring could slow production functionality. Even better solution is to put the performance monitoring PMI Aggregator on another machine.

The WPF PMI Aggregator is one example, there are several such as the coordinator that implemented the HA Manager (in some cases several as more partitions are activated in the cluster).

4.4.6 Tune the operating system to use small time slices.

Some operating systems use time slices up to 200ms, which is just too long when many runnable threads could be scheduled. Lowering the maximum time slice to 10-30ms may help by allowing more threads per second run through the available CPUs. This can lower the scheduling latency at the expense of delaying threads that take longer than 10ms to complete. Some application specific tuning may be required, but try different values to see what works best for your application. There may be more context switches, but most modern microprocessors are capable of billions of instructions per second

AIX	http://www-106.ibm.com/developerworks/eserver/articles/aix5_cpu/
Linux 2.2/2.4	http://www.linuxworld.com/story/34374.htm

4.4.7 Mixing application types must be considered carefully

If your application set has relatively short requests, then installing several similar applications on the cluster should provide reasonable throughput. However, the use of CPU intensive requests require special considerations.

One approach is to have a Partitioned J2EE Application for compute intensive application in a unique cluster within the cell, with exclusive access to specialized hardware for that purpose. Alternatively, if one needs to mix a lighter application set with a more computationally bound application set within the same cluster, the WPF framework can prove helpful if managed correctly.

To managed the mix of application types, it would be prudent to utilize the HA Manager Policy function to focus CPU intensive procedure calls to physical and logical nodes designed to handle that load. The client use the WPF framework can then route the requests to the partitions collocated with computing resource designed to fit the task within the cluster. In addition, WPF allows the administrator to dynamically modify the target endpoint if additional computer resources can be acquired under heavy load situations.

An additional approach is to create a partition set with an M of N policy with the HA Manager Policy framework. Then, set the preferred only attribute to true coupled with designating specific establishing preferred servers for those specific partitions to dedicated machines in the cluster for that purpose. Finally, create a client to direct requests to the cluster resources designed to handle that load. The M of N policy lets you assign more machines for that workload if you need it plus IIOP WLM will round robin the requests over the set of machines which are running the heavy load capable partition set.

In summary, you can achieve the same thing without partitioning by simply putting the heavy logic J2EE application and deploying it on its own cluster, and then use WebSphere Extended Deployment's dynamic cluster support to dynamically expand/contract the cluster if necessary. Please see the WebSphere Extended Deployment Infocenter for more details on the later option.

4.4.8 SMP machines preferred in partitioned implementations

Two or more processors will execute your requests faster than one for a runnable queue of N entries, and four processors are twice as fast again (in general). If you cannot segregate the applications with lighter CPU workloads onto a smaller box, then an SMP solution may be more appropriate, as SMPs are inherently less susceptible to the problems associated with mixed CPU load scenarios.

4.4.9 OnDemand LPAR Resource Advantages

If the WPF enabled applications are running within a cluster consisting of LPAR capable machines, then it is possible to allocate additional CPU resources to mitigate latency or higher demand scenarios without suffering even a partition subset outage. AIX/pSeries offers this support. For many customers, this approach could be critical.

For example, if a customer utilizes a blade center or Linux cluster, e.g. many smaller footprint servers, and wishes to even out the load against the given cluster members, an outage for at least a subset of the endpoint partitions will occur, This occurs because WPF does offer the ability balance partitions dynamically between blades, either in mass or by selectively moving one partition at a time between application servers. However, this does result in a short term outage as the partition being moved needs to be taken off-line (deactivated with a PartitionUnLoadEvent(…)) and then reactivated on another physical blade.

In the case of an LPAR capable machine, more resources can be provided to handle partitions that are receiving abnormally high number of transaction requests. Many of the IBM servers now allow administrators to literally contact the IBM web site to purchase additional LPAR resources when heavy load situations occur.

4.4.10 Dealing with hot partitions

This section provides some recommendations to deal with the case of a hot partition, or a partition that is experiencing an inordinately high workload for which the hosting server cannot keep-up

4.4.10.1 Move the busy partition(s) to a less busy server.

Use the HA Manager JMX commands to move the busy partition from one server to another or the ***wpfadmin*** command. This action will deactivate the partition on the original server and then activate it on the new server (the partition will be offline during this period). Most of the offline time is associated with the application's deactivate and

activate code processing. As mentioned in the OnDemand LPAR section above, this does result in an outage for the specific partition. The approach below is preferred over this approach.

4.4.10.2 Move the less critical partitions to another server.

This process is the inverse of the process discussed in above. In this case, to avoid an outage on the busiest partition move the less busy partitions on the same cluster member to free up CPU capacity for the busy partitions. In this scenario, the less busy partitions will incur the temporary outage, which may be more acceptable than deactivating the busy partition(s).

4.4.10.3 LPAR expansion (best option to avoid outages)

For a cluster member hosted on an LPAR, it may be possible to allocate more CPU resource to provide additional capacity for the partition. This solution does not require an outage and illustrates the major advantage of using WebSphere Extended Deployment WPF framework.

4.5 Management Script (wpfadmin) and Usage

The wpfadmin is a python script that allows the user to perform several administrative operations on a cluster. This script is not only meant for customers to use, but also provides programming examples allowing them to create their own automation command library.

The script calls a HA Manager MBean (JMXCoordinator), which then calls directly to the HA Manager runtime support to perform the operations as, described above. The following is the list of operations to be supported in the wpfadmin script, along with specific usage scenarios.

For the purposes of this section, the application WPFKeyBasedPartitionSample has been installed to a two node cluster with thirty partitions divided into two classifications: PK000001 - PK000010 belonging to class1 and PK000011 - PK000030 belonging to class2. The system wpfsample3 is the deployment manager and also has four application servers, and wpfsample2 has five application servers.

In the examples below, wpfadmin is executed to demonstrate example invocations. On Windows platforms, the wpfadmin.cmd application can be used, and referenced on the command line as “wpfadmin”. The command functions identically across all supported platforms unless documented otherwise for a specific command.

4.5.1 Management Commands

The wpfadmin script provides many commands to assist in managing a WPF environment. The commands ease the burden of setting trace specifications, manage active partition members, managed polices and many other tasks. As a customer you are encouraged to enhance the example script, and create your own as you see fit to better automate your own environment.

4.5.2 listActive

Displays the application servers hosting active partitions. This command can be scoped to only show information for a particular application, partition, and classification.

Available options:
--o <number of partitions> . The number of partitions that are printed out. If the --o is not specified; it will default to 50 partitions.
--a <application name>. Prints out partition information for the given application.
--p <partition name>. Prints out partition information for the given partition.
--class <classification name>. Prints out partition information for the given classification.

Usage: ./wpfadmin listActive

Example:

```
[root@wpfsample3 bin]# ./wpfadmin listActive
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector;  The type of process is: DeploymentManager
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000030: Server wpfsample3Network\wpfsample2\s7
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000029: Server wpfsample3Network\wpfsample2\s6
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000028: Server wpfsample3Network\wpfsample2\s5
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000027: Server wpfsample3Network\wpfsample2\s4
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000026: Server wpfsample3Network\wpfsample3\s9
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000025: Server wpfsample3Network\wpfsample3\s3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000024: Server wpfsample3Network\wpfsample3\s2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000023: Server wpfsample3Network\wpfsample3\s1
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000022: Server wpfsample3Network\wpfsample2\s8
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000021: Server wpfsample3Network\wpfsample2\s7
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000020: Server wpfsample3Network\wpfsample2\s6
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000019: Server wpfsample3Network\wpfsample2\s5
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000018: Server wpfsample3Network\wpfsample2\s4
```


WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000017: Server wpfsample3Network\wpfsample3\s9
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000016: Server wpfsample3Network\wpfsample3\s3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000015: Server wpfsample3Network\wpfsample3\s2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000014: Server wpfsample3Network\wpfsample3\s1
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000013: Server wpfsample3Network\wpfsample2\s8
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000012: Server wpfsample3Network\wpfsample2\s7
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000011: Server wpfsample3Network\wpfsample2\s6
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000010: Server wpfsample3Network\wpfsample2\s5
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000009: Server wpfsample3Network\wpfsample2\s4
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000008: Server wpfsample3Network\wpfsample3\s9
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000007: Server wpfsample3Network\wpfsample3\s3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000006: Server wpfsample3Network\wpfsample3\s2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000005: Server wpfsample3Network\wpfsample3\s1
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000004: Server wpfsample3Network\wpfsample2\s8
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000003: Server wpfsample3Network\wpfsample2\s7
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000002: Server wpfsample3Network\wpfsample2\s6
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000001: Server wpfsample3Network\wpfsample2\s5

4.5.3 listActiveWithGroups

Displays the application servers which are active for the set of groups defined by the matchset parameter passed in.

Available options:
--o <number of partitions> . The number of partitions that are printed out. If the --o is not specified; it will default to 50 partitions.
--m <matchset>. Prints out partition information for the given matchset.
Usage: ./wpfadmin listActiveWithGroups –m -gt=-p,-pc=ClassA

Example:

[root@wpfsample3 bin]# ./wpfadmin listActiveWithGroups --m -pc=class1
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Matchset set to partition_class=class1
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000010: Server wpfsample3Network\wpfsample2\s5
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000009: Server wpfsample3Network\wpfsample2\s4
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000008: Server wpfsample3Network\wpfsample3\s9
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000007: Server wpfsample3Network\wpfsample3\s3
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000006: Server wpfsample3Network\wpfsample3\s2
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000005: Server wpfsample3Network\wpfsample3\s1
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000004: Server wpfsample3Network\wpfsample2\s8
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000003: Server wpfsample3Network\wpfsample2\s7
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000002: Server wpfsample3Network\wpfsample2\s6
WPFC0050I: Application WPFKeyBasedPartitionSample, Partition PK000001: Server wpfsample3Network\wpfsample2\s5

4.5.4 countActivePartitionsOnServers

Counts the number of partitions on all servers. This command can be scoped to only show information for a particular application, partition, and classification.

Available options:
--o <number of partitions> . The number of partitions that are printed out. If the --o is not specified; it will default to 50 partitions.
--a <application name>. Prints out partition information for the given application.
--p <partition name>. Prints out partition information for the given partition.
--class <classification name>. Prints out partition information for the given classification.

Usage: ./wpfadmin countActivePartitionsOnServers

Example:

```
[root@wpfsample3 bin]# ./wpfadmin countActivePartitionsOnServers
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector;  The type of process is: DeploymentManager
WPFC0051I: Server wpfsample3Network\wpfsample3\s9: 3
WPFC0051I: Server wpfsample3Network\wpfsample3\s3: 3
WPFC0051I: Server wpfsample3Network\wpfsample3\s2: 3
WPFC0051I: Server wpfsample3Network\wpfsample2\s8: 3
WPFC0051I: Server wpfsample3Network\wpfsample3\s1: 3
WPFC0051I: Server wpfsample3Network\wpfsample2\s7: 4
WPFC0051I: Server wpfsample3Network\wpfsample2\s6: 4
WPFC0051I: Server wpfsample3Network\wpfsample2\s5: 4
WPFC0051I: Server wpfsample3Network\wpfsample2\s4: 3
```

4.5.5 countActiveGroupsOnServers

Counts the number of groups that match the matchset parameter on all servers

Available options: --o <number of partitions> . The number of partitions that are printed out. If the --o is not specified; it will default to 50 partitions.
--m <matchset>. Prints out partition information for the given matchset.
Usage: ./wpfadmin countActiveGroupsOnServers –m -gt=-p,-pc=ClassA

Example:

```
[root@wpfsample3 bin]# ./wpfadmin countActiveGroupsOnServers --m -pc=class2
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector;  The type of process is: DeploymentManager
WPFC0065I: Matchset set to partition_class=class2
WPFC0051I: Server wpfsample3Network\wpfsample3\s9: 2
WPFC0051I: Server wpfsample3Network\wpfsample3\s3: 2
WPFC0051I: Server wpfsample3Network\wpfsample3\s2: 2
WPFC0051I: Server wpfsample3Network\wpfsample2\s8: 2
WPFC0051I: Server wpfsample3Network\wpfsample3\s1: 2
WPFC0051I: Server wpfsample3Network\wpfsample2\s7: 3
WPFC0051I: Server wpfsample3Network\wpfsample2\s6: 3
WPFC0051I: Server wpfsample3Network\wpfsample2\s5: 2
WPFC0051I: Server wpfsample3Network\wpfsample2\s4: 2
```

4.5.6 list

Lists the partitions and the status of the member servers. This command can be scoped to only show information for a particular application, partition, and classification.

Available options:
--o <number of partitions> . The number of partitions that are printed out. If the --o is not specified; it will default to 50 partitions.
--a <application name>. Prints out partition information for the given application.
--p <partition name>. Prints out partition information for the given partition.
--class <classification name>. Prints out partition information for the given classification.

Usage: ./wpfadmin list

```
[root@wpfsample3 bin]# ./wpfadmin list
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector;  The type of process is: DeploymentManager
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000030
  wpfsample3Network\wpfsample2\s4
  wpfsample3Network\wpfsample2\s5
  wpfsample3Network\wpfsample2\s6
  wpfsample3Network\wpfsample2\s7*
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1
  wpfsample3Network\wpfsample3\s2
  wpfsample3Network\wpfsample3\s3
  wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000029
  wpfsample3Network\wpfsample2\s4
  wpfsample3Network\wpfsample2\s5
  wpfsample3Network\wpfsample2\s6*
  wpfsample3Network\wpfsample2\s7
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1
  wpfsample3Network\wpfsample3\s2
  wpfsample3Network\wpfsample3\s3
  wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000028
  wpfsample3Network\wpfsample2\s4
  wpfsample3Network\wpfsample2\s5*
  wpfsample3Network\wpfsample2\s6
  wpfsample3Network\wpfsample2\s7
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1
  wpfsample3Network\wpfsample3\s2
  wpfsample3Network\wpfsample3\s3
  wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000027
  wpfsample3Network\wpfsample2\s4*
  wpfsample3Network\wpfsample2\s5
  wpfsample3Network\wpfsample2\s6
  wpfsample3Network\wpfsample2\s7
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1
  wpfsample3Network\wpfsample3\s2
  wpfsample3Network\wpfsample3\s3
  wpfsample3Network\wpfsample3\s9
```

Not all output for this command is shown.

4.5.7 listGroups

Displays the groups and the status of the members.

Available options:
--o <number of partitions> . The number of partitions that are printed out. If the --o is not specified, it will default to 50 partitions.
--m <matchset>. Prints out partition information for the given matchset.

Usage: ./wpfadmin listGroups –m -gt=-p,-pc=ClassA

Example:

```
[root@wpfsample3 bin]# ./wpfadmin listGroups --m -pc=class1
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector;  The type of
process is: DeploymentManager
WPFC0065I: Matchset set to partition_class=class1
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000010
  wpfsample3Network\wpfsample2\s4
  wpfsample3Network\wpfsample2\s5*
  wpfsample3Network\wpfsample2\s6
  wpfsample3Network\wpfsample2\s7
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1
  wpfsample3Network\wpfsample3\s2
  wpfsample3Network\wpfsample3\s3
  wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000009
  wpfsample3Network\wpfsample2\s4*
  wpfsample3Network\wpfsample2\s5
  wpfsample3Network\wpfsample2\s6
  wpfsample3Network\wpfsample2\s7
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1
  wpfsample3Network\wpfsample3\s2
  wpfsample3Network\wpfsample3\s3
  wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000008
  wpfsample3Network\wpfsample2\s4
  wpfsample3Network\wpfsample2\s5
  wpfsample3Network\wpfsample2\s6
  wpfsample3Network\wpfsample2\s7
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1
  wpfsample3Network\wpfsample3\s2
  wpfsample3Network\wpfsample3\s3
  wpfsample3Network\wpfsample3\s9*
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000007
  wpfsample3Network\wpfsample2\s4
  wpfsample3Network\wpfsample2\s5
  wpfsample3Network\wpfsample2\s6
  wpfsample3Network\wpfsample2\s7
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1
  wpfsample3Network\wpfsample3\s2
  wpfsample3Network\wpfsample3\s3*
  wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000006
  wpfsample3Network\wpfsample2\s4
  wpfsample3Network\wpfsample2\s5
  wpfsample3Network\wpfsample2\s6
  wpfsample3Network\wpfsample2\s7
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1
  wpfsample3Network\wpfsample3\s2*
  wpfsample3Network\wpfsample3\s3
  wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000005
  wpfsample3Network\wpfsample2\s4
  wpfsample3Network\wpfsample2\s5
  wpfsample3Network\wpfsample2\s6
  wpfsample3Network\wpfsample2\s7
  wpfsample3Network\wpfsample2\s8
  wpfsample3Network\wpfsample3\s1*
  wpfsample3Network\wpfsample3\s2
  wpfsample3Network\wpfsample3\s3
  wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000004
  wpfsample3Network\wpfsample2\s4
```


wpfsample3Network\wpfsample2\s5
wpfsample3Network\wpfsample2\s6
wpfsample3Network\wpfsample2\s7
wpfsample3Network\wpfsample2\s8*
wpfsample3Network\wpfsample3\s1
wpfsample3Network\wpfsample3\s2
wpfsample3Network\wpfsample3\s3
wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000003
wpfsample3Network\wpfsample2\s4
wpfsample3Network\wpfsample2\s5
wpfsample3Network\wpfsample2\s6
wpfsample3Network\wpfsample2\s7*
wpfsample3Network\wpfsample2\s8
wpfsample3Network\wpfsample3\s1
wpfsample3Network\wpfsample3\s2
wpfsample3Network\wpfsample3\s3
wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000002
wpfsample3Network\wpfsample2\s4
wpfsample3Network\wpfsample2\s5
wpfsample3Network\wpfsample2\s6*
wpfsample3Network\wpfsample2\s7
wpfsample3Network\wpfsample2\s8
wpfsample3Network\wpfsample3\s1
wpfsample3Network\wpfsample3\s2
wpfsample3Network\wpfsample3\s3
wpfsample3Network\wpfsample3\s9
WPFC0046I: Application WPFKeyBasedPartitionSample, Partition PK000001
wpfsample3Network\wpfsample2\s4
wpfsample3Network\wpfsample2\s5*
wpfsample3Network\wpfsample2\s6
wpfsample3Network\wpfsample2\s7
wpfsample3Network\wpfsample2\s8
wpfsample3Network\wpfsample3\s1
wpfsample3Network\wpfsample3\s2
wpfsample3Network\wpfsample3\s3
wpfsample3Network\wpfsample3\s9

4.5.8 coreGroupStatus

Shows the core group name, coordinator names, and active servers in the core group.

Usage: ./wpfadmin coreGroupStatus

Example:

[root@wpfsample3 bin]# ./wpfadmin coreGroupStatus
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0047I: Core group name: DefaultCoreGroup
WPFC0048I: Coordinator servers:
wpfsample3Network\wpfsample2\s7
wpfsample3Network\wpfsample3\s1
wpfsample3Network\wpfsample3\s9
wpfsample3Network\wpfsample2\s5
WPFC0049I: Visible members:
wpfsample3Network\wpfsample2\nodeagent
wpfsample3Network\wpfsample2\s4
wpfsample3Network\wpfsample2\s5
wpfsample3Network\wpfsample2\s6
wpfsample3Network\wpfsample2\s7
wpfsample3Network\wpfsample2\s8
wpfsample3Network\wpfsample3Manager\Deployment Manager
wpfsample3Network\wpfsample3\nodeagent
wpfsample3Network\wpfsample3\s1
wpfsample3Network\wpfsample3\s2

wpfsample3Network\wpfsample3\s3
wpfsample3Network\wpfsample3\s9

4.5.9 move

Moves one partition to another server.

Available options:
--p <partition name>. The name of the partition to move.
--d <server>. Destination server for the partition to move to, of the form <cell>/<node>/<server>

Usage: ./wpfadmin move –p PartitionB –d CellA/NodeA/Server1

Warning: The partition will receive an outage for this command. The programmers for this application should have implemented the partitionLoadEvent(…) and partitionUnloadEvent(…) as described in the programming section. In addition, if a startup issue or another case where policies could establish proper location to avoid the use of this command, please see the management section describing the key based sample and the policy sections.

Example:

[root@wpfsample3 bin]# ./wpfadmin move --p PK000001 --d wpfsample3Network/wpfsample2/s4
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Partition set to PK000001
WPFC0065I: Destination set to wpfsample3Network/wpfsample2/s4
WPFC0054I: Partition PK000001 has moved from Server wpfsample3Network\wpfsample2\s6 to Server wpfsample3Network\wpfsample2\s4

4.5.10 balance

Balances partitions across the set of active servers. Also, if the user specifies the –id option, only the partitions for which PMI statistics are being gathered relative to the id specified will be balanced.

Available options:
--o <number of partitions> . The number of partitions to balance. If the --o is not specified, it will default to 50 partitions.
--a <application name>. Balance partitions of the given application.
--class <classification name>. Balance partitions for the given classification.
--id <PMI id>. Balance partitions for the given PMI id.

Warning: Each partition that is selected to be moved will receive an outage when this command is executed, the programmer needs to have implemented the partitionLoadEvent(…) and partitionUnloadEvent(…) as described in the programming section. In addition, if a startup issue or another case where policies could establish proper location to avoid the use of this command, please see the management section describing the key based sample and the policy sections.

Usage: ./wpfadmin balance

Example 1:

[root@wpfsample3 bin]# ./wpfadmin balance
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0054I: Partition PK000030 has moved from Server wpfsample3Network\wpfsample2\s7 to Server wpfsample3Network\wpfsample2\s4
WPFC0054I: Partition PK000029 has moved from Server wpfsample3Network\wpfsample2\s6 to Server wpfsample3Network\wpfsample2\s4
WPFC0054I: Partition PK000028 has moved from Server wpfsample3Network\wpfsample2\s5 to Server wpfsample3Network\wpfsample2\s4
WPFC0054I: Partition PK000027 has moved from Server wpfsample3Network\wpfsample2\s4 to Server wpfsample3Network\wpfsample2\s5

WPFC0054I: Partition PK000026 has moved from Server wpfsample3Network\wpfsample3\s9 to Server wpfsample3Network\wpfsample2\s5
WPFC0054I: Partition PK000025 has moved from Server wpfsample3Network\wpfsample3\s3 to Server wpfsample3Network\wpfsample2\s5
WPFC0054I: Partition PK000024 has moved from Server wpfsample3Network\wpfsample3\s2 to Server wpfsample3Network\wpfsample2\s6
WPFC0054I: Partition PK000023 has moved from Server wpfsample3Network\wpfsample3\s1 to Server wpfsample3Network\wpfsample2\s6
WPFC0054I: Partition PK000022 has moved from Server wpfsample3Network\wpfsample2\s8 to Server wpfsample3Network\wpfsample2\s6
WPFC0054I: Partition PK000021 has moved from Server wpfsample3Network\wpfsample2\s7 to Server wpfsample3Network\wpfsample2\s7
WPFC0054I: Partition PK000020 has moved from Server wpfsample3Network\wpfsample2\s6 to Server wpfsample3Network\wpfsample2\s7
WPFC0054I: Partition PK000019 has moved from Server wpfsample3Network\wpfsample2\s5 to Server wpfsample3Network\wpfsample2\s7
WPFC0054I: Partition PK000018 has moved from Server wpfsample3Network\wpfsample2\s4 to Server wpfsample3Network\wpfsample2\s8
WPFC0054I: Partition PK000017 has moved from Server wpfsample3Network\wpfsample3\s9 to Server wpfsample3Network\wpfsample2\s8
WPFC0054I: Partition PK000016 has moved from Server wpfsample3Network\wpfsample3\s3 to Server wpfsample3Network\wpfsample2\s8
WPFC0054I: Partition PK000015 has moved from Server wpfsample3Network\wpfsample3\s2 to Server wpfsample3Network\wpfsample3\s1
WPFC0054I: Partition PK000014 has moved from Server wpfsample3Network\wpfsample3\s1 to Server wpfsample3Network\wpfsample3\s1
WPFC0054I: Partition PK000013 has moved from Server wpfsample3Network\wpfsample2\s8 to Server wpfsample3Network\wpfsample3\s1
WPFC0054I: Partition PK000012 has moved from Server wpfsample3Network\wpfsample2\s7 to Server wpfsample3Network\wpfsample3\s2
WPFC0054I: Partition PK000011 has moved from Server wpfsample3Network\wpfsample2\s6 to Server wpfsample3Network\wpfsample3\s2
WPFC0054I: Partition PK000010 has moved from Server wpfsample3Network\wpfsample2\s5 to Server wpfsample3Network\wpfsample3\s2
WPFC0054I: Partition PK000009 has moved from Server wpfsample3Network\wpfsample2\s4 to Server wpfsample3Network\wpfsample3\s3
WPFC0054I: Partition PK000008 has moved from Server wpfsample3Network\wpfsample3\s9 to Server wpfsample3Network\wpfsample3\s3
WPFC0054I: Partition PK000007 has moved from Server wpfsample3Network\wpfsample3\s3 to Server wpfsample3Network\wpfsample3\s3
WPFC0054I: Partition PK000006 has moved from Server wpfsample3Network\wpfsample3\s2 to Server wpfsample3Network\wpfsample3\s9
WPFC0054I: Partition PK000005 has moved from Server wpfsample3Network\wpfsample3\s1 to Server wpfsample3Network\wpfsample3\s9
WPFC0054I: Partition PK000004 has moved from Server wpfsample3Network\wpfsample2\s8 to Server wpfsample3Network\wpfsample3\s9
WPFC0054I: Partition PK000003 has moved from Server wpfsample3Network\wpfsample2\s7 to Server wpfsample3Network\wpfsample2\s4
WPFC0054I: Partition PK000002 has moved from Server wpfsample3Network\wpfsample2\s6 to Server wpfsample3Network\wpfsample2\s5
WPFC0054I: Partition PK000001 has moved from Server wpfsample3Network\wpfsample2\s4 to Server wpfsample3Network\wpfsample2\s6

Example 2:

Another example of the balance command, only balancing the partitions that have the class2 classification:

[root@wpfsample3 bin]# ./wpfadmin balance --class class2
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Classification set to class2
WPFC0054I: Partition PK000030 has moved from Server wpfsample3Network\wpfsample2\s4 to Server wpfsample3Network\wpfsample2\s4
WPFC0054I: Partition PK000029 has moved from Server wpfsample3Network\wpfsample2\s4 to Server wpfsample3Network\wpfsample2\s4
WPFC0054I: Partition PK000028 has moved from Server wpfsample3Network\wpfsample2\s4 to Server wpfsample3Network\wpfsample2\s5
WPFC0054I: Partition PK000027 has moved from Server wpfsample3Network\wpfsample2\s5 to Server wpfsample3Network\wpfsample2\s5

WPFC0054I: Partition PK000026 has moved from Server wpfsample3Network\wpfsample2\s5 to Server wpfsample3Network\wpfsample2\s6
WPFC0054I: Partition PK000025 has moved from Server wpfsample3Network\wpfsample2\s5 to Server wpfsample3Network\wpfsample2\s6
WPFC0054I: Partition PK000024 has moved from Server wpfsample3Network\wpfsample2\s6 to Server wpfsample3Network\wpfsample2\s7
WPFC0054I: Partition PK000023 has moved from Server wpfsample3Network\wpfsample2\s6 to Server wpfsample3Network\wpfsample2\s7
WPFC0054I: Partition PK000022 has moved from Server wpfsample3Network\wpfsample2\s6 to Server wpfsample3Network\wpfsample2\s8
WPFC0054I: Partition PK000021 has moved from Server wpfsample3Network\wpfsample2\s7 to Server wpfsample3Network\wpfsample2\s8
WPFC0054I: Partition PK000020 has moved from Server wpfsample3Network\wpfsample2\s7 to Server wpfsample3Network\wpfsample3\s1
WPFC0054I: Partition PK000019 has moved from Server wpfsample3Network\wpfsample2\s7 to Server wpfsample3Network\wpfsample3\s1
WPFC0054I: Partition PK000018 has moved from Server wpfsample3Network\wpfsample2\s8 to Server wpfsample3Network\wpfsample3\s2
WPFC0054I: Partition PK000017 has moved from Server wpfsample3Network\wpfsample2\s8 to Server wpfsample3Network\wpfsample3\s2
WPFC0054I: Partition PK000016 has moved from Server wpfsample3Network\wpfsample2\s8 to Server wpfsample3Network\wpfsample3\s3
WPFC0054I: Partition PK000015 has moved from Server wpfsample3Network\wpfsample3\s1 to Server wpfsample3Network\wpfsample3\s3
WPFC0054I: Partition PK000014 has moved from Server wpfsample3Network\wpfsample3\s1 to Server wpfsample3Network\wpfsample3\s9
WPFC0054I: Partition PK000013 has moved from Server wpfsample3Network\wpfsample3\s1 to Server wpfsample3Network\wpfsample3\s9
WPFC0054I: Partition PK000012 has moved from Server wpfsample3Network\wpfsample3\s2 to Server wpfsample3Network\wpfsample2\s4
WPFC0054I: Partition PK000011 has moved from Server wpfsample3Network\wpfsample3\s2 to Server wpfsample3Network\wpfsample2\s5

4.5.11 disablePartition

Disables a partition. Depending on how policies are configured, the partition will either be enabled on another server, or will just be disabled.

Available options:
--p <partition name>. The name of the partition to disable.

Usage: ./wpfadmin disablePartition –p PartitionA

Example:

[root@wpfsample3 bin]# ./wpfadmin disablePartition --p PK000002
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Partition set to PK000002
WPFC0053I: Partition PK000002 has been stopped

4.5.12 enablePartition

Enables a partition.

Available options:
--p <partition name>. The name of the partition to enable.

Usage: ./wpfadmin enablePartition --p PartitionA

Example:

[root@wpfsample3 bin]# ./wpfadmin enablePartition --p PK000002

WASX7209I: Connected to process "dmgr" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Partition set to PK000002
WPFC0053I: Partition PK000002 has been enabled

4.5.13 addServerToCoreGroup

Adds the specified server from the DefaultCoreGroup. This command modifies the coregroup.xml found in the ND_HOME/config/cells/<cell_name>/coregroups/DefaultCoreGroup directory.

