Estimated time 1:30

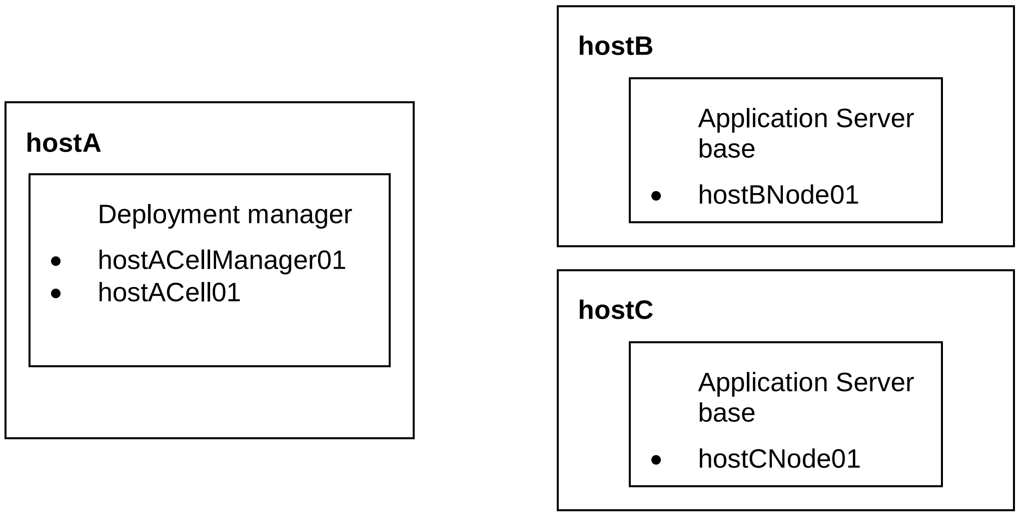
WebSphere eXtreme Scale: **Partitioning facility**

What this exercise is about

The objective of this lab is to introduce the partitioning facility included in WebSphere Extended Deployment V6.1 to provide a basic understanding of the behaviors of a partitioning application. It looks at the basics of administering a partitioning application, the programming model for creating a partitioning application, the use of polices to control placement of partitions on servers and finally addresses the use of an EJB client to a partitioning application.

Lab requirements

This lab assumes that this setup is complete before starting the lab:



- The lab requires three machines: hostA, hostB, hostC.
- Deployment manager installed on hostA
- hostB and hostC are used to run two Application Server nodes.

What you should be able to do

At the end of this lab you should be able to understand, use and describe:

- The basic requirements for implementing a Stateless Session EJB required for implementing a partitioning application.
- The default runtime handling of partitions and their allocation to servers in a cluster.
- A few of the commands provided by the wpfadmin command line tool.
- The definition and application of special policies to modify the default handling of partitions.
- The process of enabling a partitioning application to be used by EJB clients.

Introduction

In this lab exercise you will install an application which contains the minimum application requirements for having a partitioning application (essentially a Stateless Session EJB supporting some special partitioning interfaces) along with a few simple business logic methods. This application will be used to introduce the concepts of partitioning. The exercise will use administrative tools and demonstrate the behavior patterns of partitions within a running clustered WebSphere environment. These behaviors include:

- Assignment of partitions to servers within a cluster

- Automatic movement of partitions between servers for failover
- Administrative movement of partitions between servers
- Influence on behavior using various policy options

The major sections of the exercise are:

- Basic partition administration and default partition behavior
- The partitioned application session bean
- Using cluster wide policy with an application
- Using partition classification policy with an application
- Calling methods on the partitioned application

Exercise instructions

Some instructions in this lab are Windows operating-system specific. If you plan on running the lab on an operating-system other than Windows, you will need to run the appropriate commands, and use appropriate files (.sh or .bat) for your operating system. The directory locations are specified in the lab instructions using symbolic references, as follows:

Reference variable	Windows® location	AIX®/UNIX® location
<WAS_HOME>	C:\WebSphere\AppServer	/usr/WebSphere/AppServer /opt/WebSphere/AppServer
<LAB_FILES>	C:\LabfilesXD	/tmp/LabfilesXD
<LAB_NAME>	PartitionLab	PartitionLab

Windows users note: When directory locations are passed as parameters to a Java program such as EJBdeploy or wsadmin, it is necessary to replace the backslashes with forward slashes to follow the Java convention. For example, replace C:\LabFilesXD\ with C:/LabFilesXD/

The exercise is divided into seven parts. The final part, **Appendix: Common tasks used throughout the exercise** is not intended to be a unified part of the exercise, but instead contains several independent tasks described in step-by-step detail. These tasks are then referenced when they need to be performed rather than repeating the detailed instructions over and over again throughout the exercise. Once you are familiar with a task, you should be able to perform it without referring back to these instructions.