Available options:
--s <server name>. Server to add to the core group, of the form <cell>/<node>/<server>

Usage: ./wpfadmin addServerToCoreGroup --s wpfsample3Network/wpfsample2/nodeagent

Example:

[root@wpfsample3 bin]# ./wpfadmin addServerToCoreGroup --s wpfsample3Network/wpfsample2/nodeagent
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Server set to wpfsample3Network/wpfsample2/nodeagent
WPFC0067I: Server wpfsample2\nodeagent has been added to DefaultCoreGroup

4.5.14 removeServerFromCoreGroup

Deletes the specified server from the DefaultCoreGroup. This command modifies the coregroup.xml found in the ND_HOME/config/cells/<cell_name>/coregroups/DefaultCoreGroup directory.

Available options:
--s <server name>. Server to remove from the core group, of the form <cell>/<node>/<server>

Usage: ./wpfadmin removeServerFromCoreGroup --s wpfsample3Network/wpfsample2/nodeagent

Example:

[root@wpfsample3 bin]# ./wpfadmin removeServerFromCoreGroup --s wpfsample3Network/wpfsample2/nodeagent
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Server set to wpfsample3Network/wpfsample2/nodeagent
WPFC0068I: Server wpfsample2\nodeagent has been removed from DefaultCoreGroup

4.5.15 enableWPFPMI

Enables the WPF PMI module for all active servers. How PMI works will be explained in the Performance Monitoring section.
Available options:
 <level>. Level of PMI statistics that will be gathered for the wpfModule. Normally should be set to H.
 --c <cluster name>. All servers in the given cluster will have PMI enabled.

Usage: ./wpfadmin enableWPFPMI H –c cluster1

[root@wpfsample3 bin]# ./wpfadmin enableWPFPMI H --c c
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Cluster set to c
WPFC0043I: The wpfModule of PMI is enabled for cluster c and set to level H.

4.5.16 subscribeWPFPMI

Subscribes the asynchronous PMI module for WPF. Here's an example usage of this command:

./wpfadmin subscribeWPFPMI [active|cumulative] [TransactionCount|ResponseTime] <application_name> <ejb_name> <partition_count> <aggregator_interval> --c <cluster_name).

If active is passed, the data the user sees will be reset every time the aggregator interval runs its course. If cumulative is passed, the data will not be reset. If TransactionCount is passed, the latest aggregated data for each of the partitions regarding the number of transactions processed will be returned. If ResponseTime is passed, the latest aggregated data for each of the partitions regarding the minimum, maximum, and average response time will be returned. These two options allow the user to look at what partitions are processing the most transactions currently (or which partitions have the highest response times), as well as a history of the partitions that have processed the most transaction up to this time (or history of the partitions that have had the highest response times).

<application_name> and <ejb_name> specify the application and ejb for which to keep PMI statistics.
<partition_count> refers to how many partitions to keep PMI statistics, e.g. if there are 10,000 partitions in the cluster, and the partition_count is set to 20, only the top 20 partitions in terms of transaction count or response times will be stored.
<aggregator_interval> refers to the interval in seconds the aggregator waits between aggregations. The default value is 15000 milliseconds.
<cluster_name> refers to the cluster the application is running. If the PMI specification level for wpfModule is not set to H for any server in this cluster, subscribeWPFPMI will fail.

The subscribeWPFPMI command returns an id integer to the user. The user will use this id in subsequent commands when getting statistics and updating subscribe options.
All of these options can be set separately as well
Example:

[root@wpfsample3 bin]# ./wpfadmin subscribeWPFPMI cumulative TransactionCount WPFKeyBasedPartitionSample WPFKeyBasedPartition 3 15000 --c cluster1
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Cluster set to cluster1
WPFC0040I: WPF PMI has been subscribed with options range=cumulative, type=TransactionCount, application name=WPFKeyBasedPartitionSample, ejb name=WPFKeyBasedPartition, partition count=3, interval=15000
WPFC0041I: Your client id is 2. Use this in future wpfadmin PMI calls.

4.5.17 setPartitionCount

Tells the aggregator server how many partitions to keep PMI statistics for in the bulletin board. For example, if there are 10,000 partitions in the cluster, and the aggregator partition count is set to 20, the top 20 partitions in terms of transaction count and response times will be stored in the bulletin board. The –id option specifies for which subscription to update the partition count.

Usage: ./wpfadmin setPartitionCount 6 –id 12

Example:

[root@wpfsample3 bin]# ./wpfadmin setPartitionCount 4 --id 2
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Id set to 2
WPFC0066I: Partition count has been set to 4 for client id 2

4.5.18 setStatisticsRange

Tells Async PMI to gather statistics cumulatively or to reset the stats after each aggregator interval time period. The options are either cumulative or active. The –id option specifies for which subscription to update the range.

Usage: ./wpfadmin setStatisticsRange active –id 12

Example:

[root@wpfsample3 bin]# ./wpfadmin setStatisticsRange active --id 2
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Id set to 2
WPFC0066I: Statistics range has been set to active for client id 2

4.5.19 setEJBName

Tells PMI to gather statistics for the specified application. The –id option specifies for which subscription to update the ejb name.

Usage: ./wpfadmin setEJBName MyApp#MyEJB –id 12

Example:

WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Id set to 2
WPFC0066I: Ejb name has been set to MyApplication for client id 2

4.5.20 setStatisticsType

Tells Async PMI to gather statistics for transaction count or response time. The options are either TransactionCount or ResponseTime. The –id option specifies for which subscription to update the statistics type.

Usage: ./wpfadmin setStatisticsType ResponseTime –id 12

Example:

[root@wpfsample3 bin]# ./wpfadmin setStatisticsType ResponseTime --id 2
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Id set to 2
WPFC0066I: Statistics type has been set to ResponseTime for client id 2

4.5.21 setStatisticsInterval

The interval in milliseconds the aggregator server waits between aggregations. The –id option specifies for which subscription to update the interval.

<Warning, shorter interval more overhead>

Usage: ./wpfadmin setStatisticsInterval 45000 –id 12

Example:

[root@wpfsample3 bin]# ./wpfadmin setStatisticsInterval 30000 --id 2
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Id set to 2
WPFC0066I: Statistics interval has been set to 30000 for client id 2

4.5.22 getTransactionCount

Returns the latest aggregated data for each of the partitions from the bulletin board regarding the number of transactions processed in the aggregator interval. The number of partitions for which data will be returned is set during the setPartitionCount command. An optional [--top <refresh time>] can be passed in to show a Unix top-like process that updates every <refresh time> seconds with the latest transaction data.

Usage: ./wpfadmin getTransactionCount –top 30 –id 10

Example:

[root@wpfsample3 bin]# ./wpfadmin getTransactionCount --id 2
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Id set to 2

PartitionName	TransactionCount	TotalResponseTime	MinimumTime	MaximumTime
PK000030	6	1755	64	496
PK000026	6	1472	25	998
PK000025	6	3558	424	792

4.5.23 getResponseTime

Returns the latest aggregated data for each of the partitions from the bulletin board regarding the response time of transactions processed in the aggregator interval, along with the minimum, maximum, and average response time. When cumulative is set, the minimum and maximum response time is the minimum and maximum of the whole time, instead of that period, when active is set. The number of partitions for which data will be returned is set during the setPartitionCount command. An optional [-top <refresh time>] can be passed in to show a Unix top-like process that updates every <refresh time> seconds with the latest response time data.
Usage: ./wpfadmin getResponseTime –id 12

4.5.24 setTraceSpec

[Service/diagnostic related] Sets the trace specification for a single server (using --s <server_name>) or all servers (using --c <cluster name>) in the specified cluster. With option temp, the new traceSpec will only work if the servers are currently running. So if the servers are restarted, the traceSpec will go away. With option perm, the new traceSpec will be set permanently. But if the servers are currently started, the new traceSpec will not take effect until the servers are restarted.

Available options:
<temp|perm>. Enables trace while the server is running until the servers are restarted (temp) or after the server is restarted permanently (perm).
<trace spec>. trace specification.
--s <server name>. Server on which to enable trace.
--c <cluster name>. All servers in the given cluster will have the trace enabled.

Usage: ./wpfadmin setTraceSpec perm com.ibm.ws.wpf.*=all=enabled –c cluster1

4.5.25 unsubscribeWPFPMI

Notifies the server to stop collecting WPF Async PMI data when not needed.

Available options: --id <pmi id>. Unsubscribe will be called for the given id.

Usage: ./wpfadmin unsubscribeWPFPMI –id 12

Example:

[root@wpfsample3 bin]# ./wpfadmin unsubscribeWPFPMI --id 2

WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
WPFC0065I: Id set to 2
WPF PMI has been unsubscribed for client id 2

4.5.26 disableWPFPMI

Disables the WPF PMI module.

Available options: --c <cluster name>. PMI will be disabled for all servers in the given cluster.

Usage: ./wpfadmin disableWPFPMI –c cluster1

Example:

disableWPFPMI: Disables the WPF PMI module.
[root@wpfsample3 bin]# ./wpfadmin disableWPFPMI --c c
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager

WPFC0065I: Cluster set to c
WPFC0064I: The wpfModule of PMI is disabled for cluster c

4.5.27 createPolicy

The following scenario is an example of incorporating policies to manage partitions. The wpfadmin command createPolicy will be used to create four policies. The cluster has four servers that will host partitions on four different nodes: NodeA/Server1, NodeB/Server2, NodeC/Server3, and NodeD/Server4. Also each node has a backup server in case the hosting server goes down. These servers are NodeA/BackupServer1, NodeB/BackupServer2, NodeC/BackupServer3, and NodeD/BackupServer4. An application will be installed that creates 40 partitions at startup. The partitions belong to four different classifications: partitions1-10 (classification=class1) to be activated on NodeA/Server1, partitions11-20 (classification=class2) to be activated on NodeB/Server2, partitions21-30 (classification=class3) to be activated on NodeC/Server3, and partitions 31-40 (classification=class4) to be activated on NodeD/Server4. In order to achieve this, four policies will be created: Policy1, Policy2, Policy3, and Policy4. From the application server bin directory the command ./wpfadmin createPolicy /Policy1.properties will be run. Here are the contents of Policy1.properties:

CoreGroupName = DefaultCoreGroup
PolicyType = OneOfNPolicy
PolicyName = Policy1
PolicyDescription = Policy for partitions1-10
IsAlivePeriodSec = 120
QuorumEnabled = false
NumOfMatchCriteria = 2
Name_0 = -gt
Value_0 = -p
Name_1 = -pc
Value_1 = class1
Failback = true
PreferredOnly = true
NodeName_0 = NodeA
ServerName_0 = Server1
NodeName_1 = NodeA
ServerName_1 = BackupServer1

Next run ./wpfadmin createPolicy /Policy2.properties with the following in Policy2.properties:

CoreGroupName = DefaultCoreGroup
PolicyType = OneOfNPolicy
PolicyName = Policy2
PolicyDescription = Policy for partitions11-20

IsAlivePeriodSec = 120
QuorumEnabled = false
NumOfMatchCriteria = 2
Name_0 = -gt
Value_0 = -p
Name_1 = -pc
Value_1 = class2
Failback = true
PreferredOnly = true
NodeName_0 = NodeB
ServerName_0 = Server2
NodeName_1 = NodeB
ServerName_1 = BackupServer2

Next run ./wpfadmin createPolicy /Policy3.properties with the following in Policy3.properties:

CoreGroupName = DefaultCoreGroup
PolicyType = OneOfNPolicy
PolicyName = Policy3
PolicyDescription = Policy for partitions21-30
IsAlivePeriodSec = 120
QuorumEnabled = false
NumOfMatchCriteria = 2
Name_0 = -gt
Value_0 = -p
Name_1 = -pc
Value_1 = class3
Failback = true
PreferredOnly = true
NodeName_0 = NodeC
ServerName_0 = Server3
NodeName_1 = NodeC
ServerName_1 = BackupServer3

Finally run ./wpfadmin createPolicy /Policy4.properties with the following in Policy4.properties:

CoreGroupName = DefaultCoreGroup
PolicyType = OneOfNPolicy
PolicyName = Policy4
PolicyDescription = Policy for partitions31-40
IsAlivePeriodSec = 120
QuorumEnabled = false
NumOfMatchCriteria = 2
Name_0 = -gt
Value_0 = -p
Name_1 = -pc
Value_1 = class4
Failback = true
PreferredOnly = true
NodeName_0 = NodeD
ServerName_0 = Server4
NodeName_1 = NodeD
ServerName_1 = BackupServer4

Notice the match criteria for the four policies. They are set up in name/value pairs. -gt=-p denotes that the group type we're dealing with is partition. -pc=class4 denotes this policy only applies to partitions that have classification "class4".

Now the core group is set up with the four policies. Now start the node agents on nodes A-D. After a short interval, the core group changes will be synchronized to all the nodes. Assuming the application is already installed, when Server1 is started, partitions1-10 will all be activated on Server1 since Policy1 has the preferredOnly set to true. This means partitions1-10 can only be started on Server1 and BackupServer1. If BackupServer1 is started before Server1,

partitions1-10 will be activated on BackupServer1, if neither Server1 or BackupServer1 are started, the partitions will not be activated. This explains how the preferredOnly and preferred servers work. When the failback option is set to true, this means the partitions for the policy will always be activated on the most preferred server. So if partitions1-10 are active on Server1, then Server1 goes down, the partitions will activate on BackupServer1. Then if sometime in the future Server1 comes up, the partitions will go back to Server1, since it is first in the preferred server list.

Consider the following server startup scenario:

Server2 starts - partitions11-20 activate on Server2
Server3 starts - partitions21-30 activate on Server3
BackupServer3 starts - nothing happens, since Server3 is already running and it is before BackupServer3 in Policy3's preferred server list
BackupServer1 starts - partitions 1-10 activate on BackupServer1
BackupServer2 starts - nothing happens, since Server2 is already running and it is before BackupServer2 in Policy1's preferred server list
BackupServer4 starts - partitions 31-40 activate on BackupServer4

Server1 starts - partitions1-10 deactivate on BackupServer1 and activate on Server1, since failback is set to true and Server1 is before BackupServer1 in Policy1's preferred server list
Server4 starts - partitions31-40 deactivate on BackupServer4 and activate on Server4, since failback is set to true and Server4 is before BackupServer4 in Policy4's preferred server list.

4.5.28 updatePolicy

With this command, the user can update any or all of the attributes of the specified policy. Here's a list of each policy type with the attributes that can be updated:

OneOfNPolicy:
-failback <true|false>
-preferredOnly <true|false>
-quorumEnabled <true|false>
-isAlivePeriodSec <period in seconds>
-preferredServers <node1>/<server1>,<node2>/<server2>,…

MOfNPolicy:
-failback <true|false>
-preferredOnly <true|false>
-quorumEnabled <true|false>
-isAlivePeriodSec <period in seconds>
-numActive <number of servers>
-preferredServers <node1>/<server1>,<node2>/<server2>,…

StaticPolicy:
-quorumEnabled <true|false>
-isAlivePeriodSec <period in seconds>
-servers <node1>/<server1>,<node2>/<server2>,…

AllActivePolicy:
-quorumEnabled <true|false>
-isAlivePeriodSec <period in seconds>

NoOpPolicy:
-quorumEnabled <true|false>
-isAlivePeriodSec <period in seconds>

Example:

[root@wpfsample3 bin]# ./wpfadmin updatePolicy "myM-Of-N FP2 Policy" -failback true -preferredOnly false -preferredServers NodeA/Server1,NodeC/Server3
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager

The policy myM-Of-N FP2 Policy has been updated

4.5.29 Delete Policy

Deletes the specified policy from the DefaultCoreGroup

Usage: wpfadmin deletePolicy “myM-Of-N FP2 Policy”

Example:

[root@wpfsample3 bin]# ./wpfadmin deletePolicy “myM-Of-N FP2 Policy”
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
The policy myM-Of-N FP2 Policy has been deleted

4.5.30 updateJMXTimeout

Sets the timeout value for JMX commands. This modifies the coregroup.xml file in ND_HOME/config/cells/<cell_name>/coregroups/DefaultCoreGroup directory.

Example:

[root@wpfsample3 bin]# ./wpfadmin updateJMXTimeout 40
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
The JMX timeout value for the core group has been set to 40 seconds

4.5.31 updateCoreGroupCoordinators

Allows user to adjust number of coordinators for the core group as well as the preferred coordinator servers list. This modifies the coregroup.xml file in ND_HOME/config/cells/<cell_name>/coregroups/DefaultCoreGroup directory.

Available options:
-numCoordinators <number>. Number of active coordinators for the core group.
-preferredCoordinatorServers <server list>. List of servers that will be coordinators for the core group, of the form <node>/<server>,<node>/<server>,<node>/<server>,…

Example:

[root@wpfsample3 bin]# ./wpfadmin updateCoreGroupCoordinators -numCoordinators 4 -preferredCoordinatorServers wpfsample2/s7,wpfsample3/s1,wpfsample3/s9,wpfsample2/s5
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
The coordinators for the core group has been updated

4.5.32 updateHamConfig

Changes the buffer and/or maxThreads size for a server (specified by --s), cluster (specified by --c), or cell (specified by --cell). This command will modify the hamanagerservice.xml for the specified servers located in the ND_HOME/config/cells/<cell_name>/nodes/<node_name>/servers/<server_name> directory.

Available options:
-buffer <buffer size>. Size of the transport buffer.
-maxThreads <maximum threads>. Maximum number of threads for the hamanager thread pool.

--s <server name>. Server name of the form <cell>/<node>/<server>
--c <cluster name>. All servers in the given cluster will be updated.
--cell <cell name>. All servers in the given cell will be updated.

Example:

```
[root@wpfsample3 bin]# ./wpfadmin updateHamConfig -buffer 20 -maxThreads 15 --c c
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector;  The type of
process is: DeploymentManager
WPFC0065I: Cluster set to c
The MaximumSize of the ThreadPool for the hamanagerservice has been set to 15 for Server s4
The TransportBufferSize for the hamanagerservice has been set to 20 for Server s4
The MaximumSize of the ThreadPool for the hamanagerservice has been set to 15 for Server s5
The TransportBufferSize for the hamanagerservice has been set to 20 for Server s5
The MaximumSize of the ThreadPool for the hamanagerservice has been set to 15 for Server s6
The TransportBufferSize for the hamanagerservice has been set to 20 for Server s6
The MaximumSize of the ThreadPool for the hamanagerservice has been set to 15 for Server s7
The TransportBufferSize for the hamanagerservice has been set to 20 for Server s7
The MaximumSize of the ThreadPool for the hamanagerservice has been set to 15 for Server s8
The TransportBufferSize for the hamanagerservice has been set to 20 for Server s8
The MaximumSize of the ThreadPool for the hamanagerservice has been set to 15 for Server s1
The TransportBufferSize for the hamanagerservice has been set to 20 for Server s1
The MaximumSize of the ThreadPool for the hamanagerservice has been set to 15 for Server s2
The TransportBufferSize for the hamanagerservice has been set to 20 for Server s2
The MaximumSize of the ThreadPool for the hamanagerservice has been set to 15 for Server s3
The TransportBufferSize for the hamanagerservice has been set to 20 for Server s3
The MaximumSize of the ThreadPool for the hamanagerservice has been set to 15 for Server s9
The TransportBufferSize for the hamanagerservice has been set to 20 for Server s9
```

4.5.33 listPolicies

Lists the name of all of the policies in the core group.

Example:

```
[root@wpfsample3 bin]# ./wpfadmin listPolicies
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector;  The type of
process is: DeploymentManager
Policies in DefaultCoreGroup
JetStream Policy
WPF Cluster Scoped Partition Policy
WPF Node Scoped Partition Policy
WPF PMI Aggregator Policy
```

4.5.34 queryPolicy

Shows the attributes of the given policy.

Example:

```
[root@wpfsample3 bin]# ./wpfadmin queryPolicy “WPF Cluster Scoped Partition Policy”
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector;  The type of
process is: DeploymentManager
Attributes for policy WPF Cluster Scoped Partition Policy:
name: WPF Cluster Scoped Partition Policy
description: Default WPF Cluster Scoped Partition Policy
policyFactory: com.ibm.ws.hamanager.coordinator.policy.impl.OneOfNPolicyFactory
isAlivePeriodSec: -1
quorumEnabled: true
failback: false
preferredOnly: false
preferredServers:
```


[]
MatchCriteria:
-gt=-p

4.5.35 resolvePolicyForGroup

Shows the name of the policy for each partition.

Available options:
--o <number of partitions> . The number of partitions that are printed out. If the --o is not specified, it will default to 50 partitions.
--a <application name>. Prints out policy information for the partitions in the given application.
--p <partition name>. Prints out policy information for the given partition.
--class <classification name>. Prints out policy information for the partitions in the given classification.
Example:

[root@wpfsample3 bin]# ./wpfadmin resolvePolicyForGroup --class ClassA
WASX7209I: Connected to process "Deployment Manager" on node wpfsample3Manager using SOAP connector; The type of process is: DeploymentManager
Partition=PK000001
Policy name=WPF Cluster Scoped Partition Policy

Partition=PK000002
Policy name=WPF Cluster Scoped Partition Policy

Partition=PK000003
Policy name=WPF Cluster Scoped Partition Policy

Partition=PK000004
Policy name=WPF Cluster Scoped Partition Policy

Partition=PK000005
Policy name=WPF Cluster Scoped Partition Policy

Partition=PK000006
Policy name=WPF Cluster Scoped Partition Policy

Partition=PK000007
Policy name=WPF Cluster Scoped Partition Policy

Partition=PK000008
Policy name=WPF Cluster Scoped Partition Policy

Partition=PK000009
Policy name=WPF Cluster Scoped Partition Policy

Partition=PK000010
Policy name=WPF Cluster Scoped Partition Policy

4.6 Performance Monitoring

Performance monitoring is an activity in which you collect and analyze data about the performance of your applications and their environment. The product collects data on run-time and applications through the Performance Monitoring Infrastructure (PMI). This infrastructure is compatible with and extends the JSR-077 specification.

In WebSphere Extended Deployment, we create a PMI module for each partition active in the cluster. We also provide features to query the most active partitions in a user specified period.

An example using the WPF Performance Monitoring functionality is provided in the Management section. This example sequence demonstrates the WPFKeyBasedPartitionSample.ear application shipped with WebSphere Extended Deployment, and the source code is included.

4.6.1 WPF PMI Enablement

Before gathering WPF PMI data, you have to enable WPF PMI. The WPF PMI data is recorded in the system only when WPF PMI is enabled. WPF PMI can be enabled in several ways. Using the wpfadmin utility command, the Administrative Console, or the wsadmin tool can enable it. This support must be enabled before the cluster is started.

4.6.1.1 Enabling WPF PMI from the wpfadmin command

You can call "wpfadmin enableWPFPMI *level* --c *cluster*" to enable WPF PMI in a particular cluster. This is the simplest way and the recommended way to enable WPF PMI.

Steps:

- Change directory to <WAS_DEPLOYMENT MANAGER_HOME>/bin directory
- run the following command:

wpfadmin enableWPFPMI H --c <your cluster name>
You will see the following output:

```
WPFC0065I: Cluster set to was-cluster-1
WPFC0043I: The wpfModule of PMI is enabled for cluster was-cluster-1 and set
to level H.
```

Now you've successfully enabled WPF PMI.

4.6.1.2 Enabling WPF PMI from the admin console.

Or, if you prefer enable WPF PMI from admin console, you can follow the following steps to Here are the steps to enabling the WPF PMI for a particular server:

1. Open the administrative console.
2. Click **Servers** > **Application Servers** in the console navigation tree.
3. Click *server*.
4. Click the **Configuration** tab.
5. When in the Configuration tab, settings will apply once the server is restarted. When in the Runtime Tab, settings will apply immediately. Note that enablement of Performance Monitoring Service can only be done in the Configuration tab.
6. Click **Performance Monitoring Service**.
7. Select the checkbox **Startup**.
8. Add wpfModule=H to **initial specification level** field.
9. Click **Apply** or **OK**.
10. Click **Save**.
11. Restart the application server.

The changes you make will not take affect until you restart the application server.

If you want to enable WPF PMI for an entire node, you can follow the same steps except changing the navigation from **Servers>Application Servers>*server*** to System **Administration>Node Agents>*node agent***.

You can also use JMX MBean or wsadmin tool to enable the WPF PMI. This is not different from enabling other PMI modules. For more details, please refer to the WebSphere Application Server Info Center.

4.6.2 WPF PMI path

As mentioned before, there is one WPF PMI module for each partition. The Partition PMI is grouped by the application name and the EJB name. So if users want to query the PMI data for Partition p1 in Session EJB session1 of Application app1, the PMI path is “wpfModule,app1#session1,p1”. Using this PMI path, users can directly query the PMI data for a specific partition using wsadmin or JMX MBean just like you would normally do in the application server.

4.6.3 WPF PMI data aggregation

Since partitions are activated in different application servers, PMI data for all partitions are possibly spread in different application servers in a cluster. Most likely, users are interested in the N most active partitions in the whole cluster. WPF provides a PMI aggregation and sorting feature to satisfy this requirement.

WPF elects one server as the PMI aggregator, which aggregates the partition statistics data from all servers. The PMI aggregator is an HA Group with one of N policy. The policy name is called “WPF PMI Aggregator Policy”. Users can customize the policy using wpfadmin tool. At any time, there is only one PMI aggregator active in the core group. This aggregator is responsible for aggregating the statistics for all the partitioned applications. The best practice is to let this PMI aggregator run on a separate server from those housing partitions so it will not slow down business execution. Each application server is responsible for publishing all of its partition PMI data when PMI is turned on. The aggregator will run at a user-specified interval, thus cutting down the volume of stats the user will be shown. The PMI aggregator will mainly do the following:

1. Aggregate the statistics from different servers for each partition. Since partition can be moved from one server to another, the PMI data for a partition can be spread over different application servers. The PMI aggregator must gather all the PMI data for a partition from all servers.
2. Sort the PMI statistics based on what users want. For example, if users are interested in the most active (biggest number of transaction counts) 50 partitions over last 15 seconds. The PMI aggregator will calculate the PMI statistics from the last 15 seconds and sort them based on the transaction count.
3. Publish the sorted PMI statistics results.

Users can specify the statistics interests when they subscribe the WPF PMI statistics. These interests include application name and session EJB name, statistics type, statistics range, partition count, and aggregation interval. They can also changes these interest after they subscribe them.

4.6.4 WPF PMI statistics subscription

If users are interested in the PMI statistics of a particular partition or a server, they do not need to subscribe the WPF PMI statistics. They can directly retrieve it using wsadmin tool or the JMX MBean.

If users are interested in the aggregated or sorted PMI statistics, they have to subscribe the WPF PMI statistics using wpfadmin command or the WPF JMX MBean.

4.6.4.1 Subscribe WPF PMI statistics using wpfadmin command

Users can call the following wpfadmin command to subscribe the WPF PMI statistics:

```
wpfadmin.bat|wpfadmin subscribeWPFPMI
    STATISTICS_RANGE
    STATISTICSS_TYPE
    APPLICATION_NAME
    SESSIONEJB_NAME
    PARTITION_COUNT
    AGGREGATION_INTERVAL
    --c CLUSTER_NAME
```

where:
STATISTICS_RANGE: the statistics range. It could be cumulative or active.
STATISTICSS_TYPE: the statistics type. It could be TransactionCount or ResponseTime.
APPLICATION_NAME: the application name.

SESSIONEJB_NAME: the Session EJB name.
PARTITION_COUNT: the number of partitions that users are interested in.
AGGREGATION_INTERVAL: the statistics aggregation interval.
CLUSTER_NAME: the cluster in which the application runs. If the PMI specification level in any server of this cluster is not set to H, subscribeWPFPMI will fail.

The output of this command will display the subscription client ID. This client id is very important. It is the handle to your subscribed PMI interest. Any changes to this PMI interest or PMI data retrieval will require this handle.

Steps:
For example, we are interested in the top 5 partitions that have had the longest total response time in the past. We also want this PMI data will be updated every 60 seconds. We can execute the following command to subscribe the PMI interest:

```
wpfadmin subscribeWPFPMI cumulative ResponseTime WPFKeyBasedPartitionSample
WPFKeyBasedPartition 5 60000
```

You will see the following output:

```
WPFC0040I: WPF PMI has been subscribed with options range=cumulative,
type=ResponseTime, application name=WPFKeyBasedPartitionSample, ejb
name=WPFKeyBasedPartition, partition count=5, interval=60000
WPFC0041I: Your client id is 1. Use this in future wpfadmin PMI calls.
```

And the client id, as displayed, is 1.

4.6.5 WPF PMI statistics retrieval

If users are interested in the PMI statistics of a particular partition or a server, they can directly retrieve it using wsadmin tool or the JMX MBean.

If users have subscribed WPF PMI statistics using the wpfadmin command or WPFJMX MBean, they can use wpfadmin to retrieve the statistics:

- `wpfadmin.bat|wpfadmin getTransactionCount --id` *`PMI_SUBSCRIPTION_ID`* `--top` *`REFRESH_INTERVAL`*`.`
- `wpfadmin.bat|wpfadmin getResponseTime --id` *`PMI_SUBSCRIPTION_ID`* `--top` *`REFRESH_INTERVAL`*`.`

The PMI_SUBSCRIPTION_ID is the client ID you get when you subscribe the WPF PMI. If you use --top option, the output will be updated every specified refresh interval just like the Unix top command.

Also, they can also use Java code, jacl script or jython script to get the WPFJMX MBean to get the statistics similar to the examples shown in 10.5.2.

Steps:

- execute the following command to get the PMI data:
  `wpfadmin getResponseTime --id 1`
  You will see the following output
  `WPFC0065I: Id set to 1`

  PartitionName TotalResponseTime TransactionCount MinimumTime MaximumTime StartTime SumOfSquares

  `WPFC0045I: No statistics are available! Please wait and try again.`

  You don't see any statistics data. That's because there are no PMI data in the system yet. Now we can run the client to generate some statistics.
- Open a new command window or shell, execute the following command:
  `<WAS_HOME>/bin/launchClient <WAS_HOME>/installedApps/<CELL_NAME>/WPFKeyBasedPartitionSample.ear -CCproviderURL=orbaloc::`*`HOSTNAME`*`:`*`SERVER_RMI_PORT`*

where, HOSTNAME is the name of the host that has the application server., and SERVER_RMI_PORT is the RMI port of that server, for example, 9811.
You will see client runs with a similar output like this:

```
IBM WebSphere Application Server, Release 5.1
J2EE Application Client Tool
Copyright IBM Corp., 1997-2003
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class
com.ibm.websphere.wpf.client.WPFKeyBasedPartitionCl
ient
Create Partitions from PK000001 to PK000010
1st call: PK000001->partiton=PK000001,server=Tiger1/clus1srv1
2nd call: PK000001->partiton=PK000001,server=Tiger1/clus1srv1
3rd call: PK000001->partiton=PK000001,server=Tiger1/clus1srv1
1st call: PK000002->partiton=PK000002,server=Tiger1/clus1srv1
2nd call: PK000002->partiton=PK000002,server=Tiger1/clus1srv1
3rd call: PK000002->partiton=PK000002,server=Tiger1/clus1srv1
1st call: PK000003->partiton=PK000003,server=Tiger1/clus1srv1
2nd call: PK000003->partiton=PK000003,server=Tiger1/clus1srv1
3rd call: PK000003->partiton=PK000003,server=Tiger1/clus1srv1
1st call: PK000004->partiton=PK000004,server=Tiger1/clus1srv1
2nd call: PK000004->partiton=PK000004,server=Tiger1/clus1srv1
3rd call: PK000004->partiton=PK000004,server=Tiger1/clus1srv1
1st call: PK000005->partiton=PK000005,server=Tiger1/clus1srv1
2nd call: PK000005->partiton=PK000005,server=Tiger1/clus1srv1
3rd call: PK000005->partiton=PK000005,server=Tiger1/clus1srv1
1st call: PK000006->partiton=PK000006,server=Tiger1/clus1srv1
2nd call: PK000006->partiton=PK000006,server=Tiger1/clus1srv1
3rd call: PK000006->partiton=PK000006,server=Tiger1/clus1srv1
1st call: PK000007->partiton=PK000007,server=Tiger1/clus1srv1
2nd call: PK000007->partiton=PK000007,server=Tiger1/clus1srv1
3rd call: PK000007->partiton=PK000007,server=Tiger1/clus1srv1
1st call: PK000008->partiton=PK000008,server=Tiger1/clus1srv1
2nd call: PK000008->partiton=PK000008,server=Tiger1/clus1srv1
3rd call: PK000008->partiton=PK000008,server=Tiger1/clus1srv1
1st call: PK000009->partiton=PK000009,server=Tiger1/clus1srv1
2nd call: PK000009->partiton=PK000009,server=Tiger1/clus1srv1
3rd call: PK000009->partiton=PK000009,server=Tiger1/clus1srv1
1st call: PK000010->partiton=PK000010,server=Tiger1/clus1srv1
2nd call: PK000010->partiton=PK000010,server=Tiger1/clus1srv1
3rd call: PK000010->partiton=PK000010,server=Tiger1/clus1srv1
```

- 3.Wait for 60 seconds to let the aggregator sort the statistics, and then execute “wpfadmin getResponseTime –id 1” again, you will see the following output.

```
WPFC0065I: Id set to 1

PartitionName       TotalResponseTime  TransactionCount  MinimumTime  MaximumTime
StartTime     SumOfSquares
PK000004            1625               3                 87           903
1094180666309 1226203
PK000010            1543               3                 260          671
1094180667170 892385
PK000005            1423               3                 51           965
1094180666389 1099475
PK000008            932                3                 99           482
1094180666990 365326
PK000002            862                3                 217          364
1094180666038 258546
```

Notice that the TotalResponseTime in your output might be different from the one here, since the response time is generated as a random value.

4.6.6 WPF PMI statistics parameters

As mentioned before, a WPF statistic has several parameters. You have used them in the subscribeWPFPMI command.

4.6.6.1 application name and session EJB name

Partitions are grouped by the application name and the session EJB name. When subscribing the WPF PMI statistics, users have to specify the application name and the session EJB name.

Users can use "`wpfadmin.bat|wpfadmin setEJBName` *`EJB_NAME`* `[--a` *`APPLICATION_NAME`*`] --id` *`PMI_SUBSCRIPTION_ID`*" to set the EJB name and the application name for one PMI subscription. –a is used to specify the application name.

Users can choose using fully qualified EJB name or not. For example, if users want to change the EJB name to "session2" and application name to "app2", they can execute either of the following commands.

- `./wpfadmin setEJBName app2#session2 --id 1`
- `./wpfadmin set EJBName session2 --a app2 --id 1`

User can also use Java code, jacl script, or jython script to get the com.ibm.websphere.wpf.jmx.WPFJMX MBean instance, and change the statistics type via the MBean. For how to get an instance of MBean in different ways, please refer to section 5.5.2.

Steps:

- Execute "`wpfadmin setEJBName WPFKeyBasedPartitionSample#NonExistentSessionBean --id 1`". Notice `NonExistentSessionBean` EJB doesn't exist in the application.
- Wait for 60 seconds, and retrieve the statistics by executing "`wpfadmin getResponseTime --id 1`". You will not see any statistics as expected.
- Execute "`wpfadmin setEJBName WPFKeyBasedPartitionSample#WPFKeyBasedPartition --id 1`".
- Wait for 60 seconds, and retrieve the statistics by executing "`wpfadmin getResponseTime --id 1`". You will see the statistics again.

4.6.6.2 statistics type

There are two types of partition statistics users are interested in: transaction count or response time. Transaction count tracks the number of transactions in a certain period. Response time tracks the total response time of all transactions over a certain period. WPF has its own WPF PMI module, called wpfModule, which can be enabled to track these two statistics. There is only one TimeStatistics in this module, since transaction count can also be tracked by TimeStatistics objects.

Both transaction count and response time will be displayed when you retrieve the statistics. This statistics type is used for sorting purpose. When statistics type is set to transaction count, the statistics are sorted by transaction counts, and vice versa.

Users can use "`wpfAdmin.bat|wpfadmin setStatisticsType [TransactionCount|ResponseTime] --id` *`PMI_SUBSCRIPTION_ID`*" to set the statistics type for one PMI subscription.

User can also use Java code, jacl script, or jython script to get the com.ibm.websphere.wpf.jmx.WPFJMX MBean instance, and change the statistics type via the MBean. For how to get an instance of MBean in different ways, please refer to section 5.5.2.

Steps:

- Execute "`wpfadmin setStatisticsType TransactionCount --id 1`" to change the statistics type to TransactionCount.
- Wait for 60 seconds, and retrieve the statistics by executing "`wpfadmin getTransactionCount --id 1`". You will get the following output:

```
PartitionName       TransactionCount  TotalResponseTime  MinimumTime  MaximumTime
StartTime     SumOfSquares
PK000010            3                 1543               260          671
1094180667170 892385
```



```
PK000009            3                369                21          242
1094180667070 70241
PK000008            3                932                99          482
1094180666990 365326
PK000007            3                396                25          200
1094180666829 69866
PK000006            3                392                75          236
1094180666759 67882
```

You can see the statistics doesn't change much from the last output except the orders of TransactionCount and TotalResponseTime column are exchanged. However, TransactionCount, instead of TotalResponseTime, sorts this output.

4.6.6.3 StatisiticsStatistics range

There are two types of statistics range users are interested in: cumulative statistics or active statistics. Cumulative statistics track statistics from transaction one in a partition. Active statistics track statistics over the recent period. For example, users can track how many transactions happened in the latest 15 seconds.

Users can use "`wpfAdmin.bat|wpfadmin setStatisticsRange [cumulative|active] --id PMI_SUBSCRIPTION_ID`" to set the statistics range for one PMI subscription.

User can also use Java code, jacl script, or jython script to get the com.ibm.websphere.wpf.jmx.WPFJMX MBean instance, and change the statistics range via the MBean. For how to get an instance of MBean in different ways, please refer to section 5.5.2.

Steps:

- Execute "`wpfadmin setStatisticsRange active --id 1`" to change the statistics range from cumulative to active.
- Retrieve the statistics by executing "`wpfadmin getTransactionCount --id 1`". You will get the following output:

```
PartitionName       TransactionCount  TotalResponseTime  MinimumTime  MaximumTime
StartTime     SumOfSquares
PK000010            0                 0                  260          671
1094180667170 0
PK000009            0                 0                  21           242
1094180667070 0
PK000008            0                 0                  99           482
1094180666990 0
PK000007            0                 0                  25           200
1094180666829 0
PK000006            0                 0                  75           236
1094180666759 0
```

This is because the active statistics are calculated to record the transaction statistics in the last 60 seconds. You haven't generated any transactions in the last 60 seconds. That's why you get the transaction count and total response time 0.

- open a command window, run the client to generate the transactions again:

```
<WAS_HOME>\bin\launchClient
<WAS_HOME>/installedApps/<CELL_NAME>/WPFSimpleSample.ear -
CCproviderURL=orbaloc::HOSTNAME:SERVER_RMI_PORT
```

- Keep retrieving the statistics by executing "`wpfadmin getResponseTime --id 1`" until you get the following output.

```
PartitionName       TransactionCount  TotalResponseTime  MinimumTime  MaximumTime
StartTime     SumOfSquares
PK000010            3                 1695               232          827
1094180667170 1142249
PK000009            3                 2089               21           899
1094180667070 1574733
PK000008            3                 1616               99           887
1094180666990 1140270
PK000007            3                 1832               25           871
1094180666829 1228214
PK000006            3                 2210               75           895
1094180666759 1690502
```


- Change the range from active to cumulative by executing "`wpfadmin setStatisticsRange cumulative --id 1`"
- (optional) If you retrieve the statistics again, you will see a different output with transaction count as 6.

4.6.6.4 partition count

Users can also set the number of partitions they are interested in. For example, they are interested in the 50 partitions that have the largest number of transaction counts.

Users can use use "`wpfAdmin.bat|wpfadmin setPartitionCount` *`PARTITION_COUNT`* `--id` *`PMI_SUBSCRIPTION_ID`*" to set the partition count for one PMI subscription.

User can also use Java code, jacl script, or jython script to get the com.ibm.websphere.wpf.jmx.WPFJMX MBean instance, and change the partition count via the MBean. For how to get an instance of MBean in different ways, please refer to section 5.5.2.

Steps:

- Execute "`wpfadmin setPartitionCount 8 --id 1`" to change the partition count to 8.
- Keep a record of current time, and then retrieve the statistics by executing "`wpfadmin getTransactionCount --id 1`" until you see 8 partitions in the output. Calculate how many seconds has passed.

4.6.6.5 statistics aggregation interval

As mentioned earlier, the PMI aggregator only aggregates and sorts the PMI statistics in intervals. This interval is called the statistics aggregation interval.

Users can use "`wpfAdmin.bat|wpfadmin setStatisticsInterval` *`INTERVAL`*`--id` *`PMI_SUBSCRIPTION_ID`*" to set the partition count for one PMI subscription.

User can also use Java code, jacl script, or jython script to get the com.ibm.websphere.wpf.jmx.WPFJMX MBean instance, and change the statistics aggregation interval via the MBean. For how to get an instance of MBean in different ways, please refer to section 5.5.2.

Steps:

- Execute "`wpfadmin setStatisticsInterval 10000 --id 1`" to change the statistics interval to 10 seconds.
- Execute "`wpfadmin setPartitionCount 9 --id 1`" to change the partition count to 9.
- Keep a record of current time, and then retrieve the statistics by executing "`wpfadmin getResponseTime --id 1`" until you see 9 partitions in the output. Calculate how many seconds has passed. You will notice that it is much quicker to show 9 partitions than the last time to show 8 partitions.

4.6.7 WPF PMI Aggregator policy

The PMI Aggregator is a HA Group member, so you can configure the HA Group parameters, such as which server the aggregator is running, whether the aggregator runs in preferred servers, etc.

Users can use the following wpfadmin command to configure the aggregator policy:

```
wpfadmin.bat|wpfadmin setAggregatorPolicy
--failback [true|false]
--preferredOnly [true|false]
--numOfPolicyServers NUMBER_OF_SERVERS
--servers SERVER_LIST
```

where:

- failback: whether the aggregator is failedback. The recommended value is true.
- perferredOnly: Whether the aggregator is only running in preferred server list or not.
- numOfPolicyServers: the number of the servers which can house the PMI aggregator
- servers: The preferred server list separated by comma.

4.7 Scalability Related Configuration

This section is intended to point out what configuration properties may need to be changed to handle application which allocate 5,000 – 30,000 partitions. In general, customers should keep the number of actual partitions to a minimum to ensure the cluster performance is not degraded. In addition, each attribute column has a foot note explaining how the attribute can be set.

4.7.1 Configuration

For a maximum configuration IBM has tested, 10,000 partitions spread across 30 application servers, here are the recommended configuration changes.

Update core group coordinator settings using **wpfadmin updateCoreGroupCoordinators**. Set the -numCoordinators option to 4, and set the -preferredCoordinatorServers to be on different physical nodes as described earlier.

Update HA Manager service transport buffer setting for all servers in the cell using **wpfadmin updateHamConfig**. Set the -buffer option to 20 and set the --cell to the name of the cell.

Change the maximum heap size for all servers in the cell to 512 and set the Deployment Manager's maximum heap size to as big as possible (1400 if you have the physical memory). This can all be done through the WebSphere GUI and requires restarting all modified servers.

If you see OutOfMemory, JMXTimedOut, and other errors when running wpfadmin commands, try increasing the number of coordinators for the core group. This may be seen with large numbers of partitions.

Suggested Configuration for Various partition counts:

Partition Count	# Cooridnators[5]
1000	2
5,000	2
10,000	3[6]

4.8 Proxy DataSource Management

If you want to use Proxy DataSource support, you need to create the proxy datasources from the administrative console, or using wsadmin tool. WebSphere Extended Deployment ships two JDBC providers for Proxy DataSource support. One is called “Proxy DataSource JDBC Provider”, which is for connecting to non-XA-capable data sources. The other is called “Proxy DataSource JDBC Provider (XA)”, which is for connecting to XA-capable data sources.

Please refer to section 4.4.5.2 for how to create a proxy datasource for your application to use.

[5] This option is set via wpfadmin.

[6] Note, use the HA Manager policy attribute to collocate coordinators exclusively on application servers located on different physical nodes (see wpfadmin).

5 WebSphere Partitioning Facility Programming

This section describes the WPF programming environment, and how to interact with WPF programming when using the WebSphere Studio Application Developer product. For those simply looking for a summary of the WPF and Proxy Datasource APIs, they can refer to the published javadoc:

<Deployment Manager Home>\web\xd\apidocs\index.html

The goal in this document is to enable a programmer, who is already familiar with EJB development how to create Partitioned Applications. Because WPF is a very manageable environment, and the developer should understand HA Manager policies and how the administrator may manage a WPF environment, it is strongly suggested the programmers review the earlier sections of this guide to understand the capability and review the examples located in the following directory for WPF examples:

<Deployment Manager Home>\installableApps\WPF*

In addition, the Proxy Datasource sample is also included at that location, and is packaged in the ProxyDSAccountSample.ear.

Earlier in this document, there is a description of each of the sample programs and an example that steps through the configuration of a cluster, installing and finally using the example Partitioned J2EE WPFKeyBasedPartition application. The examples include the source code and can be imported into WebSphere Studio for review.

WPF requires the use of one PSSB EJB and optionally allows you to use other session EJBs as partition routable session EJBs (PRSBs) or non-routable facade session beans to PSSB beans.

5.1 Partitioned EJB Overview

The WebSphere Partitioning Facility (WPF) strives to offer the promise of high transaction rate computing, but also allows the developer to continue using the existing J2EE programming model. Other than adding WPF Framework related APIs, the developer's entity bean implementation should be fundamentally the same as with any other J2EE Application and can use the same toolset.

When the WPF Framework APIs are used, and the Partitioned Stateless Session Bean implements the WPF Framework APIs, the bean will take on a new level of function and have enhancement management functions available to the administrator.

5.1.1 Partitioned Stateless Session Bean (PSSB)

A partitioned stateless session bean (PSSB) must implement the WPF Framework API, and utilize the PartitionManager interface to create and manage partitions. In addition a <Bean>_PartitionKey.java class must be implemented is called by the workload routing to determine the partition endpoint the request should be routed to.

5.1.1.1 PSSB <Bean>_PartitionKey Routing Class

The programmer implementing WPF must implement a <Bean>_ParititionKey class to direct where the method request should be routed within the cluster. The endpoint will be a partition, and is described using a string. For example, in the WPF various samples, the following classes with the _PartitionKey are implemented and appear similar to the following.


```
package com.ibm.websphere.wpf.ejb;

/**
 * PartitionKey for Partitioned Stateless Session Bean WPFKeyBasedPartition
 */
public class WPFKeyBasedPartition_PartitionKey {


  /**
   * return the partition string as the partition key
   * @param partition
   * @return
   */
  public static String buy(String partition) {
    return partition;
  }

}
```

In this case, (from the WPFFacadePartitionSample PSSB) the bean only has a single static remote method, **buy(String)** which has the partition destination passed in. The user will receive all method argument passed to the signature, and can process them as required to determine which appropriate partition endpoint cluster member should host the work.

All methods in the remote interface of the PSSB should be implemented.

5.1.1.2 The PSSB Bean needs to have its generated stub updated

The wpfstubutil is a utility that regenerates the stub, and inserts the appropriate interfaces to enable the <Bean>_PartitionKey class to be called for each route method execution. This tool should be called each time after the EJB is deployed. The deployment process will result in the file being rewritten without these modifications.

An additional concern is administrators should never “deploy” when installing a Partitioned J2EE Application within the cluster, as this will reset the generated stub, and the WPF Framework required changes will be present at runtime. For examples of the updated stub, the samples in the installabeApps directory beginning with D_* can be reviewed.

5.1.2 Partition Routable Session Bean (PRSB)

A partition routable session bean (PRSB) is a stateless session bean packaged in the same EJB module (jar) with the PSSB, but does not implement the required PSSB interfaces. The PSSB must implement the interfaces as defined above.

The PRSB would implement the business methods and the PSSB would not specify them. There can be more than one of PRSBs implemented in the jar. This is useful if the application has several sets of partitions for multiple purposes (the partition sets must be named uniquely) and multiple session beans implementing separate pieces of a give application.

The advantage is only the business method interfaces need to be present in the PRSB (verses all the “plumbing”, e.g. the APIs supporting the WPF Framework like getPartitions, partitionUnloadEvent(…), etc…). Since the PSSB bean is collocated, the routing will be based upon the PSSB, but does enable the application writer to avoid using the PSSB implementation directly to make the code more readable. All administration functions however will utilize the PSSB bean name, this is just an abstraction.

As a reminder, the PSSB partitions for that Partitioned J2EE Application will be associated with the PSSB. For example, the –pn Group Member property (see the HA Manager section) will be the name of the PSSB bean, versus the PRSB. However, this should not cause a problem.

The programmer must use the wpfstubutil on the PRSB just as they would with the PSSB if remote methods are to be routable to the correct partition on the server.

5.1.3 Facade Interface for a Partitioned Stateless Session Bean

A PSSB and PRSB can directly provide an interface for remote method invocations to a remote server infrastructure (or locally as well if within the same server side JVM). However, often programmers would like to have a single facade to these sorts of WPF Framework bean types, and the server implementation would execute the PSSB and PRSB bean

functionality. This is not only for EJB programmers, but also for those implementing servlets for example. To implement this, we have provided a simple example, the WPFFacadePartitionSample.

This sample caches the remote home of an example PSSB bean, and provides a single, non-routable method implementation that executes the PSSB method on the server JVM. The source code is included in the WPFFacadePartitionSample.ear.

This approach does have an advantage. If the client uses only a facade interface, and the PSSB/PRSB client routing portion is executed in the server infrastructure, the general routing functionality will be faster as the client routing state information does not have to be downloaded and cached in the client JVM making the request.

For example, the WPFKeyBasedPartitionSample.ear sample client (WPFKeyBasedPartitionClient.java) uses JNDI and directly instantiates the PSSB home instance, and creates an instance from it. Each remote method invocation on the instance requires the client JVM to acquire routing information, download the client JVM, and then cache it. The download step can add overhead to the client implementation in terms of performance (the routing data is cached and stored in memory for both cases so there is not a memory footprint savings). For the case where a user solution has thousands of partitions and many clients in separate client JVMs, bottlenecks can arise when transferring this much information and if possible is better to be avoid.

However, in summary, both Façade and non-Facade approaches are supported and useful in certain scenarios.

5.1.4 WPF Requirements

This section describes key requirements to ensure a PSSB has valid WPF Framework implementation.

5.1.4.1 PSSB Local Interfaces

When implementing a PSSB, it must be a Stateless Bean and the following interfaces must be used:

Interface	Ejb-jar descriptor	Interface
Local Home	**local-home**	**com.ibm.websphere.wpf.PartitionHandlerLocalHome**
Local	**local**	**com.ibm.websphere.wpf.PartitionHandlerLocal**