Part 1: Lab setup - Configure the environment

In this part you will do everything that is needed to configure the environment that will be used for this lab exercise. It will consist of:

- Starting the Deployment Manager and Node Agents which are needed to manage the configuration and operations of the cell.
- Configuring a cluster. Note that this is a normal WebSphere cluster with no special characteristics associated with it for enabling partitioning.
- Installing the sample application. This application contains the required structure to enable partitioning.
- Binding a property into the name space. This property is used by the sample application to control which test scenario should be run.

____ 1. Start the Deployment Manager on hostA

__ a. On **hostA**, open a command prompt

__ b. Change directories to **C:\WebSphere\AppServer\profiles\dmgr\bin**

__ c. Enter the command **startmanager** to start the Deployment Manager.

__ d. Wait for the deployment manager to start. Verify that this line appears in the command prompt window.

```
ADMU3000I: Server dmgr open for e-business; process id is XXXX
```

____ 2. Start the Node Agent on hostB

__ a. On **hostB,** open a command prompt

__ b. Change directories to **C:\WebSphere\AppServer\profiles\hostBnode01\bin**

__ c. Enter the command **startnode** to start the Node Agent.

__ d. Wait for the node agent to start. Verify that this line appears in the command prompt window.

```
ADMU3000I: Server nodeagent open for e-business; process id is XXXX
```

____ 3. Start the Node Agent on hostC

__ a. On **hostC,** open a command prompt

__ b. Change directories to **C:\WebSphere\AppServer\profiles\hostCnode01\bin**

__ c. Enter the command **startnode** to start the Node Agent.

__ d. Wait for the node agent to start. Verify that this line appears in the command prompt window.

```
ADMU3000I: Server nodeagent open for e-business; process id is XXXX
```

____ 4. Open the administrative console used to administer the WebSphere environment.

__ a. On **hostA**, open a Web browser

__ b. Enter the URL: **http://localhost:9060/ibm/console/**

__ c. Enter a userID of your choice (for example, **wpfuser**) and click **OK**

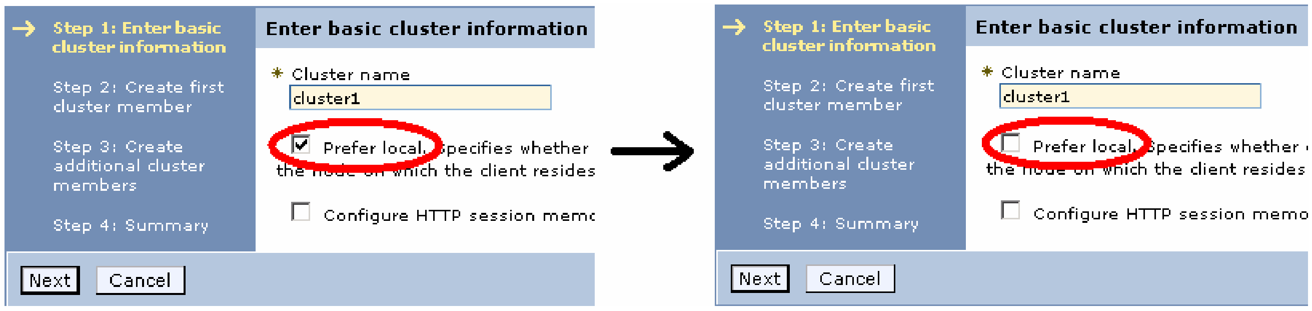
____ 5. Define the basic cluster information for cluster1

__ a. Starting in Navigation Tree (on the left side of the administrative console), open **Servers** and select **Clusters**

__ b. Click **New**

__ c. For Cluster name, enter **cluster1**

__ d. Uncheck "Prefer local" within the basic cluster information. This is required for a cluster that will contain members that use the Partitioning Facility, since calls to a partitioned EJB instance must be routed to the proper server instance, which may not be local to the client server.


Step 1: Enter basic cluster information
Step 2: Create first cluster member
Step 3: Create additional cluster members
Step 4: Summary

Enter basic cluster information

* Cluster name: cluster1
[✓] Prefer local. Specifies whether ... the node on which the client resides
☐ Configure HTTP session memo...