For example, the ejb-jar.xml for WPFKeyBasedPartition is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
"http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar id="ejb-jar ID">
  <display-name>WPFKeyBasedPartitionEJB</display-name>
  <enterprise-beans>
    <session id="WPFKeyBasedPartition">
      <ejb-name>WPFKeyBasedPartition</ejb-name>
      <home>com.ibm.websphere.wpf.ejb.WPFKeyBasedPartitionHome</home>
      <remote>com.ibm.websphere.wpf.ejb.WPFKeyBasedPartition</remote>
      <local-home>com.ibm.websphere.wpf.PartitionHandlerLocalHome</local-home>
      <local>com.ibm.websphere.wpf.PartitionHandlerLocal</local>
      <ejb-class>com.ibm.websphere.wpf.ejb.WPFKeyBasedPartitionBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <env-entry>
        <description>The number of partitions this session bean will create</description>
        <env-entry-name>NumberOfPartitions</env-entry-name>
        <env-entry-type>java.lang.Integer</env-entry-type>
        <env-entry-value>10</env-entry-value>
      </env-entry>
    </session>
  </enterprise-beans>
</ejb-jar>
```

5.1.4.2 Partition Router Object (<EJBName>_PartitionKey class)

The application programmer is responsible for creating a partition router that acts as a callback for WPF. WPF uses this callback to determine the target partition for requests. The class, which is a plain old java object (POJO), must follow these rules:

- It must reside in the same package as the EJB it supports
- The class name must be the same as the EJB and have the string _PartitionKey appended to the name
- There must be a static method for each remote method on the EJB. These methods should match the signature of each method on the EJB but should always return a String representing the EJB's partition name. WPF calls these methods when a remote method is called on the EJB.
- The method should not throw any checked exceptions.
- The customer must run wpfstubutil on the deployed ear with the partition router object to "WPF" enable for partition routing

5.1.5 WPF Restrictions

5.1.5.1 Partition Names must be Cluster Unique

Cluster Scoped Partition names must be unique within the cluster they are hosted within. For example, if a Partitioned J2EE Application creates a partition named "PartitionX", another application in the same cluster cannot create another partition with the same name.

One way to address this, is to use the PartitionManager.getApplicationName() interface, and if carefully used (specifically do not use long strings), can preappend the partition name with deployed application name to the partition name. For example, assume the administrator deployed the application with "App45v1r2", the createPartitionDefinition(…) API could preappend the App45v1r2 to PartitionX, and create a partition named "App45v1r2_PartitionX".

Node Scoped Partitions names within the same application will be the same for each application on each logical node, however, Node Scoped Partition names across different Partitioned Applications also need to be unique.

5.1.5.2 Other Restrictions

Please refer to each WPF subsystem to review specific restrictions (such as the Proxy Datasource).

5.2 Developing WPF applications with WSAD 5.1

This section describes how to use WSAD to create an Partitioned J2EE Application. This section will ensure WSAD is enabled correctly, and allow a developer to create a new Partitioned J2EE Application or work with a WPF framework example, modify it, repackage, update the Partitioned J2EE Application with the wpfstubutil and finally run the example.

For those wishing to start from scratch creating a PSSB, please refer to the next section

5.2.1 Preparing WSAD to Develop WPF Partitioned J2EE Applications

This example assumes you wish to create a PSSB (to include within a new Partitioned J2EE Application).

5.2.1.1 Getting Started with a new WPF Partitioned Application

Create a J2EE project with an EJB module and an application client. We need to add the wpf.jar from the project to the EJB module project. Right click on the ejb module and select the properties menu item.

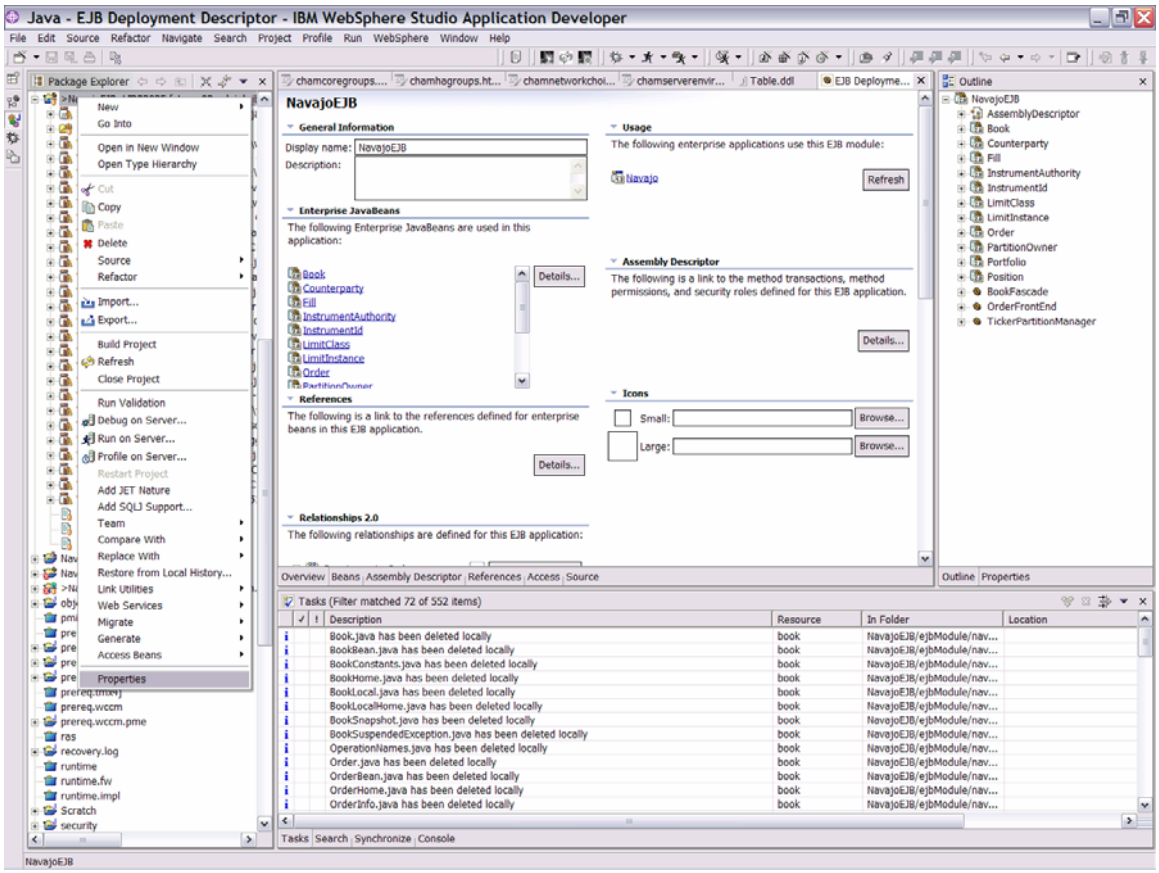


Figure 2 Selecting the project properties

This makes the following dialog appear.

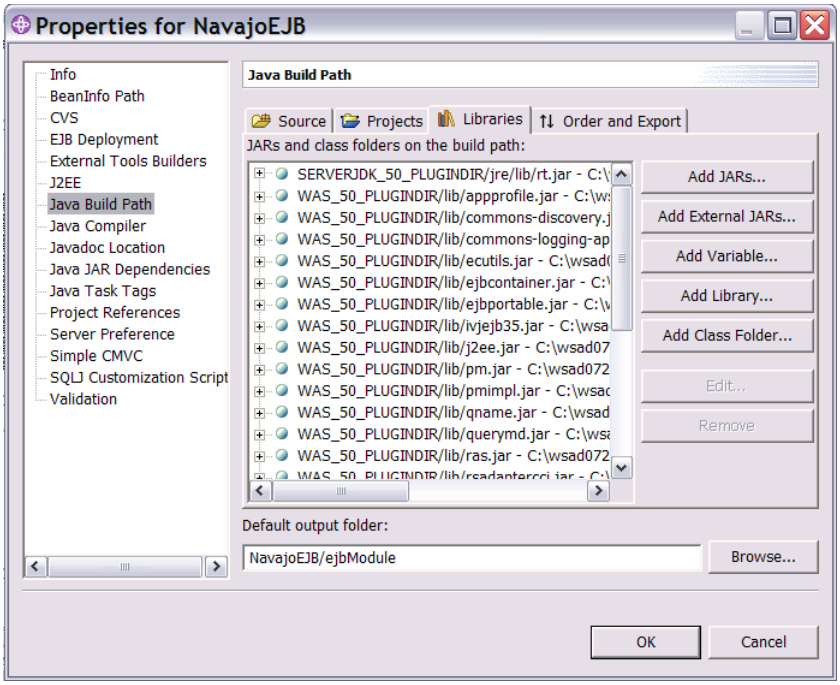


Figure 3 Editing the project class path

The next step is to click on the “Add Variable” button to define a variable indicating where you have installed the Extended Deployment product.

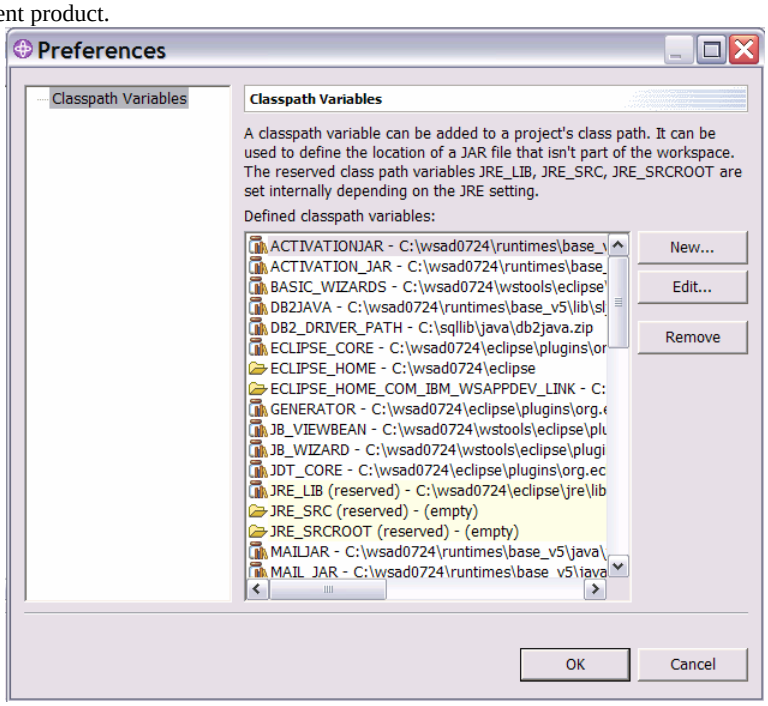


Figure 4 Variable editing screen

Now click on the “New…” button and we will make the XD_HOME variable.

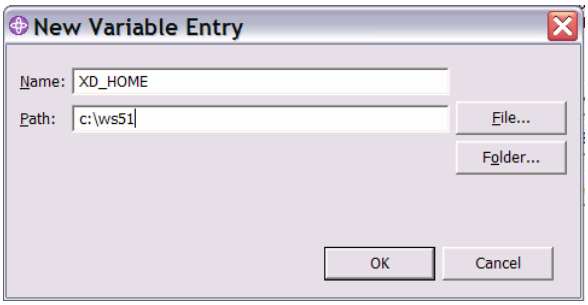


Figure 5 Add the XD_HOME variable

Click OK. Now you should see:

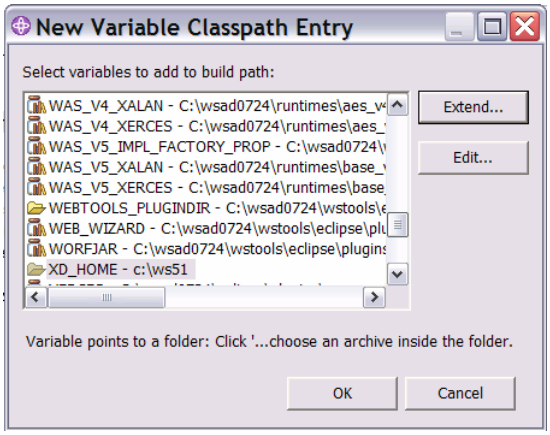


Figure 6 Variables added

There is now an XD_HOME variable. Click on the “Extend…” button. This allows you to add a jar based on this variable to your project class path. The use of a variable means that when multiple developers are using CVS then each developer can set XD_HOME to a different directory pretty easily and still use the project easily when Extended Deployment may be installed in different directories.

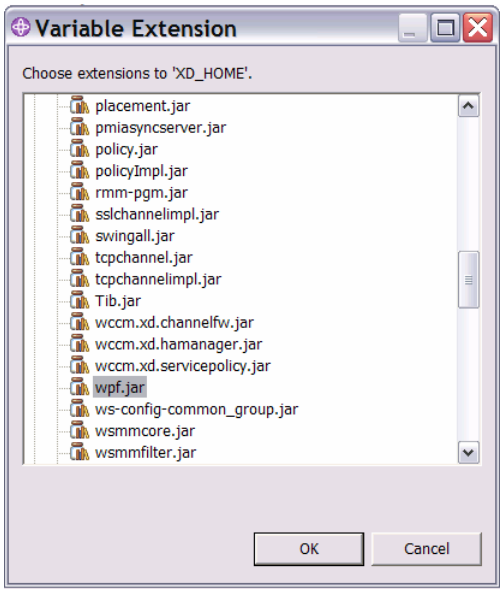


Figure 7 Selecting the required jars for the project

Expand the lib subdirectory and then select the wpf.jar and click OK. If you need any other jars then select those also. The other jars typically needed are asynchbeans.jar from the lib directory if you have also installed WBI-SF.

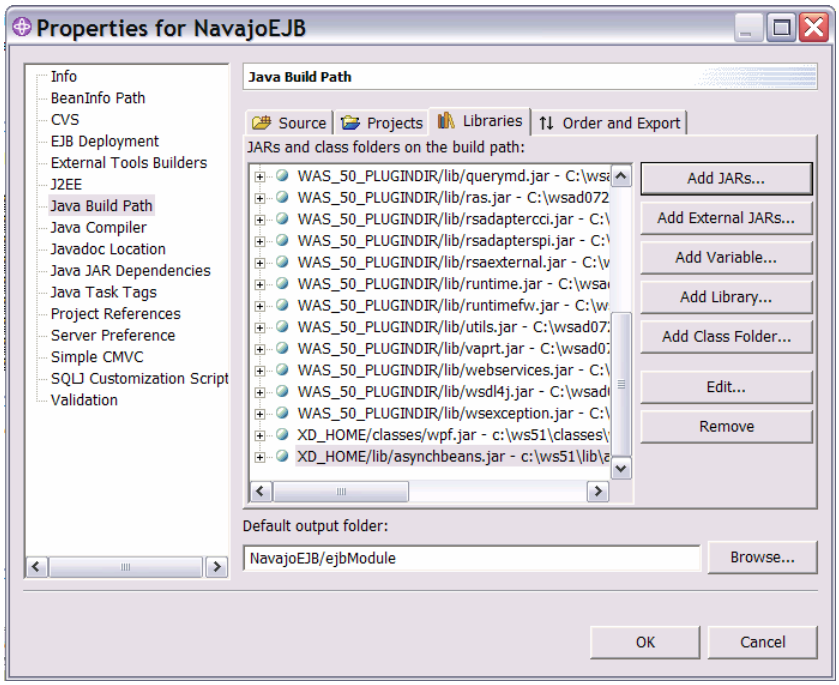


Figure 8 The finished build path screen

This shows what the build path would look like when both the wpf and asynchbeans jars are added to the classpath. Now click “OK” and your classpath is set correctly. If you are using any additional jars in third party products such as Log4J etc then they should be added using a similar approach. Define a variable and then extend it.

5.2.1.2 Creating a Partition Stateless Session Bean (PSSB) Example.

The PSSB is the center of a WPF application. It allows the application server to query the application at startup to determine which partitions the application requires. It also is used by the application server to inform the application when a partition is activated or deactivated. Activated means the HA Manager has assigned a partition to this cluster member (See the section on HA Manager policies to see how the HA Manager assigns partitions to a cluster member).

So, let us add a PSSB to your EJB module. Right click on the EJB project and click on “New”. This brings up the following screen that allows you to choose to make an Enterprise bean.

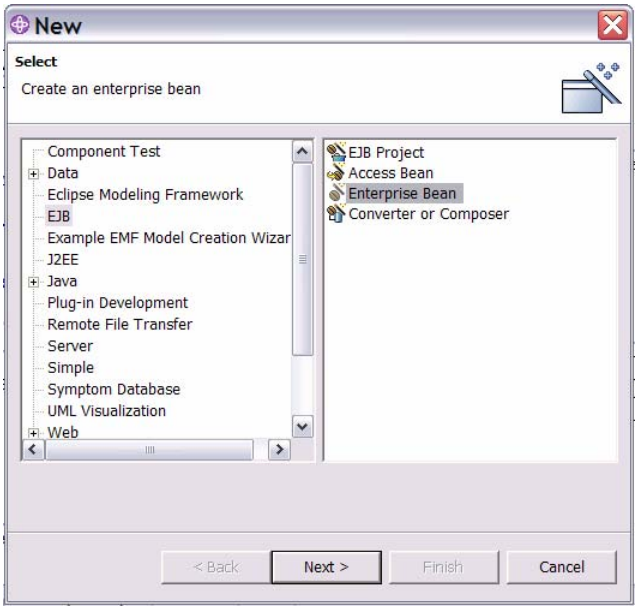


Figure 9 Making a new EJB

Click on the EJB item on the left of the dialog and then click on “Enterprise Bean” on the right. Now, click on “Next>”. You can now choose the ejb module you want to host your PSSB. The module selected should already be the correct one since that is what you clicked on to get to these dialogs. Now click on Next. We now choose that we want to make a session bean as follows:

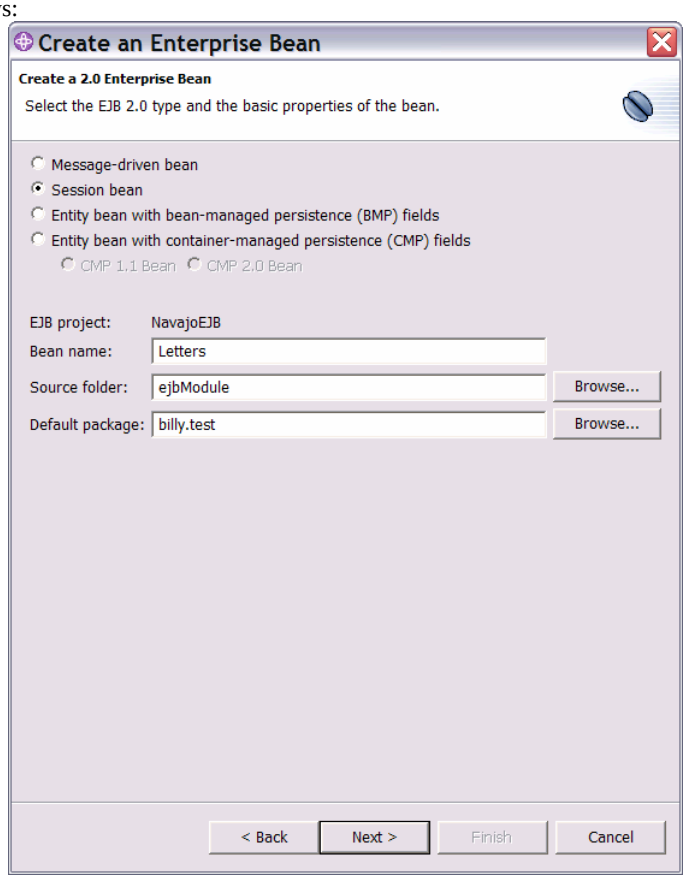


Figure 10 Creating the Partition Stateless Session Bean (PSSB)

The following dialog should now appear:

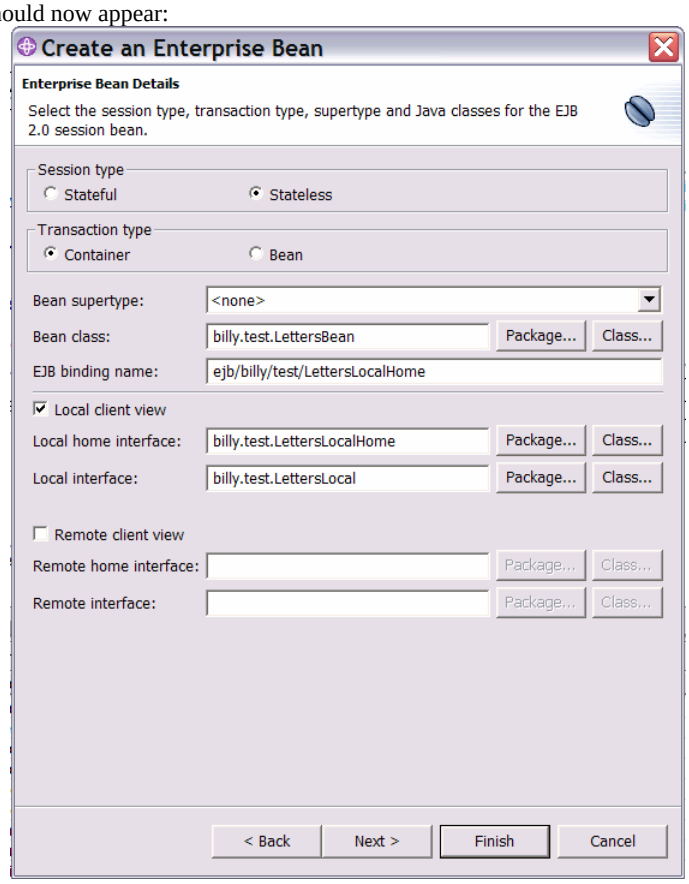


Figure 11 About to specify the local interfaces for the PSSB

The PSSB is required to use a particular local interface and local home interface. We will now override the defaults shown above and use the correct interfaces. Click on the "Class…" button next to local home interface. This brings up the following dialog.

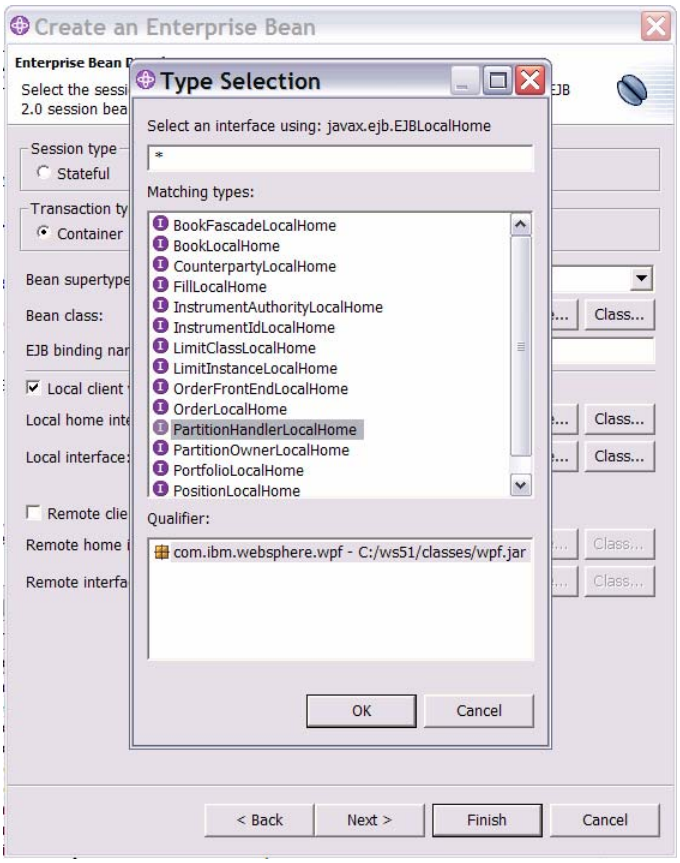


Figure 12 Selecting the PSSB home interface

Scroll down and select the PartitionHandlerLocalHome interface. You can see the package name "com.ibm.websphere.wpf" from the wpf.jar file we added to the build path previously. This dialog only shows interfaces that are usable as local home interfaces. Click OK. Now, click on the "Class…" button beside the local interface and then select the "PartitionHandlerLocal" interface and again click OK. The dialog should now look like this:

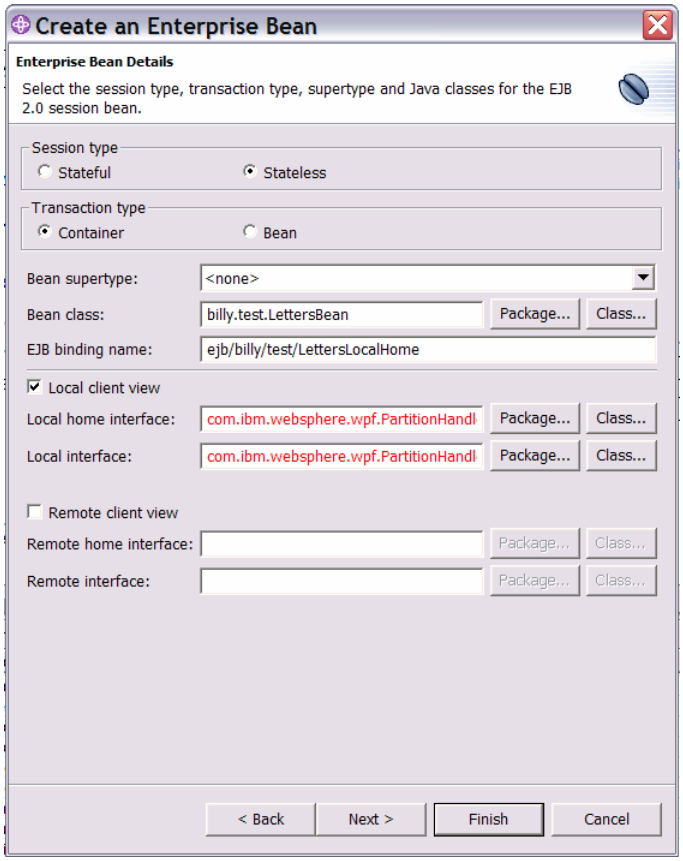


Figure 13 Completed PSSB creation dialog

Now click on the "Finish" button. The PSSB is now created and included in your EJB module. Next, we need to add the methods from the PartitionHandler to the implementation bean for the EJB. There is no built-in mechanism for doing this easily, but there is a trick we can use to get WSAD to add the methods very easily. Find the implementation bean you just added. The sample so far creates the PSSB in the billy.test package. Find this in your project and double click on LettersBean (or whatever you called your PSSB). Now change the LettersBean to also implement the PartitionHandlerLocal interface and choose Save. Then, right click on the class name in the outline and generate the missing methods on the interfaces.

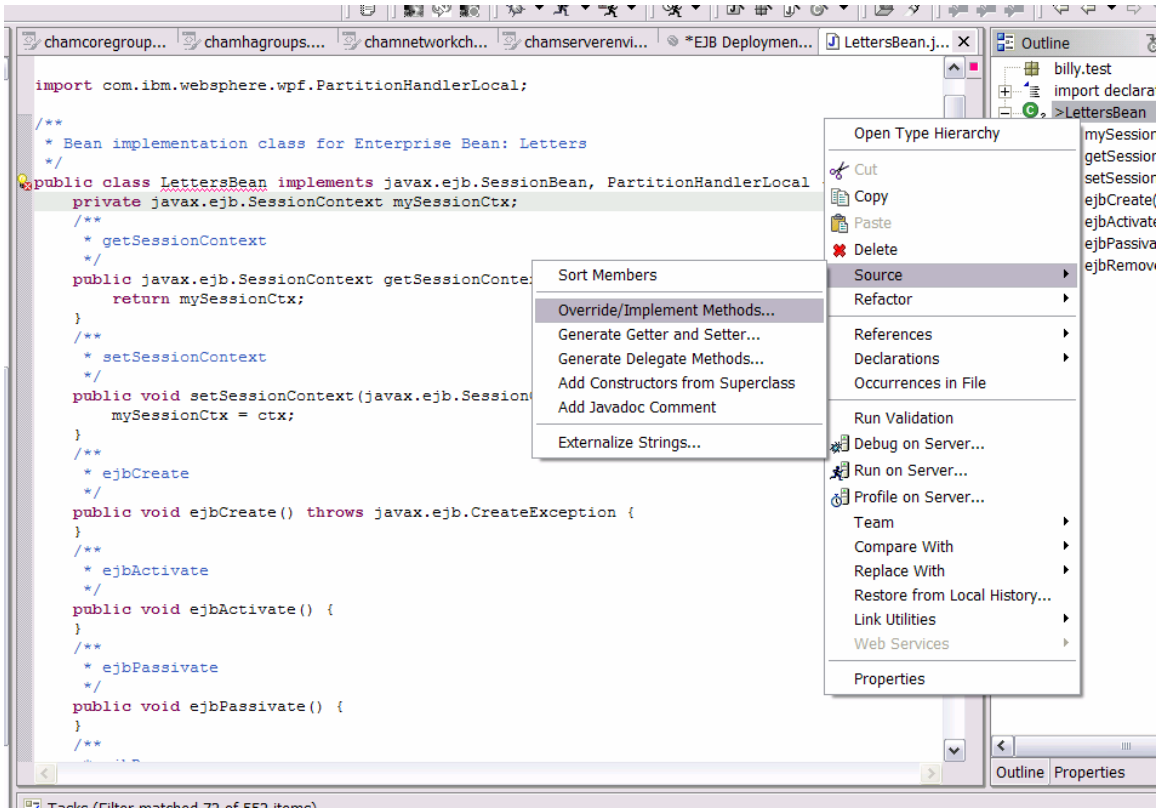


Figure 14 Adding the PSSB methods automatically.

Now you should see the following dialog:

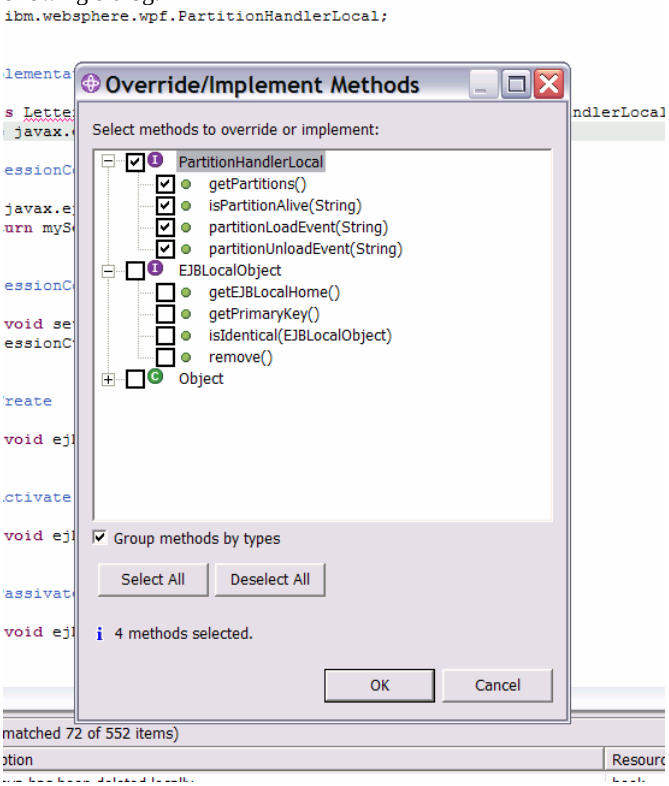


Figure 15 Selecting the PSSB methods to add to your bean

Clear the checkbox for the EJBLocalObject and then click OK. This adds the methods from the local interface in one easy step. Finally, remove the PartitionHandlerLocal interface from the LettersBean and save it. Your PSSB is now ready for customization. The screen should now look something like the following:

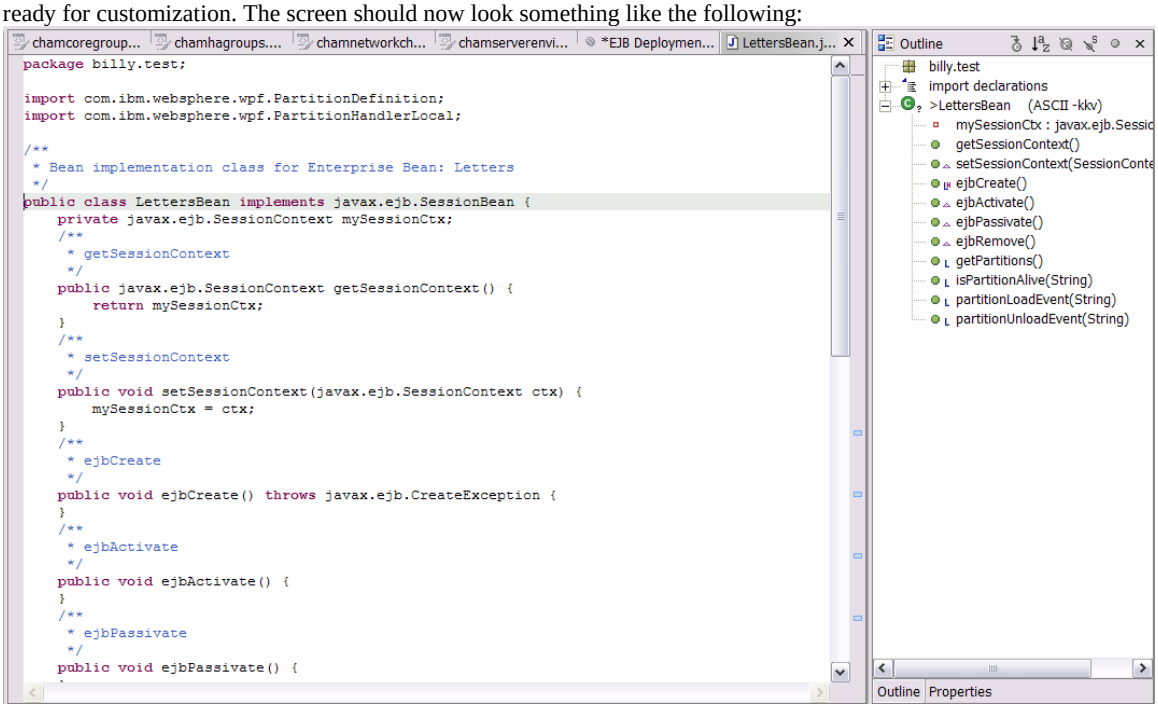


Figure 16 Completed PSSB with PSSB methods

Here you can see the 4 new methods added to the bean and you will see we removed the PartitionHandlerLocal interface from the bean implements list.

5.2.1.3 Adding some sample partitions

Add the following variable to the PSSB.

```
PartitionManager ivRuntime;
```

You'll also need to add the following import.

```
import com.ibm.websphere.wpf.*;
```

Update your setSessionContext method to look like the following:

```
public void setSessionContext(javax.ejb.SessionContext ctx) {
  mySessionCtx = ctx;
  try
  {
    InitialContext ic = new InitialContext();
    ivRuntime = (PartitionManager)ic.lookup(PartitionManager.JNDI_NAME);

  }
  catch(Exception e)
  {
    throw new EJBException("Problem initializing PSSB", e);
  }
}
```

This retrieves the PartitionManager service that is used to create data structures needed for partitioning.

5.2.1.3.1 Update the getPartitions method

Now, update the getPartitions method as follows:


```
public PartitionDefinition[] getPartitions()
{
  ArrayList partitions = new ArrayList();

  // first add some key based partitions. These are static here but could
  // just as easily be loaded from a database when the application starts.
  String[] keys = new String[] {"IBM", "CSCO", "SUNW", "BEAS", "ORCL", "MSFT", "GE"};
  for(int i = 0; i < keys.length; ++i)
  {
    PartitionDefinition p = ivRuntime.createPartitionDefinition("K_" + keys[i],
            "KEYS", PartitionScope.K_CLUSTER);
    partitions.add(p);
  }
  // now add some partitions for our hash test. We'll have a hash space of 16 slots
  // thus allowing us to scale to 16 JVMs. We can easily use a larger number but 16
  // is sufficient for this sample.
  for(int i = 0; i < 16; ++i)
  {
    PartitionDefinition p = ivRuntime.createPartitionDefinition("H_" + i,
            "HASH", PartitionScope.K_CLUSTER);
    partitions.add(p);
  }
  PartitionDefinition p = ivRuntime.createPartitionDefinition("SINGLETON");
  partitions.add(p);
  return (PartitionDefinition[])partitions.toArray();
}
```

This method shows an example of three types of partitioning schemes. The first creates a partition per ‘key’. The keys here are stock symbols. You could load the list of stock symbols (there are usually thousands) and return partition definitions for each of those.
The next scheme shows how partitions can be created to hash a set of keys on to a fixed set of partitions using a hashing scheme. The sample uses sixteen partitions.
Finally, we show how to make a singleton service for the cluster by giving it a single partition. The purpose of the PSSB is simply to allow the application to specify the set of partitions at startup time. Each partition specified allows the cluster member to be a potential activation member for the partition. Each cluster member should normally return the same set of partitions and the customer should use the HA Manager and its policies to determine partition placement. This offers both flexibility and simplicity by separating the concerns of partition definition versus partition placement (there is no need to implement complicated application logic returning different partition sets for each cluster member).
Once the cluster starts then each partition is activated on one of the cluster members. The partition

5.2.1.3.2 Update the isPartitionAlive method

Partitions may take advantage of the partition health checking mechanism. The
The partition’s isPartitionAlive method is invoked by the HA Manager to interrogate the application to check the state of a partition. If the application can determine that a partition is healthy, then it should return true. Otherwise, it should return false. The partition’s isPartitionAlive method will be called when:

- The partition is activating.
  The partitionLoadEvent method has been called but has not yet returned. The isPartitionAlive must return true during this period.
- The partition is active.
  The partitionLoadEvent returned and the cluster member is now hosting the active partition and receiving IIOP requests (or requests through other means). The isPartitionAlive method will be called periodically to check the partition health.
- The partition is deactivating.
  The partitionUnloadEvent has been called and the application is currently shutting down the partition on this cluster member.

```
public boolean isPartitionAlive(String partitionName)
{
  return isMyParititionAlive(partitionName);
}
```


5.2.2 Partitioned J2EE Application Example

This example explains how to start from one of the examples provided with WPF.

5.2.2.1 Importing the WPFKeyBasedPartitionSample Sample Application

In the <ND Install Image Location>\installableApps you will see the WPFKeyBasedPartitionSample.ear. This ear has not been deployed and contains the source for the WPF example. The D_ WPFKeyBasedPartitionSample.ear is a deployed example and for administrators who wish to load and begin experimenting with the WPF framework.

To import the sample:

- Start WSAD
  - Create a new Project
  - Select Enterprise Application Project
  - Select J2EE 1.3 Enterprise Application Project
  - Next
  - Determine a project name, e.g. “test”
  - Click Finish
- In the J2EE Hierarchy window
  - Select the “test” project
  - Hold the right mouse, select import
  - Select “Import…”
  - In the Import dialog, select “EAR file”
  - Select Next
  - Browse to the <Deployment Manager>\installableApps directory
  - Select an WPF example ear, for example, WPFKeyBasedPartitionSample.ear
  - Select Finish

At this point, normal developers can be done. The application can be changed, programmer can regenerated the deployed code with the Generate Deployment Code option and exported into a new ear file.

After the ear file is exported, wpfstubutil should be ran and reprocess the EJB stub. This command is documented later in this section.

Finally, the application can be installed (or reinstalled) in the cluster. The example in the initial portion of this text illustrates the steps to install a WPF Partitioned J2EE Application.

5.3 WPF Framework Programming Model

This section describes the programming model for the WebSphere Partition Facility (WPF). There are three aspects to the programming model.

- Partition Stateless Session Bean (PSSB).

  This is the stateless session bean used to obtain the initial set of partitions from the application at start time as well as send events such as partition activated, deactivated and isPartitionAlive to the application at the appropriate times.

- Partition Manager.

  This allows an application to create new application partitions and shutdown existing partitions. It can return the current set of application partitions in use. It allows the application to report transactions to the runtime.

- JMX commands.

The specific APIs are installed in the following directory:

<Deployment Manager Home>\web\xd\apidocs\index.html

5.3.1 PartitionDefinition

Applications create ParitionDefinition instances to describe a partition to the partition runtime.

NOTE: PartitionDefinition instances can only be created using the factory methods of the provided PartitionManager interface.

5.3.1.1 PartitionDefinition#getPartitionName

This method returns the name of the partition definition.

The method signature is:

```
String getPartitionName()
```

The partition name must be unique through the entire cluster. The application writer must ensure their partition names are unique for this version of WPF.

5.3.1.2 PartitionDefinition#getPartitionClass

This method returns the class name of the partition definition.

The method signature is:

```
String getPartitionClass()
```

5.3.1.3 PartitionDefinition#getScope

This method returns the Partition Scope of the partition definition.

The method signature is:


```
PartitionScope getPartitionScope()
```

The Scope will either be cluster or node scoped, see PartitionScope for more information.

5.3.1.4 PartitionDefinition#getAttributeMap

This returns the attribute map of the partition definition if provided when partition definition was created (this is optional). When creating an HA Group the user can create a set of attributes, which are useful in policy management. Map keys and values can only be Strings.

The method signature is:

```
Map getPartitionMap()
```

The default is an empty map with no entries.

5.3.1.5 PartitionDefinition#setPartitionAlias

This method is reserved for future use in the WPF Framework.

The method signature is:

```
void setPartitionAlias(String aliasName)
```

5.3.1.6 PartitionDefinition#getPartitionAlias

This method is reserved for future use in the WPF Framework.

The method signature is:

```
String getPartitionAlias()
```

5.3.2 PartitionScope

PartitionScope is used to specify a partition as having either cluster or node scope during PartitionDefinition creation.

5.3.2.1 PartitionScope#K_CLUSTER

This specifies cluster level scope. This means the HA Manager will apply the policy to all running cluster members.

5.3.2.2 PartitionScope#K_NODE

This specifies node level scope. This means the HA Manager will apply the policy only to the cluster members on a particular node. If a one of N policy is used, then this means one cluster member per node will be activated. If IIOP routing is being used then the requests will be spread over the active cluster members on the various nodes.

5.3.3 PartitionManager

An instance of the PartitionManager can be obtained by the application from JNDI. An instance is bound into JNDI by the runtime. The following code segment shows how the PSSB can look up an instance and then cache it using an instance variable.

```
public void setSessionContext(javax.ejb.SessionContext ctx) {
```



```
        mySessionCtx = ctx;
        try
        {
                // cache various references.
                InitialContext ic = new InitialContext();
                bookHome = (PartitionManager)ic.lookup(PartitionManager.JNDI_NAME);
        }
        catch(Exception e)
        {
                throw new EJBException(e);
        }
}
```

See the sample programmers for more example usage.

5.3.3.1 PartitionManager#JNDI_NAME

This attribute can be used by an application to retrieve the PartitionManager service using JNDI.

5.3.3.2 PartitionManager#createPartitionDefinition

This method has three signatures, and each is used to create a PartitionDefinition instance representing an application partition. This method is overloaded and the appropriate method should be chosen depending on the desired behavior.

The first method signature is:

```
PartitionDefinition createPartitionDefinition(String partitionName)
```

This is the creates a PartitionDefinition with the default classification and cluster scope. The default classification string is the value of the PartitionDefinition#DEFAULT_CLASSIFICATION ("_DFLT"). This version cannot create a unique classification per partition, specify Node Scoped partitions or provide a MAP to better manage the partitions.

The second method signature is:

```
PartitionDefinition createPartitionDefinition(String partitionName,
                            String partitionClass,
                            PartitionScope scope)
```

This creates a PartitionDefinition with the specified partition classification (partitionClass) and a specific partition scope. This version does not provide a Map of policy attributes to create the partition definition. Providing a map of extra policy attributes can offer better partition management options.

The method signature is:

```
PartitionDefinition createPartitionDefinition(String partitionName,
                            String partitionClass,
                            PartitionScope scope,
                            Map attributemap)
```

This creates a PartitionDefinition with the specified partition classification (partitionClass) and a specific partition scope. In addition, a set of attribute and value pairs can be added to the default list of HA Manager group properties for this partition. The extra attributes can be used to control the HA Manager policies more specifically. One caveat, the Map attributes are using during the runtime, so the minimizing the number and the actual data in use is important.

The first value inserted in the Map put() API is the HA Manager "key", the second will be the HA Manager "value" for that "key" pair.

An example when using the optional map parameter can be implemented something similar to the following:

```
public PartitionDefinition[] getPartitions()
{
 String names = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
 PartitionDefinition[] rc = new PartitionDefinition[names.length()];
  for(int i = 0; i < names.length(); ++i)
  {
    Map testMap = new HashMap();
    testMap.put("custom-attrib", "samplevalue");
    testMap.put("activateOn", Integer.toString(i%3));  // have 3 servers, balance at start
    rc[i] = ivRuntime.createPartitionDefinition(names.substring(i, i + 1),
                                                "MapExamle",
                                                PartitionScope.K_NODE,
                                                testMap);
      testMap = null;

    }
    logger.trace(RASITraceEvent.TYPE_LEVEL1, this, "getPartitions", "Returning partitions", rc);
    return rc;
  }
```

In this case, a Map was created that had one HA Manager key attribute set to the same value for all partitions (custom-attrib), and a second that is named “activateOn” and had a string value 0-2. This example was used in conjunction with the policy support to create a policy for each start value and request the preferred server policy be unique for every third partition. See policy sections for more detail on this approach.

5.3.3.2.1 createPartitionDefinition() Attribute Constraints

The attributes provided to the createPartitionDefinition() API must comply with the following rules, or during the create operation an HA Manager failure exception “HAMapException” will be thrown and the Partition will not be created correctly. Not only do these apply to the Map attribute’s key and value pairs, but also the Partition Name and Classification (in this case the user will be providing the values for WPF defaulted key values –pn and –pc respectively).

- The map cannot be null or empty

  (WPF guarantees against this case as WPF provides default Map values which include the Partition Name, Default Classification and other attributes documented in the HA Manager Policy section so in general this rule should not be a concern).
- All keys in the map must be Strings.
- All values in the map must be Strings.
- No **key** is allowed to start with the underbar ('_') character.
- No **key or value** is allowed to contain the comma (',') character.
- No **key or value** is allowed to contain the equals ('=') character.
- No **key or value** is allowed to contain the vertical bar ('|') character.

5.3.3.3 PartitionManager#getPartitions

This method can be called to get a list of the known partitions for this cluster member.

The method signature is:

```
String[] getPartitions()
```

5.3.3.4 PartitionManager#getApplicationName

This method can be called to get the application name determined by the administrator at deploy time. This string can be used in the name of the partition to ensure it is unique within the cluster (the deployed application name must be unique within a websphere cluster).

The method signature is:

```
String getApplicationName()
```

As a caveat, if this is used when creating a partition, we suggest you deploy the Partitioned J2EE Application with a minimal name in terms of string length, as the name is used throughout the HA Manager runtime.

One common use, can be to use this to version partition references, having the same Partitioned J2EE Application in the same cluster, simply installed with a different name, and each partition, e.g. “PartitionA” can instead be called “Appv2r2.PartitionA”. Thus, in the cluster, the user could have “App**v2r1**.PartitionA” and “App**v2r2**.PartitionA” active concurrently. The application would simply address their request to the version they are compatible with. This is just one approach to this problem.

5.3.3.5 PartitionManager#addPartition

This method is called by an application to dynamically add a new partition.

The method signature is:

```
void addPartition(PartitionDefinition name)
```

If a cluster member adds a partition then it is also asynchronously added to all current cluster members automatically. Using the Default Cluster Scoped Policy, the partition will be typically be activated often on the cluster member receiving the request, although this is not guaranteed behavior. In addition, activating a partition across a cluster is a distributed function, thus, a time delay is reasonable between the time the API returns and the actual activation of the partition occurs. Programmers should account for this.

This API is demonstrated in the WPFKeyBasedPartition example.

5.3.3.6 PartitionManager#removePartition

This method allows an application to remove a partition dynamically. If the partition is currently active on any cluster member then it is also deactivated. The partition is removed across all online cluster members.

The method signature is:

```
void removePartition(String name)
```

Deactivating a partition across a cluster is a distributed function, thus, a time delay is reasonable between the time the API returns and the actual deactivation of the partition occurs. Programmers should account for this.

For scenarios where a temporary error is suspect rather than a permanent error, the programmer should consider using the disablePartition() API. In this case, the policy mechanism can be used to determine if the partition should actually not restart, or in fact could be restarted on another cluster member if the opportunity exists.

5.3.3.7 PartitionManager#disablePartition

This dynamically disables a partition. Calling this method disables the partition on the current cluster members. This can also be accomplished via the wpfadmin command.

The method signature is:

void removePartition(String name)

When the method is invoked, the partition will either enter a deactivated state or activate immediately on another cluster member capable of hosting the partition. The partition would be deactivated depending on the current policy

settings, but by default activated on another cluster member if one is available. If partition is not automatically reactivated, wpfadmin can be used to enable the member once the administrator can review the logs and determine any action that should be done prior to reenabling the partition.

The partitionUnloadEvent(...) is not called as the application will call this method and can clean as much as is possible prior to the invocation.

5.3.3.8 PartitionManager#reportPartitionFault

This method is called by the application to indicate a problem with an active partition. This causes the HA Manager to react according to the value of the severity parameter. This method is reserved for future use, and not implemented at this time. See disablePartition() to take corrective action.

5.3.3.9 PartitionManager#reportTransactionComplete

This method is called by an application to report a transaction has just completed with a specific response time for a particular partition. This is normally used when the application is using an asynchronous method for receiving work requests as opposed to synchronous IIOP requests.

The method signature is:

```
void reportTransactionComplete(String partitionName, long responseTime_ms)
```

This API result is stored and transferred to the WPF Performance Monitoring facility. The WPF Performance Monitoring facility will track the individual results across the entire cluster. This service is discussed in several parts of this document, and examples are also included in the getting started section.

This API is demonstrated in the WPFKeyBasedPartitionSample example.

5.3.3.10 PartitionManager#setHttpPartitionManager

This indicates to WPF that this application uses HTTP Partitioning. See the HTTP Partitioning section of this document.

The method signature is:

```
void setHttpPartitionManager(HttpPartitionManagerInterface manager)
```

5.3.4 PartitionManager# reportTransactionComplete

WPF PMI contains one statistic, responseTime, which is used to measure the response time of transactions and transaction number executed on behalf of a specific partition. The response time is collected by users applications. User applications update the PMI statistics by calling the reportTransactionComplete method of the com.ibm.websphere.wpf.PartitionManager interface. Here is the method signature in the PartitionManager interface:

```
   /**
    * This should be called to inform the runtime when a transaction/operation
    * completes on this partition. This is used for records how many
    * transactions per seconds are being executed per partition.
    * @param partitionName the name of the partition.
    * @param responseTime_ms the response time for the transaction.
    * @throws IllegalStateException If the calling application has no partition handler
 bean
    */
       void reportTransactionComplete(String partitionName, long responseTime_ms)
```

Users can call PartitionManager.reportTransactionComplete in the Partition Stateless Session Bean. The best practice is that users calculate the transaction time of a transaction and then use this API to report the transaction time. Here is one example. Method buy is a method to simulate the transaction in the WPFKeyBasedPartition Bean in the

WPFKeyBasedPartitionSample application.

```
/**
     * A buy method. This method does nothing now except report the transaction
     * completion. The transaction takes a random value ranging from 0 ms to 1000ms.
     *
     * @param partitionName
     * @return the partition name
     */
    public String buy(String partitionName) {
            String serverName =
  AdminServiceFactory.getAdminService().getNodeName()+"/"+AdminServiceFactory.getAdminServ
  ice().getProcessName();

      ivManager.reportTransactionComplete(partitionName, (long) (1000 * Math.random()));

      logger.trace(
        RASITraceEvent.TYPE_LEVEL1,
        this,
        "buy",
        "The method called at " + serverName + "." + partitionName);

      return "partiton=" + partitionName + ",server=" + serverName;
    }
```

The first time the PartitionManager.reportTransactionComplete is called for a particular partition in one application server, a PMI module is created for this partition grouped by the application name and the session EJB name. For example, if the application name is “app1”, and the Session EJB name is “session1”, the PMI module is grouped by “app1#session1”. We will describe this in more details in the PMI path section.

The responseTime statistics is designed to calculate some statistics facts, for example, average response time, minimum response time, maximum response time, and sum of squares of the response times. Users can access these data by querying the PMI data using wpfadmin command, wsadmin command or MBean.

5.3.5 PartitionHandlerLocal

The PartitionHandlerLocal interface defines callback methods that enable your application to receive specific partition lifecycle events from the WPF runtime. There are 4 methods on this interface.

5.3.5.1 PartitionHandlerLocal#getPartitions

The getPartitions method is called when the application is started in a cluster member. It's important to realize that this method is called once on EVERY cluster member EVERY time the application starts. This method signature is:

```
PartitionDefinition[] getPartitions()
```

The method should return an array of PartitionDefinition objects. There should be exactly one PartitionDefinition for every partition the application can host in this cluster member. A cluster member can only be a candidate for a partition when that partition has been returned from getPartitions.

NOTE: It's recommended that the application always return the same set of partitions on each cluster member.

The HA Manager policies can be used to limit the partition to activation on certain cluster members or even 'pin' a partition to a particular cluster member.
Here is an example getPartitions method:

```
public PartitionDefinition[] getPartitions()
{
  PartitionDefinition[] rc = new PartitionDefinition[10];
  for(int i = 0; i < rc.length; ++i)
  {
    rc[i] = ivRuntime.createPartitionDefinition("" + i);
  }
  return rc;
}
```

This returns 10 partitions to the runtime. The partitions are named {0,1,2,3,4,5,6,7,8,9}. They are all created using the default classification.

5.3.5.2 PartitionHandlerLocal#partitionLoadEvent

The HA Manager calls the partitionLoadEvent when a partition is activated on a cluster member. This gives the application the opportunity to perform any required initialization prior to receiving IIOP requests for the specified partition (The partition name is provided as a string parameter). The partionLoadEvent method signature is:

```
boolean partitionLoadEvent(String partitionName)
```

The method should return true if the cluster member is ready to accept work for the specified partition. Once the method returns, the IIOP routing tables are updated and incoming IIOP requests will be delivered to this cluster member for the specified partition.

If the method returns false, then the HA Manager disables the cluster member for this partition and will try to activate the partition on a different cluster member based upon the current policy. This disabling only applies to the partition passed to the method, other partitions may still be activated on this cluster member. Applications that return false should ensure that adequate trace is enabled to allow the administrator to diagnose the problem. If the problem is determined to be transient in nature, then the administrator can use JMX to enable the cluster member again for that specific partition.

If an application is using messaging to accept incoming requests for a partition (as opposed to direct IIOP routing) then the application should subscribe at this time to the topics/queues on which the requests can arrive.

5.3.5.3 PartitionHandlerLocal#partitionUnloadEvent

The HA Manager calls this method when a partition is deactivated. The event tells the application to stop processing requests for the specified partition. The WPF runtime updates the IIOP routing table for this cluster member prior to invoking this callback. The method has the following signature:

```
void partitionUnloadEvent(String partitionName)
```

If an application is using messaging to accept incoming requests (as opposed to direct IIOP routing) then it should unsubscribe to the topics/queues on which those requests arrive. The HA Manager will not activate a replacement cluster member until this method returns.

5.3.5.4 PartitionHandlerLocal#isPartitionAlive

The isPartitionAlive method is only called when the policy controlling the partition has the isAlive attribute set to true in the CoreGroup Policy in effect for this partition set (see wpfadmin policy overview and policy command examples).

The default WPF Partition policies disable this call back, as for many partitions, the overall cluster performance can be affected. For very tightly managed environments, they may wish to manage this attribute in an active manner.

When the HA Manager policy isAlive attribute is true, this method is called every X seconds (also set in the policy) while a partition is activating/active or deactivating. The interval is in seconds and is specified using a different attribute on the controlling policy. The isPartitionAlive method is never called when the isAlive policy attribute is set to false. The method signature is:

```
boolean isPartitionAlive(String partitionName)
```

If the method returns false then the JVM is 'panicked' (the JVM is halted) and another cluster member is chosen to host the partition. The application can use this method to verify that the partition is operating correctly, perhaps by asking a peer cluster member to invoke a partition method to perform a ping type operation or sanity check from a remote JVM. If the result of this operation is not successful then the JVM knows that something is wrong and then

5.3.6 Threading issues for the PSSB callback methods

The HA Manager always calls the partitionLoadEvent before the partitionUnloadEvent, and the invocations occur serially for the same partition (they are never called concurrently for the same partition). The HA Manager maintains a dedicated thread pool for invoking these methods. When N partitions are activated, the load events are delivered to the application using this thread pool; therefore, if the thread pool has 10 threads, then at most 10 partitions are activated/deactivated at once. If there are 100 partitions to activate, then the 100 activation method calls are queued to the thread pool and the thread pool delivers the events to the application 10 at a time (approximately, dependent upon CPU count, operating system, etc.). See **Error! Reference source not found.** for more information.

5.3.7 Writing an application client

There is no difference between WPF-enabled client and normal client.

5.3.8 Modifying the EJB stubs (required after deploying ear)

The normal EJB deploy process produces cluster enabled stubs. These stubs need to be modified with a second pass in order to be partition routable. The tool provided takes the EAR and produces a new EAR with the newly modified stubs. The stubs are modified for all the partition routable EJBs. A partition routable EJB is a stateless session bean with a remote interface and an associated XXX_PartitionKey router Class (POJO).

The wpfStubUtil is a tool located in the <WAS_ND_HOME>/bin directory that should be run on partitioned ear files after ejbdeploy has been run. Here's the usage for wpfStubUtil (the order of the options is important, and tool may fail if order is not preserved):

wpfStubUtil -ear <earname> -jar <jarname> -class <classname> -temp <temp working directory> [optional flags: -stubUpdateClasspath <classpath> -verbose -stubDebug -keep -rmicextclasspath <classpath> -extdirs <javac extdirs>]

The -ear, -jar, -class, and -temp are all required. The -stubUpdateClasspath command is optional, but if it is used it must be the first argument after the temp working directory. An explanation of each option:

-ear <ear name>: The ear containing the partitioned application.
-jar <jar name>: The ejb jar within the ear that contains the partitioned EJB.
-class <class name>: The remote interface class of the bean.
-temp <temp working directory>: The directory where all the work takes place
-stubUpdateClasspath <classpath>: (optional) The tool will append its classpath with what the user passes.
-verbose: (optional) Verbose option for the tool output.
-stubDebug: (optional) Puts extra debug output in the stub. Should not be used in production environment.
-keep: (optional) Does not delete the contents of the temp working directory after tool is finished.
-rmicextclasspath <classpath>: (optional) Appends what the user passes in to the rmic classpath.
-extdirs <javac extdirs>: (optional) Appends what the user passes in to the javac extdirs.

The wpfStubUtil can update any session EJBs with XXX_PartitionKey class defined. These Session EJBs can be either PRSB or PSSB.

An example of running wpfStubUtil for an ear containing a facade PRSB bean:

```
C:\stub>\ws\Application server\bin\wpfStubUtil.cmd -ear D_WPFFacadePartitionSample.ear -jar
WPFKeyBasedPartitionEJB.jar -class com/ibm/websphere/wpf/ejb/WPFKeyBasedPartition.class -temp \working
A subdirectory or file \working already exists.
WPFC0069I: Unpacking ear file D_WPFFacadePartitionSample.ear
WPFC0070I: Unpacking jar file WPFKeyBasedPartitionEJB.jar
WPFC0071I: Running rmic to generate stub source
[rmic output - removed]
 [done in 2334 ms]
WPFC0072I: Updating stub source
WPFC0073I: Compiling modified stub source
WPFC0074I: Rejaring jar file WPFKeyBasedPartitionEJB.jar
WPFC0075I: Rejaring ear file D_WPFFacadePartitionSample.ear
Cleaning up
```

An example of running wpfStubUtil for two ears containing partitioned routable session beans. Notice the D_StockAccount.ear requires an extra jar file (StockEJB.jar) to be appended to the rmic classpath (-rmicextclasspath), stubUpdateClasspath (-stubUpdateClasspath), and javac ext dirs (-extdirs):

```
C:\stub>\ws\Application server\bin\wpfStubUtil.cmd -ear D_Stock.ear -jar StockEJB.jar -class
wpf/test/stock/ejb/ProcessStock.class -temp \working
A subdirectory or file \working already exists.
WPFC0069I: Unpacking ear file D_Stock.ear
WPFC0070I: Unpacking jar file StockEJB.jar
WPFC0071I: Running rmic to generate stub source
[rmic output - removed]
 [done in 2324 ms]
WPFC0072I: Updating stub source
WPFC0073I: Compiling modified stub source
WPFC0074I: Rejaring jar file StockEJB.jar
WPFC0075I: Rejaring ear file D_Stock.ear
Cleaning up

C:\stub>\ws\Application server\bin\wpfStubUtil.cmd -ear D_StockAccount.ear -jar StockAccountEJB.jar -class
wpf/test/stockaccount/ejb/Process.class -temp \working -stubUpdateClasspath \stub\StockEJB.jar -rmiextclasspath
\stub\ StockEJB.jar -extdirs \stub\StockEJB.jar
A subdirectory or file \working already exists.
WPFC0069I: Unpacking ear file D_StockAccount.ear
WPFC0070I: Unpacking jar file StockAccountEJB.jar
WPFC0071I: Running rmic to generate stub source
[rmic output - removed]
[done in 2333 ms]
WPFC0072I: Updating stub source
WPFC0073I: Compiling modified stub source
```



```
WPFC0074I: Rejaring jar file StockAccountEJB.jar
WPFC0075I: Rejaring ear file D_StockAccount.ear
Cleaning up
```

An example of running wpfStubUtil for an ear containing only a partitioned stateless session bean:

```
C:\stub>\ws\Application server\bin\wpfStubUtil.cmd -ear D_WPFKeyBasedPartitionSample.ear -jar
WPFKeyBasedPartitionEJB.jar -class com/ibm/websphere/wpf/ejb/WPFKeyBasedPartition.class -temp \working
A subdirectory or file \working already exists.
WPFC0069I: Unpacking ear file D_WPFKeyBasedPartitionSample.ear
WPFC0070I: Unpacking jar file WPFKeyBasedPartitionEJB.jar
WPFC0071I: Running rmic to generate stub source
[rmic output - removed]
[done in 2373 ms]
WPFC0072I: Updating stub source
WPFC0073I: Compiling modified stub source
WPFC0074I: Rejaring jar file WPFKeyBasedPartitionEJB.jar
WPFC0075I: Rejaring ear file D_WPFKeyBasedPartitionSample.ear
Cleaning up
```

There are some instances where the wpfStubUtil cannot find a class even though it is located in a jar that is specified in the classpath. In these cases, unjar all of the classes from the jar containing the class it cannot find in the -temp <temp directory>.

5.4 Data Partitioning Patterns

This pattern maps a subset of the data to a single server only. This server can then aggressively cache that data because it knows all requests for that subset are being routed to a single server. The data can be partitioned a number of ways. We can use either a variable set of partitions or a fixed set of partitions.

When using a one of N policy for the partitions, the HA Manager will activate each partition on exactly one cluster member. This is very useful if an application has some long-lived tasks that would benefit from running only on a single cluster member at a time.

5.4.1 Variable Partition Set

The use of an object key as the partition name is perhaps the best example of the variable partition set pattern. The routing POJO extracts the data key from each request, and the request is then routed to the cluster member that the partition was assigned to by the HA Manager.

One advantage of this approach is that if a specific key becomes 'hot', from a load perspective, then the key can be moved independently of other partitions to another server. One disadvantage of this pattern is that there are likely to be many partitions, and therefore a higher system overhead associated with their administration/management. WPF can scale to a very large number (~20k) partitions, but applications should always strive to minimize the number of partitions for each application.

5.4.2 Fixed Partition Set.

We can use any hashing scheme that maps the keys to a fixed set of partitions, for example, you could use a normal hashing algorithm that takes the key and converts it to an integer. This type of hash function typically returns a 32 bit or 64 bit integer, which in general produces too large a space of partitions. Instead, apply a modulo to the generated hash to limit the total number of partitions. For example, we could hash the key and then use the 128 as the modulo (number of partitions). The application should create 128 partitions, the first one named 0 and the last one named 127, so (128 in total). This is a good compromise as this pattern can potentially scale to 128 cluster members before we have more servers than partitions.

Another approach is to use ranges. This is still a form of hashing. The partitions could be named "A-F", "G-N", "O-R", "S-Z". The routing POJO would then examine the key from the request and return a string from one of these ranges.

5.4.3 Singleton Pattern

This pattern can be used for a clustered application requiring a singleton to perform some task within the cluster. The PSSB would create a PartitionDefinition whose name is based on the singleton, for example "RECORD_PURGER". The singleton's task is to purge removed records from the database in batch mode.

Upon startup, the HA Manager would assign the singleton partition to exactly one cluster member. In addition, the HA Manager policy could be configured to specify a preferred server for the singleton, or even 'pin' the singleton to a specific server.

5.4.4 Hash based partitioning.

Here, we hash the data keys and then use the resulting hash code modulo some maximum number as the partition names. The application returns a set of partitions numbered from 0 till 511. This allows the application to hash the request keys and then use the hash code modulo 512 as the partition name.

5.4.5 Slave Multiple Reader/Master Single Writer Pattern

The optimal application for this pattern requires a single writer for an application partition at a time for high write performance. This writer uses a write through cache for database access. So long as there are no external applications modifying the database then the writer can cache very aggressively and virtually eliminate queries against the database. The read load on a given partition however would be very high. The application also has a requirement to not allow heavy read activity to slow down the writer. It is also permissible for the reads to be slightly stale. We also want to be able to distribute the readers for a partition across multiple cluster members to spread the system load.

5.4.6 Partition Specific CMP data

This is data specific for an active partition in a server. We want to cache this data on the cluster member that has the active partition and then use a write through cache to write it to the database when it changes. We don't have to worry about invalidations by other cluster members because WPF guarantees the partition is on exactly one cluster member. This allows the application to work as single server speed without loss of data integrity in a cluster. Clearly, the database can be modified by other applications then this cannot be used. If this is the case then option C cached with optimistic update is probably the best answer but it depends greatly on the read/write ratio of the application.

5.4.6.1 Option A CMPs

This is an important approach for partitions with specific data access patterns. If we used the approach in **Error! Reference source not found.** then we can potentially run in to a lot of problems. The main problem with Option C CMPs with lifetime in cache is that WebSphere provides no API for removing such a CMP instance from the cache. WebSphere does provide a JMS mechanism that can remove an object but this isn't efficient for this scenario. So, objects cached using life time in cache cannot be removed programmatically from the cache by the application. This makes option C with lifetime cache unsuitable for partition data. The only option that will work is option A. Option A CMPs can be thought of as cached CMP although it's not technically a cache. The J2EE specification says that if an option A CMP method throws an unexpected exception then the bean instance must be removed from the container and the transaction currently active is rolled back. We will take advantage of this to implement the partition data cache.

5.4.6.2 Efficiently removing an option A CMP instance

We will now describe the most efficient way to remove such an instance from the option A pool. First, add a method to your CMP bean.

```
public void invalidateFromCache()
{
}
```

Add this method to the CMP local interface. Modify the descriptor for the method so that it uses TX_REQUIRED

You can do this using the EJB descriptor editor by clicking on the assembly tab and then click the "Add…" button on the container transactions panel. Select the CMP and then click on the invalidateFromCache method. Select "Required" from the combo box and then click OK. You should then see something like **Error! Reference source not found.**. Now, we'll add a home method like the one below:

```
public void ejbHomeInvalidateCompleteOrders(Collection/*<OrderLocal>*/ orders)
{
  Iterator iter = orders.iterator();
  while(iter.hasNext())
  {
    OrderLocal order = (OrderLocal)iter.next();
      // this just adds the option A bean to this transaction so that when we rollback
below
      // it gets discarded, i.e. removed from the cache.
    order.invalidateFromCache();
  }
  myEntityCtx.setRollbackOnly();
}
```

Add this method to the beans local home by right clicking on the method, select Enterprise Beans and then promote to local home. Set this method transaction attribute to RequiresNew also using the same approach as before. This method takes a List of OrderLocal (our CMP is called Order) and removes them from the option A 'cache'. Note, that we could have taken a List of keys but then we'd need to execute a findByPrimaryKey and then call the method. This is more expensive than the approach taken here. The method basically iterates over the list and calls the invalidate method on each order in the list. The invalidate method associates the option A bean with the current transaction. After each bean is associated with the transaction, we mark it as roll back only. When the method returns, the transaction is rolled back and all the associated option A beans are discarded from the cache. Here is an example. We have some business logic that accepts a new order. As part of accepting a new order, it's possible that several other orders may be completed. The acceptOrder method returns a List<OrderLocal> to the caller. The caller must then pass this list to the home method above to remove them from the cache. They are not deleted from the database, just the cache. Here is an example:


```
BookLocal book = ivBookHome.findByPrimaryKey(ivOrder.getSymbol());
Collection/*<OrderKey>*/ completedOrders =
                        book.acceptOrder(session, pub, ivOrder, cache);
// invalidate all completed orders from the CMP option A cache.
try
{
  if(!completedOrders.isEmpty())
        ivOrderHome.invalidateCompleteOrders(completedOrders);
}
catch(Exception e)
{
  // ignore expected exception
}
```

This shows us looking up a Book CMP, calling its accept order method. This method returns the orders that need to be removed from the cache as a result of the call. The acceptOrder method uses a RequiresNew transaction so its transaction is committed automatically when this method returns. We then just use the home method on Order to remove the returned CMPs from the cache. The acceptOrder method just collects all the Order instances that need to be removed like this:

```
public Collection/*<OrderLocal>*/ acceptOrder(…)
       Collection completeOrders = new LinkedList();
              OrderLocal buyer = ...;
              OrderLocal seller = ...;
       ... some business logic....
              if(buyer.getIsComplete())
                     completeOrders.add(buyer);
              if(seller.getIsComplete())
                     completeOrders.add(seller);
       …
       return completedOrders;
}
```

This code demonstrates the application removing a cached object that it has finished processing. If the application didn't do this then the cache has both useful and useless instances in it. The useful ones are records that it still needs or are pending or not completed. There is no point in caching completed records so an application can use this approach to remove 'completed' objects.

5.5 Proxy DataSource Development

5.5.1 CMP Datasource Overview

Applications that use CMP beans normally specify a single database to use with one CMP EJB. That means, all the CMP EJB instances of one type will read and write to one database node, which will become the performance bottleneck. . You could deploy the CMP beans N times, once for each database but this is not very flexible for the following reasons:

- Requires N copies of the code with N JNDI names etc
- Requires a deploy step when a database is added.
- It is just not that easy to manage.

WebSphere Extended Deployment has a feature that allows the application to tell WebSphere which datasource to use before the transaction starts. This means that when a cluster member receives a request for a particular application partition then it can tell the CMP runtime to use a specific DataSource for the duration of the current transaction. This allows the directed transaction pattern to be used with the application. This allows such applications to increase their availability and allows the database tier to scale horizontally on blade type environments. The applications can also take advantage of the MAPPER table pattern to very flexibly manage data and move partitions around to better manage the operational aspects of an application such as how to move a very busy partition to a lightly loaded database node for performance reasons.

Due to testing efforts, in WAS Extended Deployment 5.1, only support DB2 UDB and Oracle are supported with the proxy datasource. Here are the supported JDBC drivers:

- DB2 v8.1 FP7 (v8.2) legacy CLI-based JDBC drivers.
- DB2 v8.1 FP7 (v8.2) Universal type 2 JDBC drivers.
- DB2 v8.1 FP7 (v8.2) Universal type 4 JDBC drivers.
- Oracle 9i thin client JDBC drivers

If you need support for other databases or JDBC drivers, please contact IBM support for more details.

5.5.2 Proxy DataSource programming model

In the current WebSphere Application Server programming model, you can only specify one CMP connection factory, which corresponds to one datasource, for one CMP EJB. Under the new proxy datasource support, we still specify only one CMP connection factory for one CMP EJB, however, during the run time, this proxy datasource will route the connection requests to different underlying datasources.

Which underlying datasource the proxy datasource will route to is decided by the application. At the beginning of every transaction, users can use WebSphere Extended Deployment -specific API to specify which underlying datasource this current transaction will use.

Here is the general programming model for the proxy datasource support:

- Application developers use a Session EJB as the session façade. The Session EJB is required to use CMP EJB local interfaces to interact with CMP EJBs.

- Application developers or assemblers define all the resource references for CMP EJB persistence in the session EJB. For example, if you expect your CMP EJBs be persisted in three different database servers, you will need to create three different resource (datasource) reference names.

- Create the datasources. Creating the datasources consists of two steps: creating all underlying datasources and creating the proxy datasource. The application deployer creates datasources for all of the database servers used for CMP EJB persistence in the WebSphere administration space. For example, the deployer creates four datasources for three different database servers with JNDI name “jdbc/WestAccountDS”, “jdbc/EastAccountDS”, “jdbc/NorthAccountDS”, and “jdbc/SouthAccountDS”. The application deployer then creates a proxy datasource using the proxy DataSource JDBC provider or proxy DataSource JDBC provider (XA) provided only in WebSphere Extended Deployment. In the customer property “jndiNames” of

the proxy datasource, the deployer add the JNDI names of all the datasources just defined in the format of "dsJNDIName_1;dsJNDIName_2; dsJNDIName_3… dsJNDIName_N". In this example, the jndiNames will be set to "jdbc/WestAccount;jdbc/EastAccount;jdbc/NorthAccount;jdbc/SouthAccount".

- In the EJB applications, application developers will provide a mapping for CMP EJB attributes (one or more attributes) to the resource references defined in the session EJBs. This mapping can be stored in an XML, database, and property file or even programmatically, whichever the way they want. Based on this mapping, the developer knows which datasource a transaction should use. Here is an example of creating mapping programmatically.