Next | Cancel

→

☐ Prefer local. Specifies whether ... the node on which the client resides

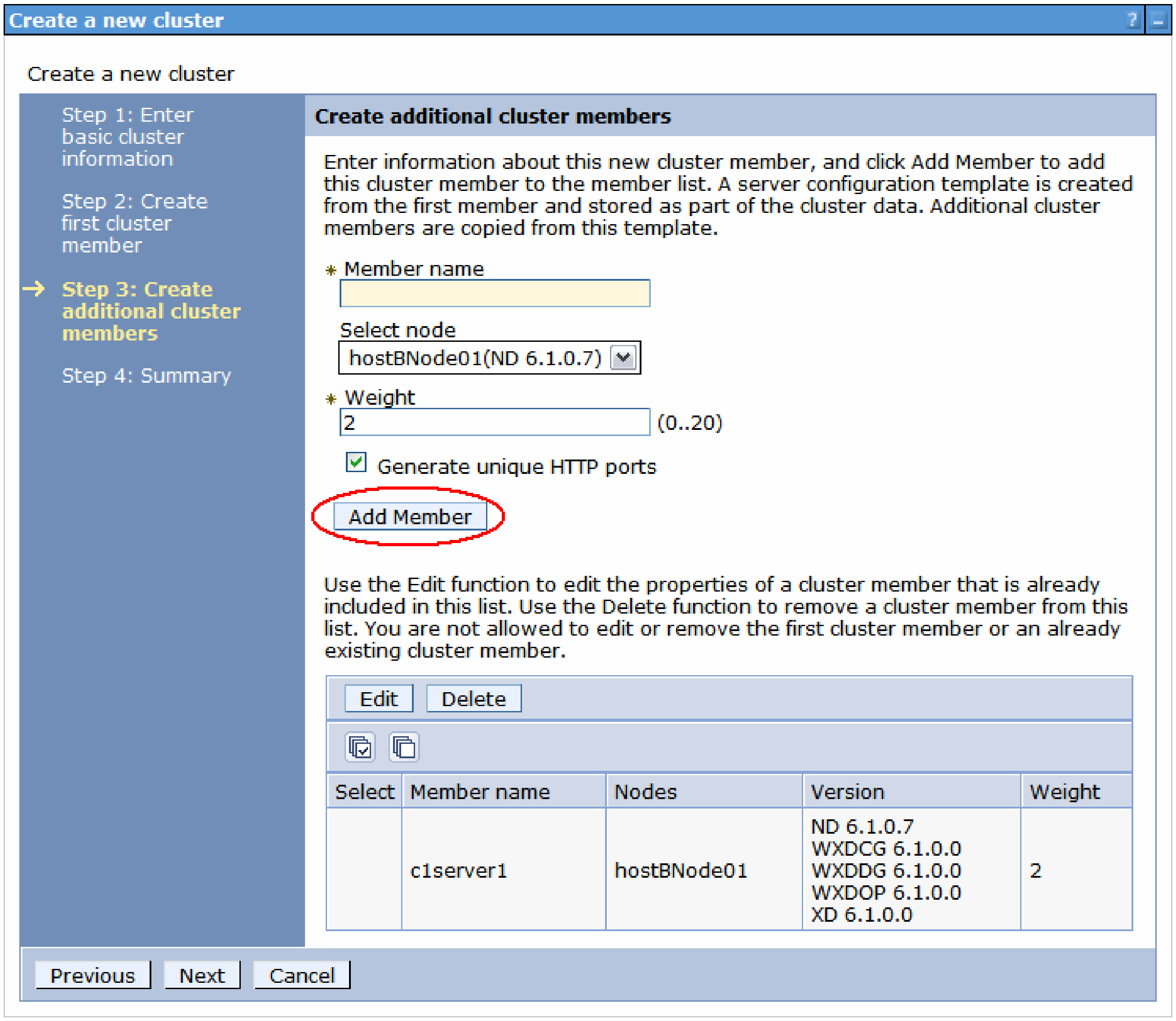
__ e. Click **Next**

____ 6. Define the servers in the cluster

__ a. For Member name, enter **c1server1**

__ b. For Select Node, select **hostBnode01** and then click **Next**

__ c. Now that one member is defined an **Add member** button now appears. The cluster member you just defined, c1server1, should appear in the list at the bottom of the panel.



Create a new cluster

Create a new cluster

Step 1: Enter basic cluster information

Step 2: Create first cluster member

→ **Step 3: Create additional cluster members**

Step 4: Summary

Create additional cluster members

Enter information about this new cluster member, and click Add Member to add this cluster member to the member list. A server configuration template is created from the first member and stored as part of the cluster data. Additional cluster members are copied from this template.

* Member name

Select node
hostBNode01(ND 6.1.0.7)

* Weight
2 (0..20)

✓ Generate unique HTTP ports

Add Member

Use the Edit function to edit the properties of a cluster member that is already included in this list. Use the Delete function to remove a cluster member from this list. You are not allowed to edit or remove the first cluster member or an already existing cluster member.

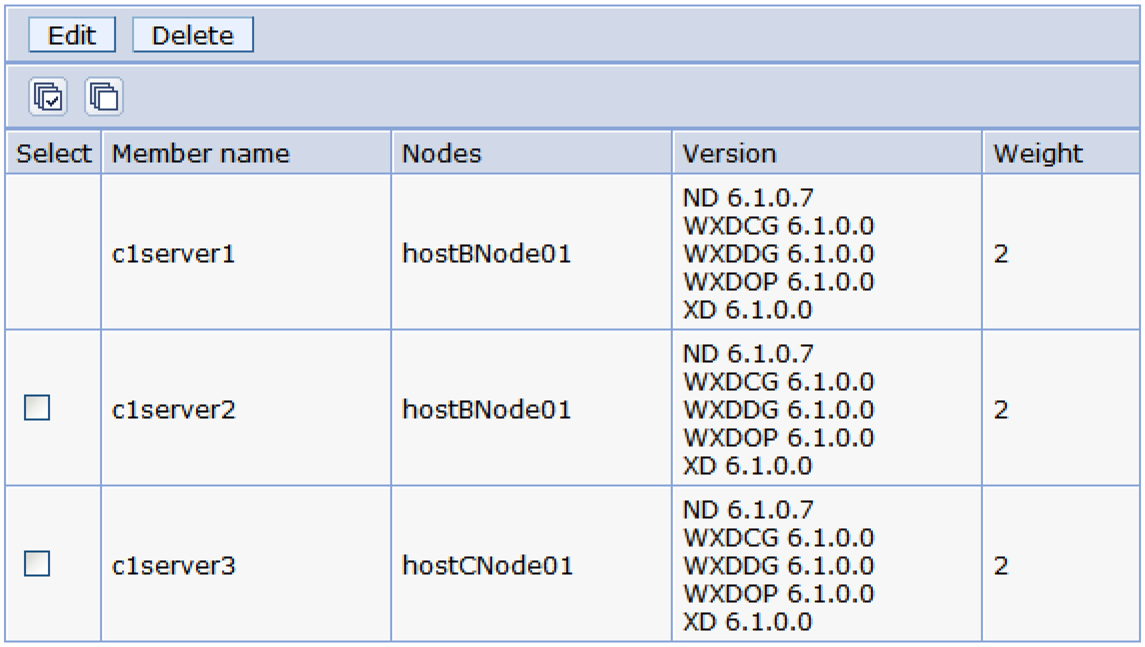
Edit Delete

Select	Member name	Nodes	Version	Weight
	c1server1	hostBNode01	ND 6.1.0.7 WXDCG 6.1.0.0 WXDDG 6.1.0.0 WXDOP 6.1.0.0 XD 6.1.0.0	2

Previous Next Cancel

__ d. In the Member name field, type **c1server2,** select **hostBnode01** again, and click **Add member**. The server should appear in the list at the bottom of the panel.

__ e. In the Member name field, type **c1server3,** select **hostCnode01**, and click **Add member**. The server should appear in the list at the bottom of the panel