```
if (accountId.startsWith("w")) {
    return resrefs[0];
}
else if ((accountId.startsWith("e")) {
    return resrefs [1];
}
else if ((accountId.startsWith("n")) {
    return resrefs [2];
}
else if ((accountId.startsWith("s")) {
    return resrefs [3];
}
```

In this example, account IDs starting with "w" (west) are mapped to the first JNDI name; account IDs starting with "e" (east) are mapped to the second JNDI name; etc.

- Session EJBs can lookup WSDataSourceHelper from the JNDI namespace (one lookup in one EJB lifecycle) in the setSessionContext method, and then call `WSDataSourceHelper.resolveDataSourceReference(String)` to get the global JNDI name for every resource reference name.

- During the beginning of the transaction, the session bean calls `WSDataSourceHelper.setCurrentDataSource(dsJndiName)` with the global JNDI name to specify which datasource should be used. Session EJBs can then do normal CMP operations in that transaction.

5.5.3 API

The main API for the proxy datasource support is WSDataSourceHelper. Here is the WSDataSourceHelper interface.

```
/**
 * <p>WSDataSourceHelper interface is an interface used for CMP multiple
 * datasource support (also called proxy datasource). Users can lookup an
 * instance of WSDataSourceHelper from the JNDI name space using JNDI name
 * WSDataSourceHelper.JNDI_NAME.</p>
 *
 * <p>There are two helper methods in this interface.</p>
 * <ul>
 * <li><b>resolveDataSourceReference(String)</b>:
 * This helper method is used to resolve the global JNDI name of the
 * datasource associated with a resource reference. For example, if a
 * resource reference "jdbc/myDS1" is mapped to a datasource with global
 * JNDI name "jdbc/Bank1", method call resolveDataSourceReference("jdbc/myDS1")
 * will return "jdbc/Bank1".</li>
 *
 * <li><b>setCurrentDataSourceJndiName(String)</b>:
 * This method is used to set the JNDI name (not the resource reference name) of
 * the datasource that the current transaction will access. Currently,
 * one transaction can only access one datasource.</li>
 * </ul>
 *
 * @ibm-api
 */
public interface WSDataSourceHelper {

    /** The JNDI name for user to look up an instance of WSDataSourceHelper */
    String JNDI_NAME = "java:comp/env/com/ibm/websphere/proxyds/WSDataSourceHelper";

    /**
     * Resolve the datasource reference to the global JNDI name. For example,
     * if a resource reference "jdbc/myDS1" is mapped to a datasource with
     * global JNDI name "jdbc/Bank1", resolveDataSourceReference("jdbc/myDS1")
     * will return "jdbc/Bank1".
     *
     * @param dsResRefName resource reference name
     * @return the resolved datasource global JNDI name for this resource reference.
     * @exception ResRefNotFoundException indicates the resource reference name
     * cannot be found.
     */
    String resolveDataSourceReference(String dsResRefName) throws
ResRefNotFoundException;

    /**
     * <p>Set the JNDI name (not the resource reference name) of the datasource
     * that the current transaction will access. Currently, one transaction can
     * only access one datasource.</p>
     *
     * <p>During the development time, the developers cannot know the global JNDI
     * name of the datasource that a resource reference will be mapped to. The
     * only known fact is the resource reference name. The recommneded practice is
     * to call the resolveDataSourceRefrence(String) method to get the JNDI name of
     * the mapped datasource, and then call setCurrentDataSourceJndiName with the
     * global JNDI name.</p>
     *
     * @param dsJndiName the current datasource JNDI name
     */
    void setCurrentDataSourceJndiName(String dsJndiName);

}
```

5.5.4 Developing application using proxy datasource support in WSAD.

ProxyDSAccountSample application contains one Application Client module and one EJB module. The EJB module, ProxyDSAccountSampleEJB, has four EJBs:

- Account: CMP EJB
- AccountOwner: CMP EJB. One Account CMP EJB has one or more AccountOwner EJBs.

- AccountTransaction: This is a façade session bean to access CMP EJBs. It is also a Partitioned Routable Session Bean (PRSB).
- AccountPartitionBean: This is a Partitioned Stateless Session Bean (PSSB).

Accounts and account owners are stored in two databases, one located in the west coast, and the other located in the east coast. The database in the west coast has database name as “westtest”, and the database in the east coast has database name as “easttest”. All the accounts stored in the westtest have a prefix “W”, and all the accounts stored in the easttest have a prefix “E”.

When the Façade AccountTransaction gets a request from the client to do account operations, it examines the accountId. If the accountId starts with “W”, the database westtest will be used. If the accounted starts with “E”, the database easttest will be used.

Here we will develop a J2EE application that uses proxy datasoure features in WebSphere Studio Application Developer. Our goal is to develop the ProxyDSAccountSample shipped with WebSphere Extended Deployment. The ProxyDSAccountSample is shipped in WAS_HOME/installableApps/ProxyDSAccountSample.ear file. Before we start the project, please extract all the source codes from this ear file. We will need to copy the source codes to complete the exercise.

5.5.4.1 Create ProxyDSAccountSample project

a. Create a ProxyDSAccountSample J2EE Project by select “File/New/Enterprise Application Project”, and click “next”.
b. Select “create J2EE 1.3 Enterprise Application Project”, and then click “Next”.
c. Set the “Project name” to “ProxyDSAccountSample”, and then click “Next”.
d. Click “New Module…”. Leave the “Application Client Project” and “EJB Project” checked, and uncheck “Web Project” and “Connector Project”, and you will see the panel as shown in Figure 4.4.1.

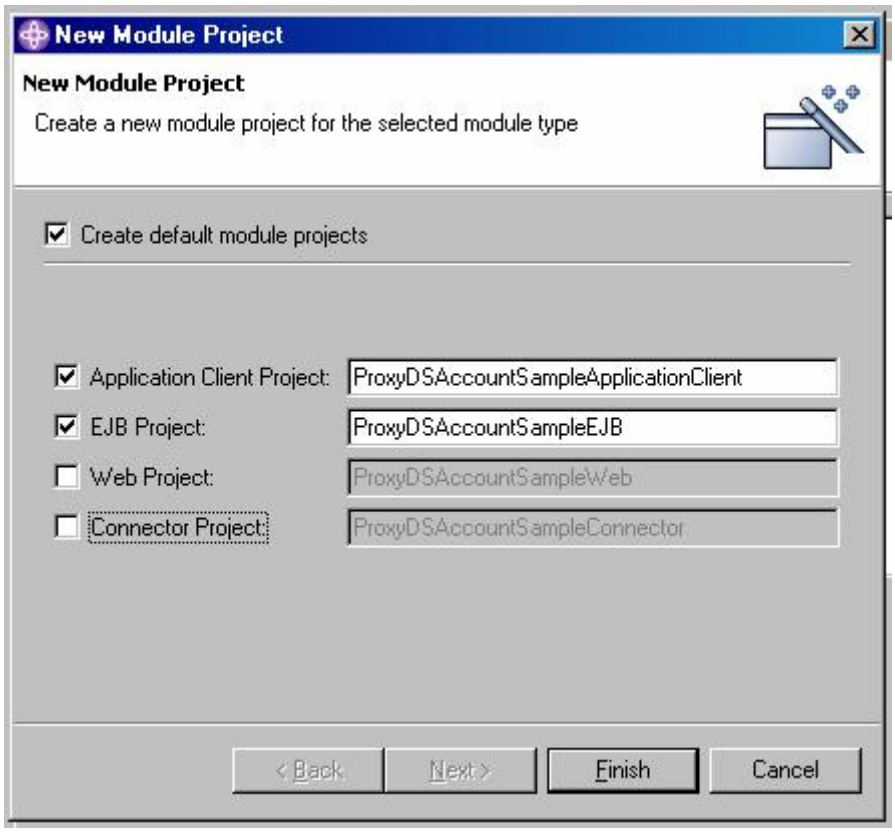


Figure 4.4.1 New Module Project.

e. Click “Finish” to finish creating the modules, and then click “Finish” to finish creating the application project. Click “yes” if there is a window to ask you switch to J2EE perspective. You will see the enterprise application project ProxyDSAccountSample, the application client project ProxyDSAccountSampleApplicationClient, and the EJB project ProxyDSAccountSampleEJB are created.

5.5.4.2 Create entity EJBs Account and AccountOwner

In order to simplify the sample, we will use only two entity beans: Account and AccountOwner. Again, for the reason of simplification, there is a one to many relationship between Account and AccountOwner.

f. Click “File/New/Enterprise Bean” to create a CMP EJB. Select “ProxyDSAccountSampleEJB” as the project name, and then click “Next”

g. You will see the following picture 4.4.2. Check “Entity bean with container-managed persistence (CMP)” field. Check “CMP 2.0 Bean”. Type “Account” as the “Bean name”, and “com.ibm.websphere.proxyds” as the “Default package”.

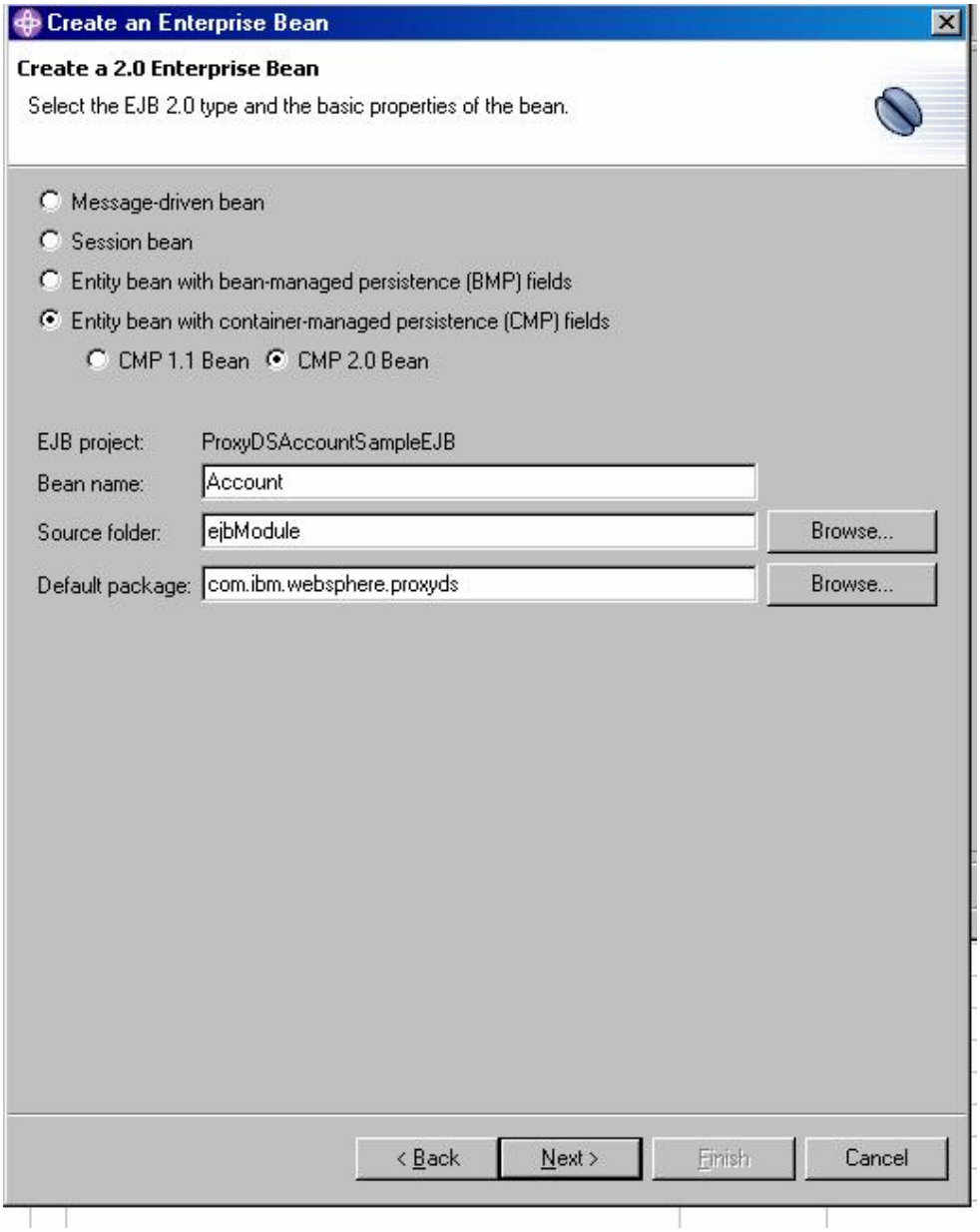


Figure 4.4.2 Create Account CMP EJB

h. Click “Next”, and you will see the panel as shown in Figure 4.4.3. Leave the “local client view” checked and the “remote client view” unchecked.
i. Create the following four CMP fields
accountId. Type String. This is also the key field.
balance. Type double
creationData. Type java.sql.Timestamp
openBalance. Type double.

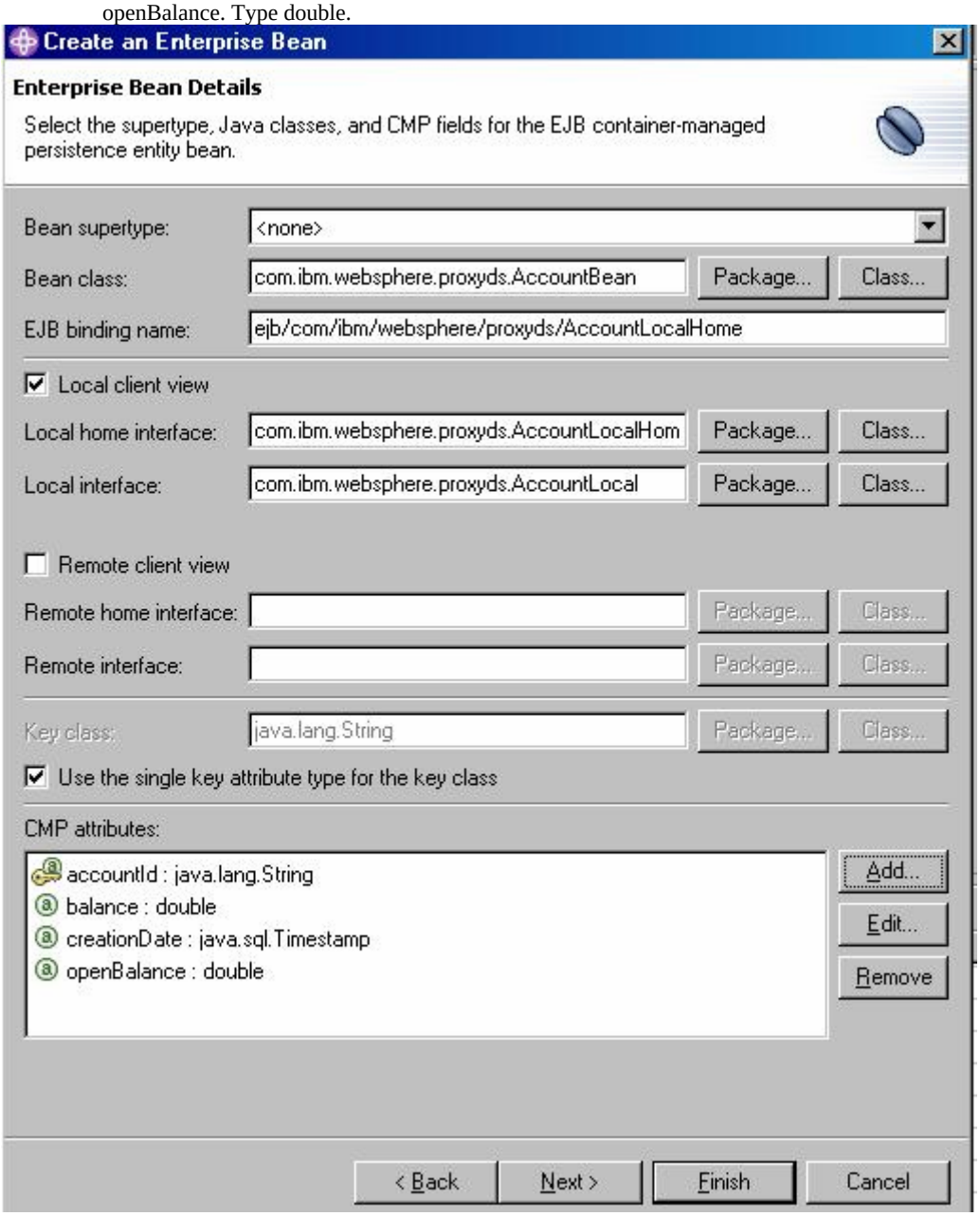


Figure 4.4.3 create the Account CMP EJB.

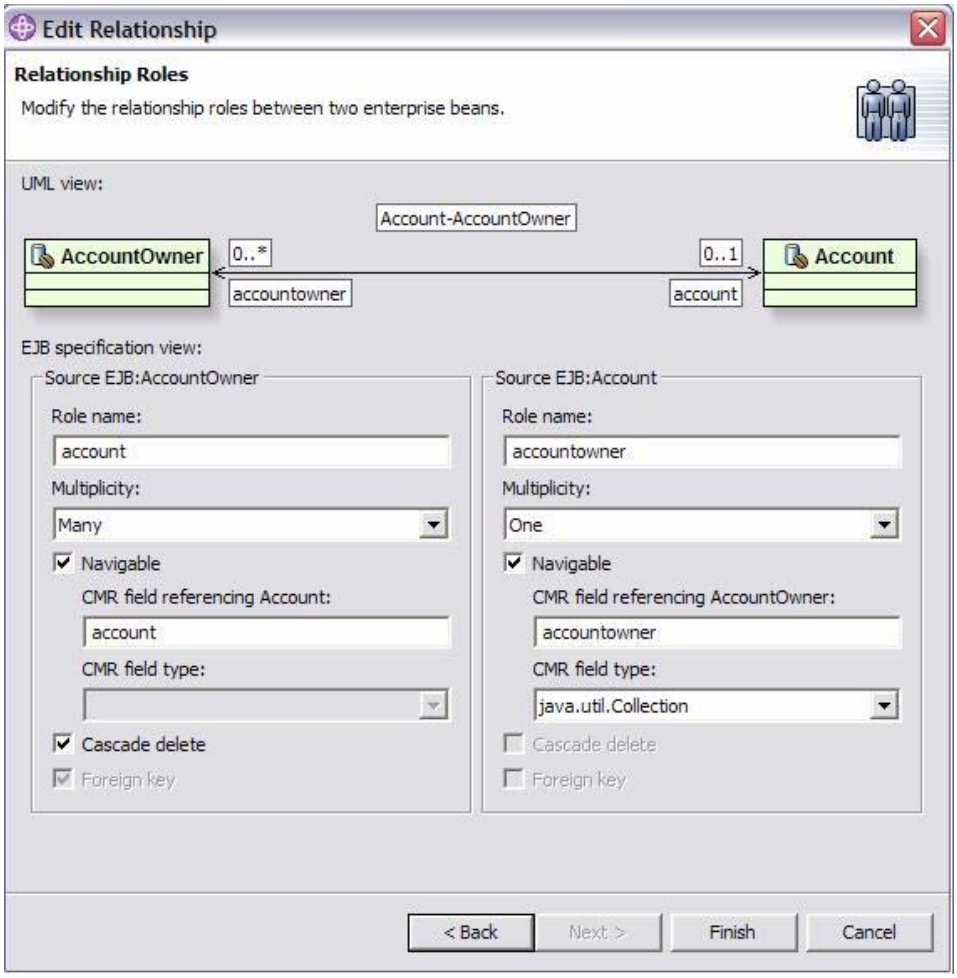
j. Click “Finish”.

Now you’ve created the CMP EJB Account, follow the same steps from 6) to 10) to create CMP EJB AccountOwner with the following CMP fields:

ssn. Type String. This is also the key filed.
name. Type String
address. Type String
phone. Type String

Now, you can create a one-to-many relationship between Account and AccountOwner.

k. Click “Beans” tab on the right panel. Select “Account” from the “Beans” list.
l. Scroll down to “Relationships” list, then click “Add”.
m. Select “Account” and “AccountOwner” as the “source EJB”.
n. Click “Next”, Select “Many” for the ”Multiplicity” for role account, then check “cascade delete” for role account. Leave others as default.
o. Click “Finish”.



4.4.4 Create CMR between CMP EJB Account and AccountOwner

5.5.4.3 Create PRSB AccountTransaction.

In this step, we will create a Partition Routable Session Bean (PRSB) AccountTransaction. This session bean will serve as both routable bean and façade bean to access the CMP EJBs.

p. Click “File/New/Enterprise Bean” to create an EJB. Select “ProxyDSAccountSampleEJB” as the project name, and then click “Next”
q. Check “Session Bean” field. Input “AccountTransaction” as the “Bean name”, and “com.ibm.websphere.proxyds” as the “Default package”.
r. Click Next. Leave all fields as default,
s. Click “Finish”

t. Copy the AccountTransaction.java, AccountTransactionBean.java from the extracted ProxyDSAccountSample java code into the workspace. You will see several compilation errors, which we will resolve later.

5.5.4.4 Create PSSB AccountPartitionBean

This example will use both WPF and Proxy Datasource features. In order to use WPF feature, we need to create a Partitioned Stateless Session Bean (PSSB).

u. Click “File/New/Enterprise Bean” to create an EJB. Select “ProxyDSAccountSampleEJB” as the project name, and then click “Next”
v. Check “Session Bean” field. Type “AccountPartitionBean” as the “Bean name”, and “com.ibm.websphere.proxyds” as the “Default package”.
w. Click Next.
x. Uncheck the “Remote client view”, and check the “Local client view”.
y. Type “com.ibm.websphere.wpf.PartitionHandlerLocalHome” as the local home, and “com.ibm.websphere.wpf.PartitionHandlerLocal” as the local interface.
z. Click “Finish”
aa. Copy the AccountPartitionBean.java from the extracted ProxyDSAccountSample java code into the workspace. You will see several compilation errors, which we will resolve later.

5.5.4.5 Set the classpath

In order to develop an application with the proxy datasource support, you will need the proxyds.jar from the WAS Application Server server lib directory.

If you don’t have Extended Deployment lib directory created as a variable, follow the following steps to create a variable.

bb. Click Windows/Preferences, then expand “Java”, then click “Classpath Variables”.
cc. Click “New” to create a new variable XD_LIB, and set the path of it as shown in Figure 4.4.5.

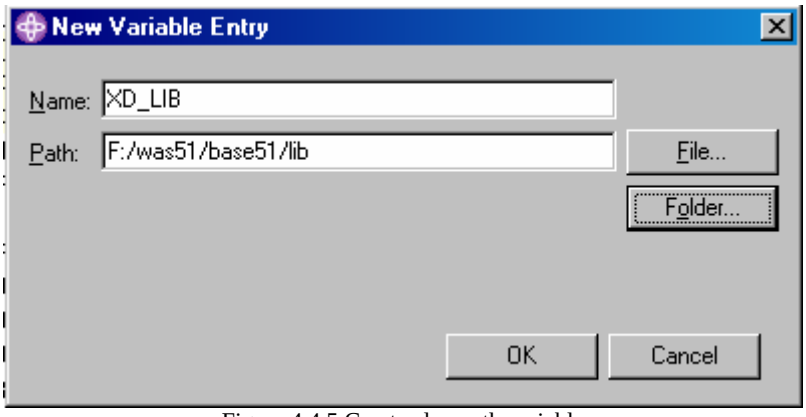


Figure 4.4.5 Create classpath variable

Now we add the proxyds.jar into the build path.

dd. In the J2EE perspective, expand “EJB modules”, right click “ProxyDSAccountSampleEJB”, then select “Java build path”.
ee. On the panel, click “Libraries” tab, and then click “Add Variable”. Select “XD_LIB” from the list, and click “Extend…”. Select proxyds.jar and click Ok as shown in Figure 4.4.6.

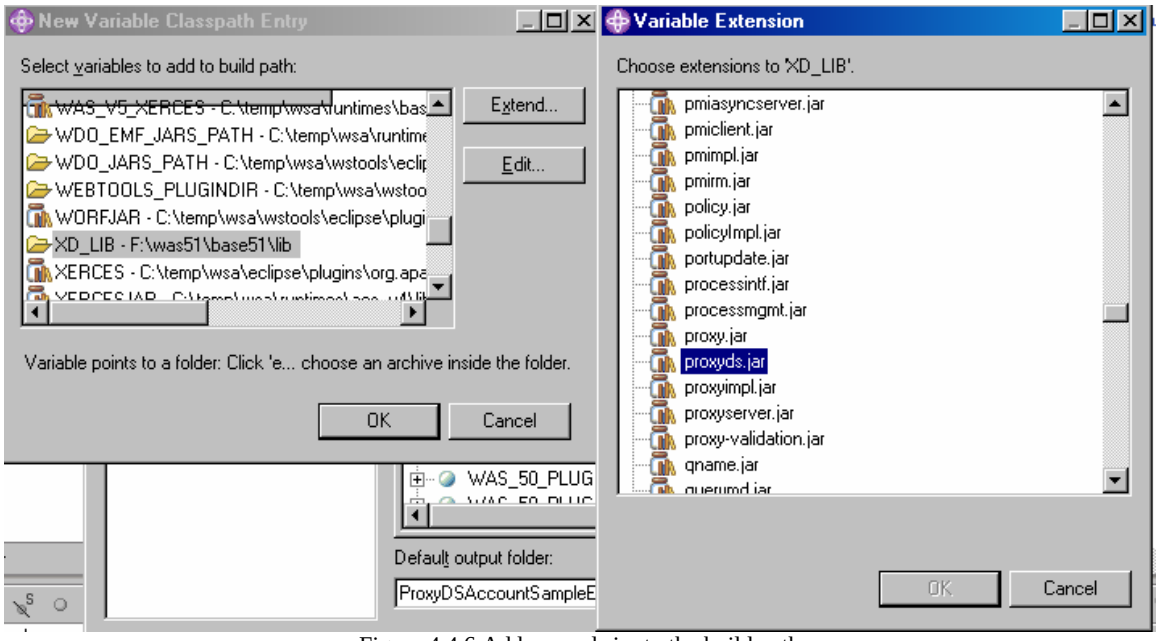


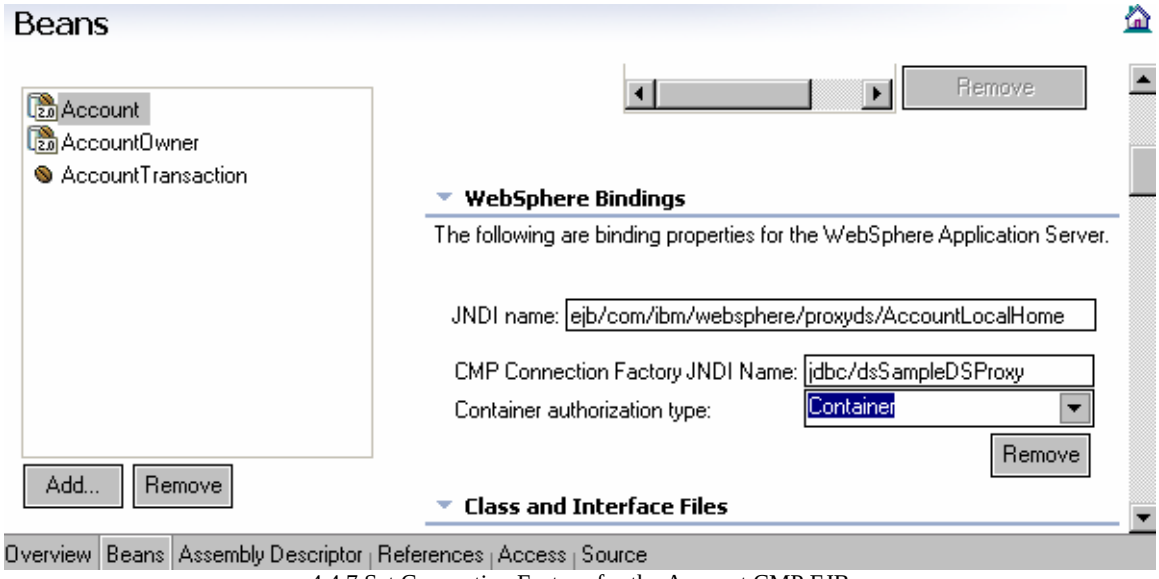
Figure 4.4.6 Add proxyds.jar to the build path

Repeat the same step to add wpf.jar to the build path, since we use WPF feature in this EJB project.

5.5.4.6 Set the connection factory for the CMP EJBs.

Now we will configure the CMP connection factory for the CMP EJBs.

ff. In the J2EE perspective, expand EJB Modules/Entity Bean, and then click “Account”.
gg. One the right panel (the EJB deployment descriptor panel”, click “Beans” tab. You will see the panel as shown in Figure 4.4.7.
hh. Set the “CMP Connection Factory JNDI Name” to “jdbc/dsSampleDSProxy”. This is the JNDI name of the proxy datasoure that will be created by the deployer. If the deployer wants to create another proxy datasource, he can modify this field using during assembly time or deployment time.
ii. Set the “Container authorization type” to “Container”.



4.4.7 Set Connection Factory for the Account CMP EJB

Repeat the same step for AccountOwner. AccountOwner uses the same CMP connection factory JNDI name "jdbc/dsSampleDSProxy"

5.5.4.7 Configure the resource references for the session bean.

As we briefly described in the programming model section, the session bean will need to create the resource references for the datasources the CMP EJB will use via the Proxy DataSource, and then specify the datasource to be used in the transactions.

As we described previously, the CMP EJB will use two DB2 datasources, one to access west coast database westtest, and the other to access east coast database easttest. So we need to create two datasource references for the session bean AccountTransaction.

jj. Select AccountTransaction from the beans list on the References tab of the EJB deployment descriptor.
kk. Click "Add….". Select "Resource reference" from the panel and then click "Next". You will see a panel as shown in Figure 4.4.7.
ll. Set the "Name" as as "jdbc/myDS1",
mm. Set the "Type" as "javax.sql.Datasource".
nn. Set the "Authentication" as "Container".
oo. Set the "Sharing scope" as "Shareable".
pp. Click "Finish".

Repeat the same step to create a resource reference "jdbc/myDS2".

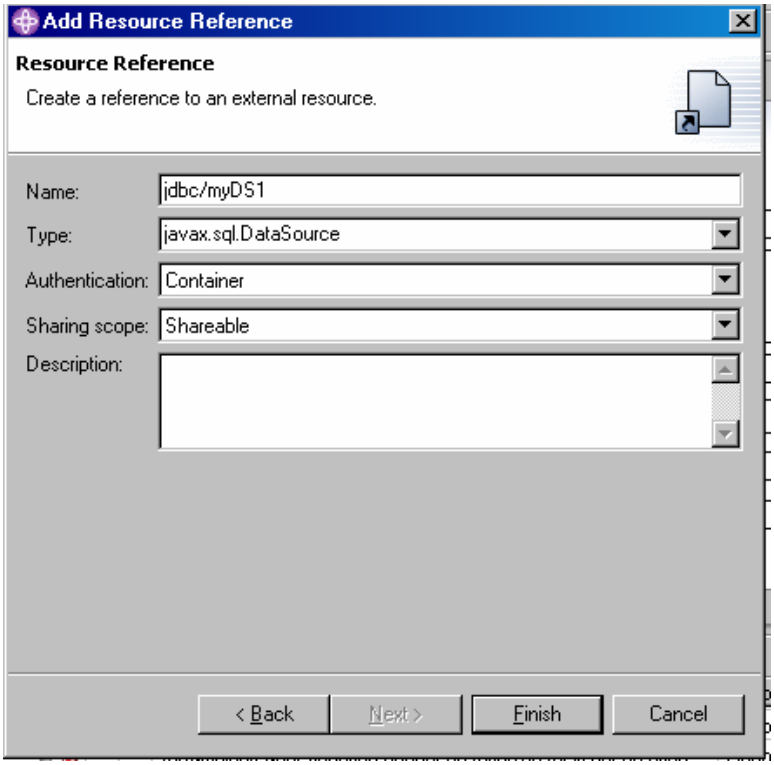


Figure 4.4.8 Add resource reference for "jdbc/myDS1".

Now, you've create two resource references for the session bean AccountTransaction. On the references panel, set the websphere binding for "jdbc/myDS1" to "jdbc/WestDS", and the websphere binding for "jdbc/myDS2" to "jdbc/EastDS".

5.5.4.8 Programming the session bean.

Open AccountTransactionBea.n.java on the WSAD, and examine the code. In the session bean, we define the resource reference name and their resolved JNDI names. The following code in this bean defines these attributes.

```
        /** DataSource resource reference 1 */
        private static String resRef1 = "jdbc/myDS1";

        /** DataSource resource reference 2 */
        private static String resRef2 = "jdbc/myDS2";

        /** Global datasource JNDI name for the DataSource resource reference 1 */
        private String globalDSJNDIName1 = null;

        /** Global datasource JNDI name for the DataSource resource reference 2 */
        private String globalDSJNDIName2 = null;
```

We also define a WSDataSourceHelper instance, so the session bean can use it to resolve the datasource JNDI name and set the datasource JNDI name for the transaction.

```
    /**
     * the WSDataSourceHelper instance used to set the datasource JNDI name
     * for the current transaction and resolve the datasource resource reference
     * to the global JNDI name
     */
    WSDataSourceHelper dsHelper = null;

  Now, lets look at the setSessionContext method

    public void setSessionContext(javax.ejb.SessionContext ctx) {
      mySessionCtx = ctx;

      // Lookup the WSDataSourceHelper, and the EJB local homes
      try {
        InitialContext ic = new InitialContext();
        dsHelper = (WSDataSourceHelper) ic.lookup(WSDataSourceHelper.JNDI_NAME);
        accountHome = (AccountLocalHome) ic.lookup(accountHomeJNDIName);
        accountOwnerHome = (AccountOwnerLocalHome) ic.lookup(accountOwnerHomeJNDIName);
      }
      catch (Exception e) {
        throw new EJBException(e);
      }

      // Get the server name
      AdminService service = AdminServiceFactory.getAdminService();
      SERVER_NAME =
        service.getCellName()
          + "/"
          + service.getNodeName()
          + "/"
          + AdminServiceFactory.getAdminService().getProcessName();

      // Resolve the resource references to the global datasource JNDI names
      try {
        globalDSJNDIName1 = dsHelper.resolveDataSourceReference(resRef1);
        globalDSJNDIName2 = dsHelper.resolveDataSourceReference(resRef2);
      }
      catch (ResRefNotFoundException rrnfe) {
        throw new EJBException(rrnfe);
      }
    }
```

We look up the WSDataSourceHelper form the JNDI namespace using the following statement and
`dsHelper = (WSDataSourceHelper) ic.lookup(WSDataSourceHelper.JNDI_NAME);`

Then we lookup the AccountLocalHome and AccountOwnerLocalHome as normal. After that, we use the AdminService API to get the server name. We use this server name to verify the partition routing is correct. Users don't have to do this in their applications. The WPF runtime will guarantee the right partition routing behavior.

Then we call the following statement to resolve the datasource references.
`globalDSJNDIName1 = dsHelper.resolveDataSourceReference(resRef1);`

During the application development time, the developer doesn't know which JNDI name this resource reference will be mapped to. Proxy Datasource component provide an API for the application to get the JNDI name of the datasource, This JNDI name datasource will be used to tell runtime which datasource should be used.

Now, lets look at the business method. Here is the code snippet for one of the business methods "withdraw" in the AccountTransactionBean.java

```
    /**
     * withdraw money from an account
     * @param accountId
     * @param amount
     * @return
     * @throws InSufficientFundException
     */
    public String withdraw(String accountId, float amount) throws
  InSufficientFundException {
      // Set the datasource this transaction will access.
      setDataSource(accountId);

      try {
        AccountLocal account = accountHome.findByPrimaryKey(accountId);
        account.withdraw(amount);
      }
      catch (ObjectNotFoundException onfe) {
        throw new EJBException(onfe);
      }
      catch (FinderException fe) {
        throw new EJBException(fe);
      }
      return SERVER_NAME;
    }
```

The difference between this method and a normal method is that we add the following statement:

```
// Set the datasource this transaction will access.
setDataSource(accountId);
```

Here is the method setDataSource:

```
    /**
     * Set the datasource the CMP is going to use for the current transaction.
     * If the accountID starts with W, datasource 1 will be used. If the accountID
     * starts with E, datasource 2 will be used.
     *
     * @param s
     */
    private void setDataSource(String s) {
      if (s.startsWith("W")) {
        dsHelper.setCurrentDataSourceJndiName(globalDSJNDIName1);
      }
      else {
        dsHelper.setCurrentDataSourceJndiName(globalDSJNDIName2);
      }
    }
```

What the method setDataSource does is to use WSDataSourceHelper to set the current datasource JNDI name for the current transaction. If the accountId starts with "W", which means the account ID should be stored in west coast database, the JNDI name globalDSJNDIName1 will be set on the thread. If the accountId starts with "E", which means the account ID should be stored in east coast database, the JNDI name globalDSJNDIName2 will be set on the thread.

Now, your application has been enabled the proxyds datasource support. In the next step, we will show how to configure the DB2 datasources and the proxy datasource in the next section.

5.5.5 Configure Proxy DataSource In WebSphere Extended Deployment

In this section, we will show how to configure Proxy DataSource in WebSphere Extended Deployment to make ProxyDSAccountSample work.

5.5.5.1 Configure your database

In order to run the sample, you need to create two DB2 databases, either in one database node or two database nodes. Here we create two DB2 databases, westtest and easttest, and connect to it using the authorized user ID and password. I will use “dbuser1” and “xxxxxx” as the user name and password for this example. Replace in the following example or scripts with your user IDs and passwords.

Find the META-INF/Table.ddl from the extracted artifacts of ProxyDSAccountSample.ear, and then execute the following commands to create tables in those two databases.
>db2 create database westtest
>db2 create database easttest
>db2 connect to westtest user dbuser1 using xxxxxx
>db2 –tvf Table.ddl
>db2 connect to easttest user dbuser1 using xxxxxx
>db2 –tvf Table.ddl

5.5.5.2 Configure the data sources.

5.5.5.2.1 Create J2C Authentication alias

1) Open the administrative console.
2) One the left panel, expand Security/JAAS Configuration, and then click "J2C Authentication Data".
3) Click New, Fill in the following to the panel:
   Alias: alias1
   User ID: dbuser1
   Password: xxxxxx
4) Click OK.

5.5.5.2.2 Create DB2 data sources.

5) Expand “Resources” on the left panel, Click “JDBC Providers”.
6) Create a cell-wide DB2 Universal JDBC provider. In order to create a cell-wide JDBC provider, you need to set the scope to cell level. Make sure you see a red arrow on the left of the cell.
7) The classpath of the JDBC provider uses DB2 JDBC driver jar files. You need to define the variables as the path where your DB2 JDBC driver jar files reside. If you have multiple nodes, and DB2 JDBC driver jar files are in different directories, you need to override the variables appropriately in the node level.
8) Click “Apply”, and then click “Data Sources”.
9) Under this JDBC provider, click “New” to create a data source with JNDI name as "jdbc/WestDS" and name as “WestDS”.
10) Click "Use this Data Source in container managed persistence (CMP)".
11) Select “<CELL>/alias1” as the Component-managed Authentication Alias and the Container-managed Authentication Alias.
12) Leave others as default. Click “Apply”.
13) Click “custom properties”. Set databaseName to “westtest”, set driverType to 4, set the serverName to your database server name, and set the PortNumber to your database server number.
14) Navigate to the JDBC provider panel.
15) Under the same DB2 Universal JDBC provider, create a datasource with JNDI name as "jdbc/EastDS" and name as “EastDS”,
16) Click "Use this Data Source in container managed persistence (CMP)".
17) Select “<CELL>/alias1” as the Container-managed Authentication Alias.
18) Leave others as default. Click “Apply”.
19) Click “custom properties”. Set databaseNane to “easttest”, set driverType to 4, set the serverName to your database server name, and set the PortNumber to your database server number.

5.5.5.2.3 Create Proxy data source

20) Expand Resources/JDBC Providers
21) Select "Proxy DataSource JDBC Provider" from the list. Click “Apply”.
22) On the Proxy DataSource JDBC Provider. Click “Apply”.
23) Click "Data Sources".

24) Click "New". You will see the panel as shown in Figure 4.4.9

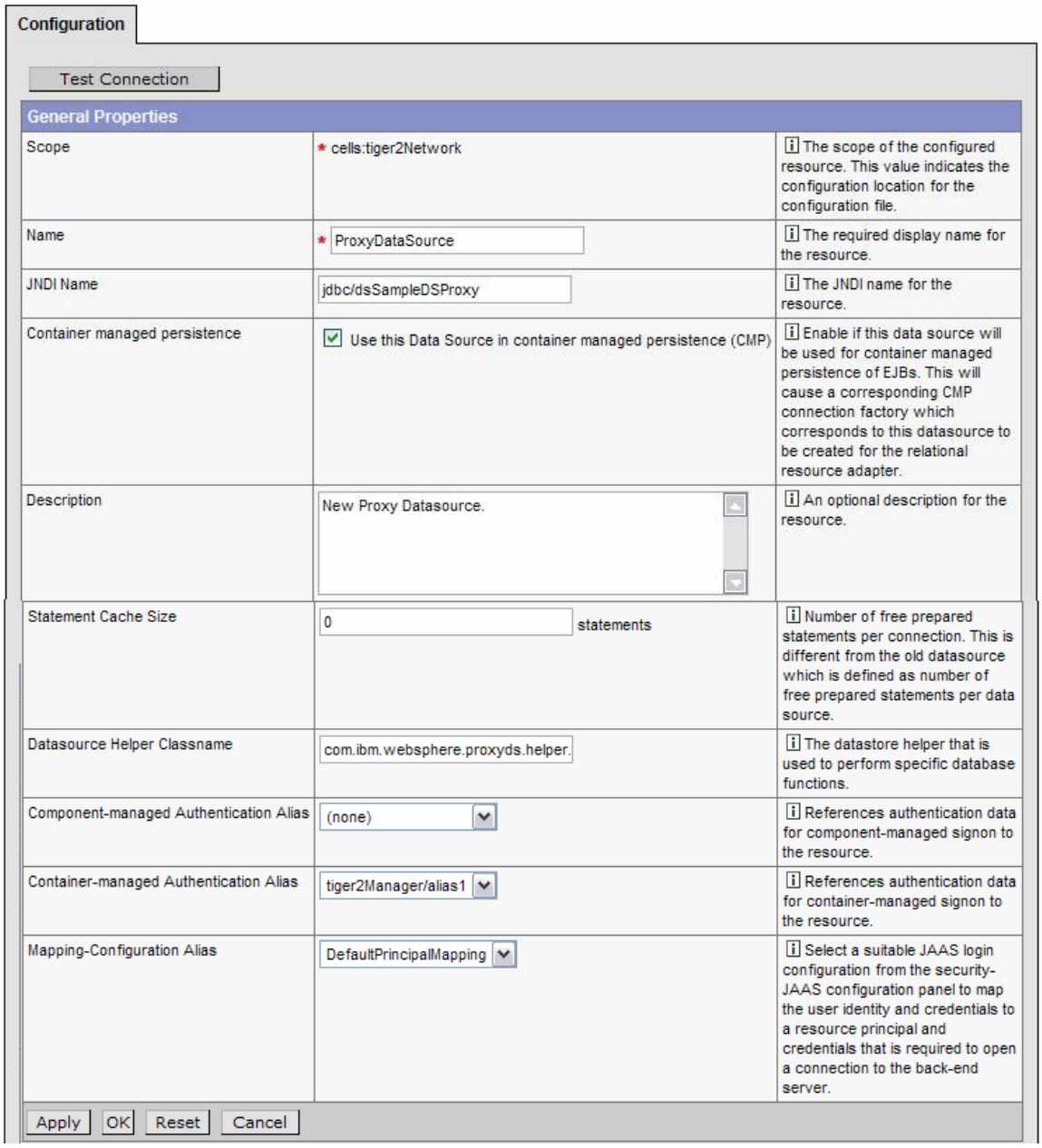


Figure 4.4.9 Create proxy datasource jdbc/dsSampleDSProxy.

25) Set the following properties.

- Set Name to "dsSampleDSProxy"
- set JNDI Name to "jdbc/dsSampleDSProxy",
- Check "Use this Data Source in container managed persistence (CMP)".
- Set the DataStore Helper to “com.ibm.websphere.proxyds.helper.DB2ProxyDSUniversalDataStoreHelper"
- Select “<CELL>/alias1” as the Container-managed Authentication Alias
- Leave others as default. Remember the statement cache size has to be 0.

26) Click custom properties. Set jndiNames to “jdbc/WestDS;jdbc/EastDS” and set statementCacheSizes to 10

Click Save to save all the configurations.

If you click “Test Connection” button for this proxy datasource, it will not work.

5.5.6 Install D_ProxyDSAccountSample.ear application.

Install the D_ProxyDSAccountSample.ear in the cluster you have created. Do not deploy the EJB when you install the app. Remember to map the application modules to the cluster instead of the default server server1.

5.5.7 Run the application client

Start the cluster.

In the <WAS_ND_HOME>/bin directory, do the following
```
wpfadmin listActive --a ProxyDSAccountSample --o 200
```

You will see 200 partitions are being activated.

In the client window, execute the following command in the <WAS_HOME>/installedApps/<cell>/ directory:
```
..\..\bin\launchClient.bat ProxyDSAccountSample.ear -CCBootstrapPort=<BOOTSTRAP_PORT> run
```

Where, the BOOTSTRAP_PORT is the bootstrap port of any server in this node, for example, 9811.

You will see the following output:[I1]
```
IBM WebSphere Application Server, Release 5.1
J2EE Application Client Tool
Copyright IBM Corp., 1997-2003
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class
com.ibm.websphere.proxyds.client.ProxyDSAccountSampleClient
----Move partition E0001 and W0001 to different servers using the following command.
    <WAS_DEPLOYMENT MANAGER_HOME>/bin/wpfadmin.cmd|wpfadmin move --p E0001|W0001 --d
<DESTINATION_SERVER>
----Press <ENTER> to continue..........
```

Issue the commands mentioned in the output to move partitions E0001 and W0001 to different servers. Then hit <ENTER>. The following will be displayed in the application client window:

```
1.Create Account E_ACCT0001 with balance 1000 for Jian
    in server <SERVER1>
2.Create Account W_ACCT0001 with balance 2000 for Josh
    in server <SERVER2>

----Check the server to see the routing is correct.
----Check your database. The previous records are added to two different databases you
have configured.
    Press <ENTER> to continue..........
```

Make sure <SERVER1> is the server to which you just moved partition E0001, and <SERVER2> is the server to which you just moved partition W0001.

You can also check the database to make sure account W_ACT0001 is created in westtest database, and account E_ACCT0001 is created in easttest database. Then hit <ENTER>. And the following will be displayed in the application client window:

```
3.Delete Account W_ACCT0001
4.Delete Account E_ACCT0001
5.Sample finishes.
```

5.5.8 DataStore Helper classes.

Currently, WebSphere Extended Deployment 5.1 Proxy DataSource only supports DB2 UDB and Oracle.

When creating the proxy datasource for DB2, the following datastore helpers have to be used:

- com.ibm.websphere.proxyds.helper.DB2ProxyDSDataStoreHelper: for DB2 CLI-based legacy JDBC driver
- com.ibm.websphere.proxyds.helper.DB2UniversalProxyDSDataStoreHelper: for DB2 universal JDBC driver

When creating the proxy datasource for Oracle, the following datastore helper has to be used:

- com.ibm.websphere.proxyds.helper.OracleProxyDSDataStoreHelper

5.5.9 Restrictions

There are some restrictions for the proxy datasource support in WebSphere Extended Deployment 5.1. Please refer section 6.4 for the proxy datasource restrictions and tips.

5.6 WPF PMI Client Programming

This section describes how to subscribe to WPF PMI statistics (see the management section describing how to enable this infrastructure and the PSSB programming section to include reportTransactionComplete(…) in your application).

5.6.1 Subscribe WPF PMI statistics using WPFJMX MBean.

Instead of using wpfadmin tool, we can also use WPFJMX MBean to subscribe WPF PMI. Here we describe the following ways to get the MBean instance.

5.6.2 subscribe WPF PMI statistics using Java code

Here is an example of how to subscribe and unsubscribe WPF PMI statistics using Java code. Users can modify them to change statistics types, ranges, etc.


```
package com.ibm.websphere.wpf.jmx;

import java.util.Properties;
import java.util.Set;

import javax.management.InstanceNotFoundException;
import javax.management.MBeanException;
import javax.management.MalformedObjectNameException;
import javax.management.ObjectName;
import javax.management.ReflectionException;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;

/**
 * Creates a node scoped resource.xml entry for a DB2 XA datasource.
 * The datasource created is for CMP use.
 *
 * You need following to run with the classpath
 * $WAS HOME\lib\admin.jar;
 * $WAS HOME\lib\wasjmx.jar;
 * $WAS_HOME\lib\wasx.jar
 */
public class PMIJMXSample {

    /**
     * Main method.
     * arg[0] is the connection type, such as SOAP.
     * arg[1] is the connection host, such as localhost.
     * arg[2] is the connections port, such as 8879
     * arg[3] is the optional process. Default is Deployment Manager
     */
    public static void main(String[] args)
        throws
            MalformedObjectNameException,
            ConnectorException,
            MBeanException,
            ReflectionException,
            InstanceNotFoundException {

        System.out.println("Sample starts");

        // Initialize the AdminClient.
        String connectorType = args[0];
        String connectorHost = args[1];
        String connectorPort = args[2];

        String jmxServer = "Deployment Manager";
        if (args.length >= 4) {
            jmxServer = args[3];
        }

        Properties adminProps = new Properties();
        adminProps.setProperty(AdminClient.CONNECTOR TYPE, connectorType);
        adminProps.setProperty(AdminClient.CONNECTOR HOST, connectorHost);
        adminProps.setProperty(AdminClient.CONNECTOR PORT, connectorPort);
        AdminClient adminClient = AdminClientFactory.createAdminClient(adminProps);

        ObjectName queryName = new ObjectName("WebSphere:type=WPF,process=" + jmxServer
+ ",*");

        ObjectName wpfJMX = null;
        Set s = adminClient.queryNames(queryName, null);
        if (!s.isEmpty()) {
            wpfJMX = (ObjectName) s.iterator().next();
        }
        else {
            System.out.println("WPF JMX MBean was not found");
            System.exit(0);
        }

        String subscribeWPFPMIName = "subscribeWPFPMI";
        String[] subscribeSignatures =
            new String[] {
                "java.lang.String",
                "java.lang.String",
                "java.lang.Integer",
                "java.lang.Integer",
                "java.lang.Integer",
```



```
                "java.lang.Integer" };
        Object[] subscribeParams =
            new Object[] {
                "WPFKeyBasedPartitionSample",
                "WPFKeyBasedPartition",
                new Integer(0),
                new Integer(1),
                new Integer(5),
                new Integer(60000)};

        // invoke the subscribeWPFPMI method
        Long id = (Long) adminClient.invoke(wpfJMX, subscribeWPFPMIName,
subscribeParams, subscribeSignatures);

        System.out.println("Subscription ID is " + id);

        String unsubscribeWPFPMIName = "unsubscribeWPFPMI";
        String[] unsubscribeSignatures = new String[] { "java.lang.Long" };
        Object[] unsubscribeParams = new Object[] { id };

        // invoke the unsubscribeWPFPMI method

        adminClient.invoke(wpfJMX, unsubscribeWPFPMIName, unsubscribeParams,
unsubscribeSignatures);

    }

}
```

5.6.3 subscribe WPF PMI statistics using Jacl code

Users can write a jacl script and then use "wsamdin –f *JACL_FILE*" to subscribe WPF PMI. Here is an Jacl example of subscribing and unsubscribing WPF PMI.

```
# Jacl script to show how to subscribe and unsubscribe WPF PMI

# get WPF Mbean
set wpf [lindex [$AdminControl queryNames "type=WPF,process=Deployment Manager,*"] 0]
puts $wpf

# subscribe WPF PMI
set id [$AdminControl invoke $wpf subscribeWPFPMI "WPFKeyBasedPartitionSample
WPFKeyBasedPartition 0 1 5 60000"]
puts $id

# unsubscribe WPF PMI
$AdminControl invoke $wpf unsubscribeWPFPMI $id
```

5.6.4 subscribe WPF PMI statistics using Jython code

Users can write a jython script and then use "wsamdin –lang jython –f *JYTHON_FILE*" to subscribe WPF PMI. Here is an jython example of subscribing and unsubscribing WPF PMI.

```
# Jython script to show how to subscribe and unsubscribe WPF PMI

# get WPF Mbean
strObjectName=AdminControl.queryNames("type=WPF,process=Deployment Manager,*")
objectName = AdminControl.makeObjectName(strObjectName)
wpf = TypedProxy.makeProxy(AdminControl, objectName, "com.ibm.websphere.wpf.jmx.WPFJMX")

# subscribe WPF PMI
id = wpf.subscribeWPFPMI("WPFKeyBasedPartitionSample", "WPFKeyBasedPartition", 0, 1, 5,
60000)
print id

# unsubscribe WPF PMI
wpf.unsubscribeWPFPMI(id)
```


5.7 HTTP Partitions

This section describes the necessary packaging and deployment actions required to develop HTTP WPF applications.

5.7.1 Anatomy of An HTTP WPF Application

As in IIOP WPF, each HTTP WPF application must also contain a Partitioned Stateless Session Bean (PSSB). This bean interacts with a HttpPartitionManager interface to make requests of HTTP WPF. An application may optionally contain one or more web modules that may also use the HttpPartitionManager. A Servlet may register for WPF events such as the loading and unloading of partitions by the WPF subsystem. Finally, in addition to the application.xml contained in the META-INF directory of an enterprise application archive (EAR), an application may also specify partition information in a partitions.xml file.

Websphere Application Server Extended Deployment ships with a sample HTTP WPF application in the archive httpwpfsample.ear. This is located in the <WAS_ROOT>/installableApps directory by default. This application contains an application-generic PSSB, a web archive with a simple Servlet, and the appropriate packaging including a sample partitions.xml file. Its contents, though subject to change, are included here:

```
1  http.wpf.generic.ejb.jar
2  http.wpf.sample.web.war
3  META-INF/MANIFEST.MF
4  META-INF/application.xml
5  META-INF/partitions.xml
   doc\readme.txt
```

5.7.2 Packaging: Specifying HTTP Partitions in partitions.xml

The HTTP Partitioning function requires the specification of two configuration lists: expressions and partitions. The expression list consists of all regular expressions that will be used to extract valid partition names from incoming HTTP requests. The partitions list contains all valid partition names identifying all partitions that should be managed by HA Manager and activated in back-end target servers.

There are two ways for supplying this information: 1) using the HttpPartitionManager, and 2) including a partitions.xml file in the META-INF directory of an enterprise archive. A sample partitions.xml file is provided here:

```
6  <?xml version="1.0" encoding="UTF-8"?>
7  <wpfhttp:Partitions xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
   xmlns:wpfhttp="http://www.ibm.com/websphere/application server/schemas/6.0/wpfhttp.xmi"
   xmi:id="http.wpf.sample">
8    <ExpressionList xmi:id="ExpressionList_1095302397199">
9      <RequestExpression xmi:id="RequestExpression_1095302397199"
   MatchExpression="(user=)(.*)$" ClassifyFormula="$2"/>
10   </ExpressionList>
11   <PartitionList xmi:id="PartitionList_1095302397199">
12     <PartitionEntry xmi:id="PartitionEntry_1095302397199" Name="adolfo"/>
13     <PartitionEntry xmi:id="PartitionEntry_1095302397200" Name="isabelle"/>
14   </PartitionList>
15 </wpfhttp:Partitions>
```

This partitions file identifies matching regular expressions (in the ExpressionList section) with two strings: the match expression and the classify formula. The match expression determines how we match on a portion of the URL and query string. The classify formula indicates the portion of the URL and query string that specifies the partition once the expression has been matched.

Note that XML has a special set of characters that cannot be used in normal XML strings. Most notably among them is the ampersand character, '&', which is the character used to separate variables in the query string. As a result, a match expression containing the ampersand character will result in generated XML such as this:

```
16     <RequestExpression xmi:id="RequestExpression_1095302397199"
   MatchExpression="(user=)(.*)&amp;" ClassifyFormula="$2"/>
```

Though not recommended, one may modify a partitions.xml manually. Care must be taken to ensure that these special set of characters is not used when editing a partitions.xml file. For more information on these characters, please refer to the XML specification.

In lieu of manual edit, an Eclipse (WebSphere Studio Application Developer) plug-in will be made available via web download at the http://www.ibm.com/developerworks web site.
The second portion of this file (the PartitionList) is much more straightforward as it simply lists all valid partition names. This list is communicated with WPF during application startup to inform WPF of the partitions it should activate.

5.7.3 Packaging: HttpPartitionBean: A Generic PSSB

The HTTP Partitioning function provides a generic Partitioned Stateless Session Bean (named HttpPartitionBean) to interact with WPF. It is contained within the EAR file of the sample HTTP WPF application (httpwpfsample.ear). As such, all HTTP WPF applications must include this EJB as a module in the enterprise archive. As a result, the application.xml of the EAR file should include a section for this module, such as:

```
17      <module id="EjbModule_1090414118785">
18          <ejb>http.wpf.generic.ejb.jar</ejb>
19      </module>
```

Note that this EJB may be modified by the user to invoke HttpPartitionManager services or handle partition events. As an alternative, a Servlet may invoke the HttpPartitionManager directly. In fact, an application may invoke HTTP Partitioning with both the PSSB and Servlets simultaneously.

5.7.4 Deployment: Co-locating the Generic PSSB and Servlets

The HttpPartitionBean provides the mechanism for triggering WPF events in the HTTP WPF application. As such, it is imperative that this bean reside on the same target servers as each partition-enabled Servlet. In the general and recommended case, all Servlets and the HttpPartitionBean should map to the same cluster, however, this is not a strict requirement. Different partition-enabled Servlets (web modules) may be mapped to different clusters only if the HttpPartitionBean is active on all servers where a Servlet may come active.

5.7.5 Packaging: The HttpPartitionFilter

Each Servlet contained within a web application that accesses a partition must make use of the HttpPartitionFilter. This filter ensures that incoming HTTP requests destined for a specific partition are delivered correctly to the application instance housing that partition. Small timing windows exist during partition movement where an ODR's partition-to-server mapping is temporarily out-of-date (due to a small delay in propagation this information from the target servers to the ODR). In these case, the Servlet filter ensures that the request is not delivered to an incorrect application instance. Rather, the Servlet filter detects the inconsistency and redirects the request to the appropriate target server.

To insert the HttpPartitionFilter into the processing chain, the filter must declared and mapped in the web.xml deployment descriptor of the web module archive (WAR file). An example of this is illustrated here:

```
20      <filter>
21          <filter-name>HttpPartitionFilter</filter-name>
22          <display-name>HttpPartitionFilter</display-name>
23          <filter-class>com.ibm.websphere.http.wpf.HttpPartitionFilter</filter-class>
24      </filter>
25      <filter-mapping>
26          <filter-name>HttpPartitionFilter</filter-name>
27          <url-pattern>/*</url-pattern>
28      </filter-mapping>
```

5.7.6 Deployment: Loading Servlets at Start-Up

When using the Servlet API, a Servlet may specify the expressions and partitions associated with an HTTP WPF application. As a result, the Servlet must load at application start-up to ensure that these specifications are available to

the HTTP Partitioning function, and subsequently the ODR. In this manner, the ODR has the sufficient information to forward requests to the appropriate target servers. Note that this is not a requirement if the Servlet API is not used to specify expressions and partitions, such as is the case with using the EJB API for this or when including a partitions.xml file with the application.

This can be done by specifying the <load-on-startup> directive with some negative value in the Servlet definition (in web.xml of the web archive):

```
29      <servlet>
30          <servlet-name>ListPartitions</servlet-name>
31          <display-name>ListPartitions</display-name>
32          <servlet-class>http.wpf.sample.web.ListPartitions</servlet-class>
33      <load-on-startup>-1</load-on-startup>
34      </servlet>
```

5.7.7 HTTP Programming Interfaces

The HTTP WPF API can be used by EJBs and Servlets to perform the following actions:

1. Specify request expression and valid partition names to WPF and HTTP Partitioning
2. Receive notification regarding load and unload events of partitions
3. Receive queries to determine if a partition is still alive

The APIs leverage the HttpPartitionManger that can be used to register callbacks for partition events as well as make partition-related requests, such as allocating a request expression.

5.7.7.1 The HttpPartitionManager

The cornerstone of the HTTP WPF API is the HttpPartitionManager that can be accessed from both Servlet and EJB environments in this fashion:

```
35  HttpPartitionManager httpPartitionManager = HttpPartitionManager.instance;
```

This interfaces provides all the basic methods used by Servlets and EJBs to interact with the HTTP Partitioning function.

5.7.7.2 The HttpPartitionExpression

Another fundamental building block associated with the HTTP WPF API is the HttpPartitionExpression that represents a request expression, including the match expression and classify formula portions:

```
36  public interface HttpPartitionExpression {
37      /**
38       * Get the expression string.
39       */
40      public String getMatchExpression();
41
42      /**
43       * Get the classify formula.
44       */
45      public String getClassifyFormula();
46  }
```

The HttpPartitionManager is used to allocate an HttpPartitionExpression using the following HttpPartitionManager method:

```
47      /**
48       * Create an HttpPartitionExpression.
49       */
50      public HttpPartitionExpression createHttpPartitionExpression (String expression, String formula);
```

5.7.7.3 The Notification Interface

The Servlet API consists a method for receiving partition callabacks (regarding partition events) using the HttpPartitionNotification interface. The notification interface is as follows:

public interface HttpPartitionNotification {

```
51      /*
52       * Fetch the vector of partition strings from the web module.
53       */
54      public Vector getPartitions();
```



```
55
56      /*
57       * Fetch the array of expressions from the web module.
58       */
59      public HttpPartitionExpression[] getExpressions();
60
61      /*
62       * Queries the web module to determine if the specified partition is still alive.
63       */
64      public boolean isPartitionAlive(String partitionName);
65
66      /*
67       * Indicates that this partition has been loaded by WPF.
68       */
69      public boolean loadEvent(String partitionName);
70
71      /*
72       * Indicates that this partition has been loaded by WPF.
73       */
74      public void unloadEvent(String partitionName);
75  }
```

Though this interface is very similar to that supported by the Application Server WPF function, there are some minor differences. The most notable of these is the inclusion of a getExpressions() method that is used by the HTTP Partitioning function to gather the request expressions that will be used with this application.
A Servlet can register itself (or some other class) to receive notifications by invoking the HttpPartitionManager.registerNotfication() method, for example, in its init() method:

```
76      public void init() throws ServletException {
77              System.out.println(className+": Registering notification ");
78              httpPartitionManager.registerNotification(appName, this);
79      }
80
81      public void destroy() {
82              System.out.println(className+": Deregistering notification ");
83              httpPartitionManager.deregisterNotification(appName, this);
84      }
```

This example also illustrates the use of the HttpPartitionManager.deregisterNotfication() method in the Servlet's destroy() method. For correctness, all registered notifications must be deregistered.

5.7.7.4 Other Servlet API HttpPartitionManager Methods

In addition to registration and deregistration of notifications and creating request expressions, the HttpPartitionManager supports a number of other functions useful to Servlets:

```
85      /**
86       * Get the list (Vector of Strings) of active partitions.
87       */
88      public Vector getActivePartitions(String appName);
89
90      /**
91       * Add a partition.
92       */
93      public void addPartition(String appName, String partitionName);
94
95      /**
96       * Remove a partition.
97       */
98      public void removePartition(String appName, String partitionName) throws
    UnknownPartitionException;
```

The getActivePartitions() method returns the Vector of Strings corresponding to the partitions that the HttpPartitionManager believes to be active at a given time. The addPartition() and removePartition() methods allow a servlet (and an EJB) to register new partition names with HTTP Partitioning. In the case of addPartition, this will result in the activation of the partition in some application instance in the cluster.

5.7.8 An Example

Listed below are the entire contents of the sample HTTP WPF application Servlet.

```
99  public class ListPartitions extends HttpServlet implements Servlet, HttpPartitionNotification {
100   private static String className = "ListPartitions";
101   private static String appName = "http.wpf.sample";
102   private static HttpPartitionManager httpPartitionManager = HttpPartitionManager.instance;
103
104   public void init() throws ServletException {
105           System.out.println(className+": Registering notification ");
106           httpPartitionManager.registerNotification(appName, this);
107   }
108
109   public void destroy() {
110           System.out.println(className+": Deregistering notification ");
111           httpPartitionManager.deregisterNotification(appName, this);
112   }
113
114   public void doGet(HttpServletRequest req, HttpServletResponse resp)
115           throws ServletException, IOException {
116
117           resp.setContentType("text/html");
118           ServletOutputStream out = resp.getOutputStream();
119           out.println("<html>");
120           out.println("<head><title>Hello World</title></head>");
121           out.println("<body>");
122           out.println("<h1>Hello World</h1><h1>");
123           out.println(listPartitions());
124           out.println("</h1>");
125           out.println("</body></html>");
126   }
127
128   public void doPost(HttpServletRequest req, HttpServletResponse resp)
129           throws ServletException, IOException {
130   }
131
132   public static String listPartitions() {
133           HttpPartitionManager httpPartitionManager = HttpPartitionManager.instance;
134           Vector partitions = httpPartitionManager.getActivePartitions(appName);
135           String output = "Number of Partitions: "+partitions.size()+":\nPartitions:\n";
136           for (int i=0; i< partitions.size(); i++)
137              output = output.concat(partitions.elementAt(i) + "\n");
138           return output;
139   }
140
141   public Vector getPartitions() {
142           System.out.println(className+": getPartitions ");
143           Vector myVect = new Vector();
144           myVect.add("jian");
145           myVect.add("lou");
146           System.out.println(className+": getPartitions number of partitions: "+myVect.size());
147           return myVect;   // Do not override partitions
148   }
149
150   public HttpPartitionExpression[] getExpressions() {
151           System.out.println(className+": getExpressions ");
152           HttpPartitionExpression[] expressions = new HttpPartitionExpression[2];
153           expressions[0] = httpPartitionManager.createHttpPartitionExpression("(user=)(.*)&",
    "$2");
154           expressions[1] = httpPartitionManager.createHttpPartitionExpression("(user=)(.*)$",
    "$2");
155           System.out.println(className+": getExpressions number of expressions
    "+expressions.length);
156           return expressions;   // Do not override expressions
157   }
158
159   public boolean loadEvent(String partitionName) {
160           /*
161            * now is a good time to start caching relevant data
162            */
```



```
163             System.out.println(className+": load "+partitionName);
164
165             return true;
166     }
167
168     public void unloadEvent(String partitionName)
169     {
170             /*
171              * now is a good time to flush relevant cached data
172              */
173             System.out.println(className+": unload "+partitionName);
174     }
175
176     public boolean isPartitionAlive(String partitionName)
177     {
178             /*
179              * can check if a partition is still active
180              */
181             System.out.println(className+": isPartitionAlive ");
182             return true;
183     }
184 }
```

This example illustrates how the HttpPartitionManager is used to specify two partition names ("lou" and "jian" in this case) and two request expressions using the getPartitions() and getExpressions() methods of the notification interface. If this Servlet were not to specify these (if, for example, they were provided in an accompanying partitions.xml file or specified by the EJB), these methods would return null.

5.7.9 The EJB API: Extending HttpPartitionBean

The HttpPartitionBean serves as the link between WPF and HTTP WPF applications. It registers the Application Server WPF PartitionManager with the HttpPartitionManager. When using the Servlet API, the sample HttpPartitionBean shipped with WAS Extended Deployment need not be modified.

One may, however, modify HttpPartitionBean to perform any function available via the Servlet API. By using the EJB API, partitions and expressions may be specified, partitions may be added, and the HttpPartitionManager may be used as in the Servlet API. This section describes in further detail, how this may be done. In addition, it describes the internals of the HttpPartitionBean.

5.7.9.1 EJB-Specific HttpPartitionManager Methods

The HttpPartitionBean is aware of both the Application Server WPF PartitionManager as well as the HttpPartitionManager. These agents work in conjunction to perform partition operations. The HttpPartitionManager contains additional methods that are employed by the HttpPartitionBean, including the setPartitionManager() method that registers the PartitionManager with the HttpPartitionManager. This should be done when initializing the PartitionManager and HttpPartitionManager.

Other methods that can be used by the EJB API include:

```
185     /**
186      * Set the Application Server WPF PartitionManager.
187      */
188     public void setPartitionManager(String appName, PartitionManager manager);
189
190
191
192     /**
193      * Set the HTTP expressions in the WAS config.
194      */
195     public void setExpressions(String appName, HttpPartitionExpression[] expressions);
196
197
```

The setPartitions() and setExpressions() allow the EJB to inform the HttpPartitionManager of partition and expression specifications. If the EJB were to determine the list of partitions and expressions, it would invoke these methods.

5.7.9.2 HttpPartitionBean Details

The following details the contents of the HttpPartitionBean. As is, the default HttpPartitionBean provides the minimal functionality required for HTTP Partitioning. When using only the Servlet API to interact with partitions, no modification is required in this code. This simple PSSB only needs to be supplied with the application.

```
198 public class HttpPartitionBean implements javax.ejb.SessionBean {
199     private javax.ejb.SessionContext mySessionCtx;
200     private PartitionManager partitionManager;
201     private HttpPartitionManager httpPartitionManager;
202
203     /**
204      * getSessionContext
205      */
206     public javax.ejb.SessionContext getSessionContext() {
207             return mySessionCtx;
208     }
209     /**
210      * setSessionContext
211      */
212     public void setSessionContext(javax.ejb.SessionContext ctx) {
213             mySessionCtx = ctx;
214             try
215                {
216                     InitialContext ic = new InitialContext();
217                     partitionManager =
    (PartitionManager)ic.lookup(PartitionManager.JNDI_NAME);
218                     httpPartitionManager = HttpPartitionManager.instance;
219                     partitionManager.setHttpPartitionManager(httpPartitionManager);
220                     String appName = partitionManager.getApplicationName();
221                     httpPartitionManager.setPartitionManager(appName, partitionManager);
222                }
223             catch(Exception e)
224                {
225                     throw new EJBException(e);
226                }
227     }
228     /**
229      * ejbCreate
230      */
231     public void ejbCreate() throws javax.ejb.CreateException {
232     }
233             /**
234              * ejbActivate
235              */
236     public void ejbActivate() {
237     }
238     /**
239      * ejbPassivate
240      */
241     public void ejbPassivate() {
242     }
243     /**
244      * ejbRemove
245      */
246     public void ejbRemove() {
247     }
248
249     /**
250      * @return
251      */
252     public PartitionDefinition[] getPartitions() {
253             return new PartitionDefinition[0];
254     }
255
256     /**
```



```
257      * This is called when a specific partition is assigned to this server process.
258      * @param partitionName
259      * @return
260      */
261     public boolean partitionLoadEvent(String partitionName)
262     {
263             return false;
264     }
265
266     /**
267      * This is called when previously assigned partition is withdrawn from this server.
268      * @param partitionName
269      */
270     public void partitionUnloadEvent(String partitionName)
271     {
272     }
273
274     /**
275      * This may be called periodically to verify that this server is functioning correctly if
276      * it was assigned a partition.
277      * @param partitionName
278      * @return
279      */
280     public boolean isPartitionAlive(String partitionName)
281     {
282             return false;
283     }
284 }
```

The getPartitions() method shown in line 254 simply returns an empty partition definition array since this EJB is not specifying any partitions. If this EJB returned partitions, the setPartitions() method would need to be called on the HttpPartitionManager. Additionally, the EJB could specify HTTP request expressions by invoking the setExpressions() method.

Note, however, that once established with the HttpPartitionBean.getPartitions() (of the session bean), any subsequent partitions must be added with the HttpPartitionManager.addPartition() method.

5.7.10 Mixing Programming Interfaces and partitions.xml

While it is recommended that only one of the three partition and expression specification methods is used with each application (the three methods are partitions.xml, Servlet API, and EJB API), this is not a strict requirement. If partitions or expressions are specified with two, or even three, of the specification methodologies, the **union** of the list of partitions and expressions will be used. This may be advantageous during the development phases of an enterprise application.

6 Problem Resolution

6.1 Client Invocation Problems

6.1.1 Launchclient

Error:

launchclient c:\was\base51\installedApps\Cell\WPFKeyBasedPartitionSample.ear -CCproviderURL=corbaloc::localhost:9813

```
IBM WebSphere Application Server, Release 5.1
J2EE Application Client Tool
Copyright IBM Corp., 1997-2003
WSCL0012I: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application Client class com.ibm.websphere.wpf.client.WPFKeyBasedPartitionClient
WSCL0100E: Exception received: java.lang.reflect.InvocationTargetException
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:85)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:58)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:60)
      at java.lang.reflect.Method.invoke(Method.java:391)
      at com.ibm.websphere.client.applicationclient.launchClient.createContainerAndLaunchApp(launchClient.java:638)
      at com.ibm.websphere.client.applicationclient.launchClient.main(launchClient.java:425)
      at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:85)
      at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:58)
      at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:60)
      at java.lang.reflect.Method.invoke(Method.java:391)
      at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:189)
Caused by: com.ibm.websphere.naming.CannotInstantiateObjectException: Exception occurred while the JNDI NamingManager wa
s processing a javax.naming.Reference object. [Root exception is javax.naming.CommunicationException: A communication fa
ilure occurred while attempting to obtain an initial context with the provider URL: "corbaloc::localhost:9813".  Make sur
e that any bootstrap address information in the URL is correct and that the target name server is running.  A bootstrap
address with no port specification defaults to port 2809.  Possible causes other than an incorrect bootstrap address or
unavailable name server include the network environment and workstation network configuration. [Root exception is org.om
g.CORBA.COMM_FAILURE: WRITE_ERROR_SEND_1  vmcid: 0x49421000  minor code: 50  completed: No]]
      at com.ibm.ws.naming.util.Helpers.processSerializedObjectForLookupExt(Helpers.java:931)
      at com.ibm.ws.naming.urlbase.UrlContextHelper.processBoundObjectForLookup(UrlContextHelper.java:152)
      at com.ibm.ws.naming.java.javaURLContextRoot.processBoundObjectForLookup(javaURLContextRoot.java:398)
      at com.ibm.ws.naming.urlbase.UrlContextImpl.lookup(UrlContextImpl.java:1278)
      at com.ibm.ws.naming.java.javaURLContextRoot.lookup(javaURLContextRoot.java:196)
      at com.ibm.ws.naming.java.javaURLContextRoot.lookup(javaURLContextRoot.java:137)
      at javax.naming.InitialContext.lookup(InitialContext.java:361)
      at com.ibm.websphere.wpf.client.WPFKeyBasedPartitionClient.main(WPFKeyBasedPartitionClient.java:110)
      ... 13 more
```

Possible Causes:

- Cluster has not been made operational.
- Incorrect port used when dispatching the request.
- Cluster is starting, or started, but the partitions have not been activated to receive requests.

Explanation:

In general it can take a few minutes for all partitions to enter the activated state. For the scenarios common in development scenarios, e.g. reasonably few in number often the partitions will be active and ready for work as soon as the server reports “Open for e-business”.

However, as the partition numbers are larger, more cluster members are in the overall cluster or production related work begins, it is important to note that large number of partitions may take several minutes to fully activate. Due to the possible number of partitions, even printing out trace line for one partition for key scenarios could drastically affect performance, the default case is to not print out any specific messages that describe the status of a WPF partition.

However, users can enable a trace spec, WPFSTATUS. This will print out a message for each partition upon activation and other important life cycle events. For those preparing for production, this can be turned on and the general time to document the startup time expected for the particular implementation of WPF. It is suggested that for production scenarios this trace not be used, but rather for pre-production documentation generation or problems in production when it is not clear a partition is activated and it should be. Assuming you have a cluster named "cluster", the trace specification can be set for all nodes with the following command:

wpfadmin setTraceSpec perm WPFSTATUS=all=enabled –c cluster

The following is the example output to expect.

```
wsadmin -lang jython -f wpfadmin.pty setTraceSpec perm WPFSTATUS=all=enabled --c cluster
WASX7209I: Connected to process "Deployment Manager" on node CellManager using SOAP connector;  The type of process is: DeploymentManager
WPFC0065I: Cluster set to cluster
WPFC0059I: Setting trace to WPFSTATUS=all=enabled for Server
cluster_member_1(cells/Cell/nodes/NodeA/servers/cluster_member_1:server.xml#Server_1)
WPFC0059I: Setting trace to WPFSTATUS=all=enabled for Server
cluster_member_2(cells/Cell/nodes/NodeA/servers/cluster_member_2:server.xml#Server_1)
WPFC0059I: Setting trace to WPFSTATUS=all=enabled for Server
cluster_member_3(cells/Cell/nodes/NodeA/servers/cluster_member_3:server.xml#Server_1)
```

After the trace is enabled on the cluster members, trace entries will appear in the log directory for each partition (with other information, this text was filtered). The trace snapshot below is an example:

```
[10/14/04 11:53:53:287 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000002 is successfully created and open
for e-Business at this server.
[10/14/04 11:53:53:647 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000001 is successfully created and open
for e-Business at this server.
[10/14/04 11:53:54:639 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000004 is successfully created and open
for e-Business at this server.
[10/14/04 11:53:55:600 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000008 is successfully created and open
for e-Business at this server.
[10/14/04 11:53:56:041 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000003 is successfully created and open
for e-Business at this server.
[10/14/04 11:53:56:091 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000005 is successfully created and open
for e-Business at this server.
[10/14/04 11:53:56:241 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000010 is successfully created and open
for e-Business at this server.
[10/14/04 11:53:56:411 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000006 is successfully created and open
for e-Business at this server.
[10/14/04 11:53:56:692 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000007 is successfully created and open
for e-Business at this server.
[10/14/04 11:53:56:732 CDT] 1f3052f7 XDClusterAdap I WPFR0002I: Partition routing cluster for partition PK000009 is successfully created and open
for e-Business at this server.
```

In this case, the last partition started at 11:53:56, as compared to the server's own log entry:

```
10/14/04 11:53:47:649 CDT] 719092f5 WsServer      A WSVR0001I: Server cluster_member_2 open for e-business
```

Consequently in this case, it took approximately from 11:53:47 when the application server started until 11:53:56 when the last partition was activated and enrolled in the cluster to receive requests. This is about 9 seconds for 10 partitions. This is not a general rule as startup policies can affect this a great deal and the startup sequence can vary depending on server start order and timing, but the general point is that client requests will be rejected until the partitions are accessible from the client viewpoint.

6.2 Transaction Related

6.2.1 Transaction Rollback Distributed Transaction Time Out

Problem: Transactions might rollback when using WebSphere Partition Feature.

Cause: When you use WebSphere Partition Feature, you might hit TransactionRolledbackException. This could be due to the fact that transactions time out when a distributed transaction is re-imported onto the originating server causing the current transaction to

rollback. This problem has been fixed in PQ93714. For more details, please look at the description of PQ93714

Recommendation: Please contact IBM support for a fix for this problem.

6.3 Workload Routing

6.3.1 Routing Problem determination

If one of your partitions are not activated or all of your partitions are activated but routing cluster data are not created for some activated partition, you will see the following error messages:

```
1st call: PK000573->partiton=PK000573,server=hao/s5
2nd call: PK000573->partiton=PK000573,server=hao/s5
3rd call: PK000573->partiton=PK000573,server=hao/s5
1st call: PK000574->partiton=PK000574,server=hao/s5
2nd call: PK000574->partiton=PK000574,server=hao/s5
3rd call: PK000574->partiton=PK000574,server=hao/s5
java.rmi.RemoteException: CORBA NO_IMPLEMENT 0x49421040 No; nested exception is:

        org.omg.CORBA.NO_IMPLEMENT:   vmcid: 0x49421000  minor code: 40  completed: No
        at com.ibm.CORBA.iiop.UtilDelegateImpl.mapSystemException(UtilDelegateImpl.java:257)
        at javax.rmi.CORBA.Util.mapSystemException(Util.java(Inlined Compiled Code))
        at com.ibm.websphere.wpf.ejb._WPFKeyBasedPartition_Stub.buy(_WPFKeyBasedPartition_Stub.java(Compiled Code))
        at com.ibm.websphere.wpf.ejb._WPFKeyBasedPartition_Stub.buy(_WPFKeyBasedPartition_Stub.java(Compiled Code))
        at com.ibm.websphere.wpf.ejb._WPFKeyBasedPartition_Stub.buy(_WPFKeyBasedPartition_Stub.java(Compiled Code))
        at com.ibm.websphere.wpf.ejb._WPFKeyBasedPartition_Stub.buy(_WPFKeyBasedPartition_Stub.java(Compiled Code))
        at com.ibm.websphere.wpf.ejb._WPFKeyBasedPartition_Stub.buy(_WPFKeyBasedPartition_Stub.java(Compiled Code))
        at com.ibm.websphere.wpf.ejb._WPFKeyBasedPartition_Stub.buy(_WPFKeyBasedPartition_Stub.java(Compiled Code))
        at com.ibm.websphere.wpf.ejb._WPFKeyBasedPartition_Stub.buy(_WPFKeyBasedPartition_Stub.java(Compiled Code))
        at com.ibm.websphere.wpf.ejb._WPFKeyBasedPartition_Stub.buy(_WPFKeyBasedPartition_Stub.java(Compiled Code))
        at com.ibm.websphere.wpf.ejb._WPFKeyBasedPartition_Stub.buy(_WPFKeyBasedPartition_Stub.java(Compiled Code))
        at com.ibm.websphere.wpf.client.WPFKeyBasedPartitionClient$1.run(WPFKeyBasedPartitionClient.java:90)
        at java.lang.Thread.run(Thread.java:567)
Caused by: org.omg.CORBA.NO_IMPLEMENT:   vmcid: 0x49421000  minor code: 40  completed: No
        at com.ibm.ws.wpf.selection.WPFSelectionManager.targetForwarded(WPFSelectionManager.java:450)
        at com.ibm.ws.wlm.client.WLMClientRequestInterceptor.receive_other(WLMClientRequestInterceptor.java(Compiled Code))
        at com.ibm.rmi.pi.InterceptorManager.iterateReceiveOther(InterceptorManager.java(Compiled Code))
        at com.ibm.rmi.corba.ClientDelegate.intercept(ClientDelegate.java(Compiled Code))
        at com.ibm.rmi.corba.ClientDelegate.invoke(ClientDelegate.java(CompiledCode))
        at com.ibm.CORBA.iiop.ClientDelegate.invoke(ClientDelegate.java(Compiled Code))
        at com.ibm.rmi.corba.ClientDelegate.invoke(ClientDelegate.java(CompiledCode))
        at com.ibm.CORBA.iiop.ClientDelegate.invoke(ClientDelegate.java(Compiled Code))
        at org.omg.CORBA.portable.ObjectImpl._invoke(ObjectImpl.java(Inlined Compiled Code))
        ... 11 more


org.omg.CORBA.NO_IMPLEMENT: No Cluster Data Available  vmcid: 0x49421000  minor code: 42  completed: No
        at com.ibm.ws.wpf.plugin.WPFServerRequestInterceptor.forwardRequest(WPFServerRequestInterceptor.java:525)
        at com.ibm.ws.wpf.plugin.WPFServerRequestInterceptor.partitionForward(WPFServerRequestInterceptor.java:400)
        at com.ibm.ws.wpf.plugin.WPFServerRequestInterceptor.checkAndForward(WPFServerRequestInterceptor.java:364)
        at com.ibm.ws.wpf.plugin.WPFServerRequestInterceptor.receive_request_service_contexts(WPFServerRequestInterceptor.java:229)
        at com.ibm.rmi.pi.InterceptorManager.iterateReceiveContext(InterceptorManager.java:669)
        at com.ibm.rmi.iiop.ServerRequestImpl.runInterceptors(ServerRequestImpl.java:122)
        at com.ibm.rmi.iiop.Connection.doWork(Connection.java:2176)
```

If you see the above error message, the first thing to do is to list Active Partitions as follows:

wpfadmin countActivePartitionsOnServers --o 6000

The following execution will similarly occur:

wsadmin -lang jython -f wpfadmin.pty countActivePartitionsOnServers --o 6000

```
WASX7209I: Connected to process "Deployment Manager" on node haoManager using SOAP conn
ector;  The type of process is: DeploymentManager
WPFC0065I: Override set to 5000
```

WPFC0051I: Server haoNetwork\IBMCluster2\s6: 0
WPFC0051I: Server haoNetwork\IBMCluster2\s5: 1,674
WPFC0051I: Server haoNetwork\hao\s3: 1,673
WPFC0051I: Server haoNetwork\hao\s1: 1,673
Total number of partitions is 5020

6.4 Proxy Datasource

6.4.1 Session Bean must use local interface to invoke CMP EJB

The session bean has to use local interface to invoke the CMP EJBs. Remote interfaces of CMP EJBs are not supported and will cause unexpected behavior.

6.4.2 Specify Datasource at Beginning of each Transaction

At the beginning of every transaction (method), the Session bean has to specify the datasource the current transaction will use by using API WSDataSourceHelper.setCurrentDataSourceJNDIName(String). If the session bean does not specify the datasource JNDI name. The first datasource JNDI name from the jndiNames custom property of the proxy datasource will be used, and a warning will be logged.

6.4.3 Performance Monitoring

Problem: Performance Monitoring Instrumentation is not available for statement cache used in Proxy DataSource.

Description: Proxy DataSource uses a special statement cache for caching the prepared statements. In this release, there is no performance monitoring instrumentation for this statement cache.

6.4.4 Test Connection Non-functional

Problem: Test Connection does not work for Proxy DataSource.

Description: "Test Connection" button from the administrative console will not work for proxy datasource. Even if you configure your proxy datasource in a right way, you will see exceptions if click the "Test Connection" button. Trying to use testConnection function from the MBean will not work either.

Recommendation: If you want to make sure the datasource is configured right, click "Test Connection" button of the underlying datasources for the proxy datasource. For example, if your proxy datasource has custom property "jdbc/dsName1;jdbc/dsName2", you can test the connection to datasources with JNDI name jdbc/dsName1 and jdbc/dsName2.

Configuring Proxy DataSource Statement Cache

Topic: Configure the right statement cache size for a Proxy Datasource.

Description: When you create a proxy datasource, please set the statement cache size to 0 in the datasource panel. The right way to configure the statement cache size for the proxy datasource is using the custom property statementCacheSizes. If the value is a single integer, the statement cache sizes for all underlying datasources will be set to that value. Alternatively, you can set the statement cache size for individual underlying datasources using semicolon separator using the format value1;value2;...;value2. For example, 10;20;30 means the statement cache size for the underlying datasource 1 is 10, the statement cache size for the underlying datasource 2 is 20, and the statement cache size for the underlying datasource 3 is 30.

6.4.5 Override the Datastore Helper class when creating the proxy datasource.

Topic: Override the Datastore Helper class when creating the proxy datasource.

Description: Currently, there is only two proxy datasource JDBC provider for all supported databases, one for non-XA, and one for XA. Users have to specify the right datastore helper class when creating the datasources. If a wrong kind of datastore helper class is specified, users might get exceptions. For example, if your underlying datasources use DB2 Universal JDBC provider, the datastore helper class should be orverriden as "com.ibm.websphere.proxyds.helper.DB2UniversalDSProxyDataStoreHelper.