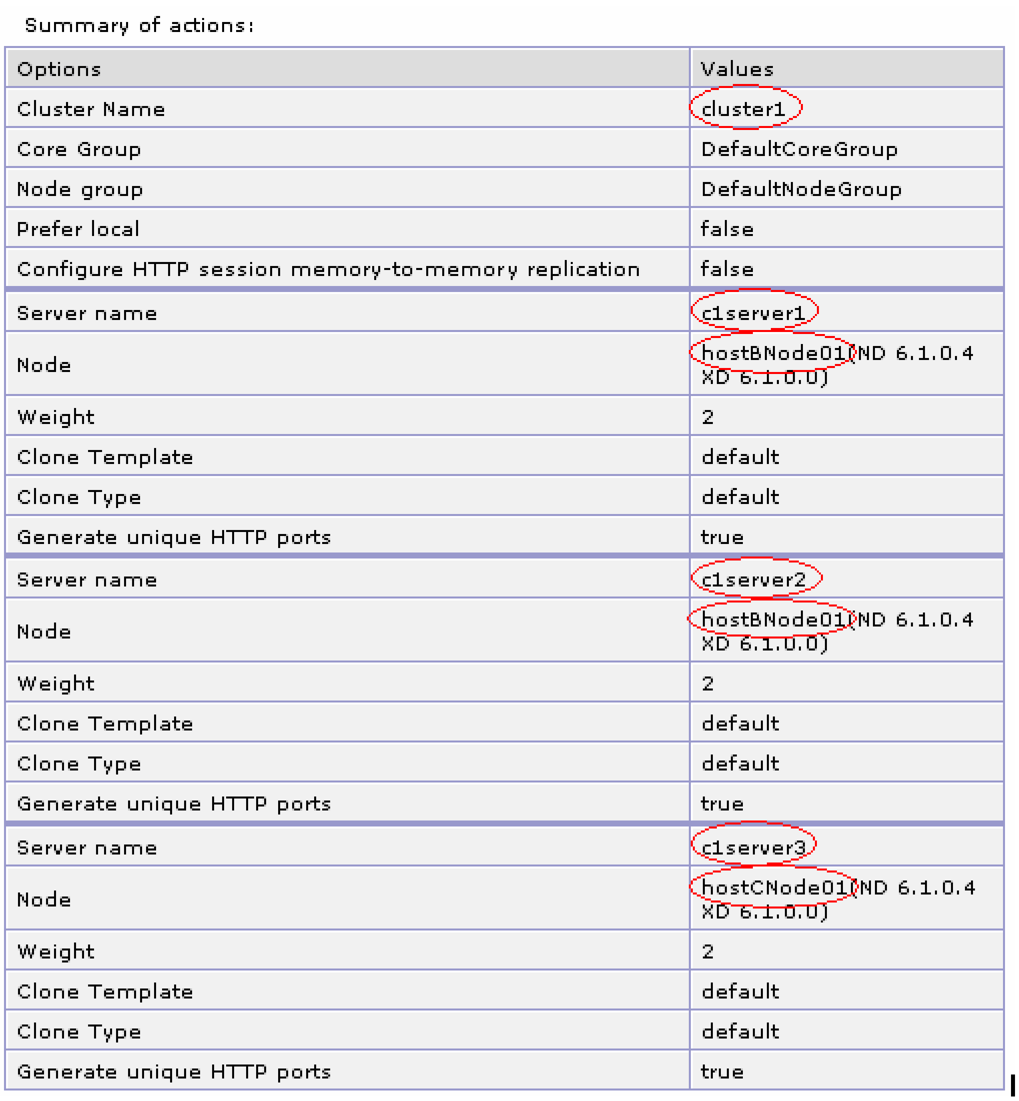


Select	Member name	Nodes	Version	Weight
	c1server1	hostBNode01	ND 6.1.0.7 WXDCG 6.1.0.0 WXDDG 6.1.0.0 WXDOP 6.1.0.0 XD 6.1.0.0	2
☐	c1server2	hostBNode01	ND 6.1.0.7 WXDCG 6.1.0.0 WXDDG 6.1.0.0 WXDOP 6.1.0.0 XD 6.1.0.0	2
☐	c1server3	hostCNode01	ND 6.1.0.7 WXDCG 6.1.0.0 WXDDG 6.1.0.0 WXDOP 6.1.0.0 XD 6.1.0.0	2

____ 7. Complete creation of the cluster.

__ a. Click **Next**, to proceed to the **Summary** panel.

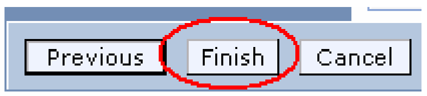
__ b. Review the summary to ensure the highlighted values are set:



Summary of actions:

Options	Values
Cluster Name	cluster1
Core Group	DefaultCoreGroup
Node group	DefaultNodeGroup
Prefer local	false
Configure HTTP session memory-to-memory replication	false
Server name	c1server1
Node	hostBNode01(ND 6.1.0.4 XD 6.1.0.0)
Weight	2
Clone Template	default
Clone Type	default
Generate unique HTTP ports	true
Server name	c1server2
Node	hostBNode01(ND 6.1.0.4 XD 6.1.0.0)
Weight	2
Clone Template	default
Clone Type	default
Generate unique HTTP ports	true
Server name	c1server3
Node	hostCNode01(ND 6.1.0.4 XD 6.1.0.0)
Weight	2
Clone Template	default
Clone Type	default
Generate unique HTTP ports	true

__ c. Click **Finish**.



____ 8. **SAVE the Configuration** (see **Common Task 1: SAVE the configuration on page 34**).

____ 9. Test to ensure the cluster can be successfully started before proceeding with the rest of the exercise.

__ a. **Start the Cluster** (see **Common Task 2: Start the cluster on page 35**).

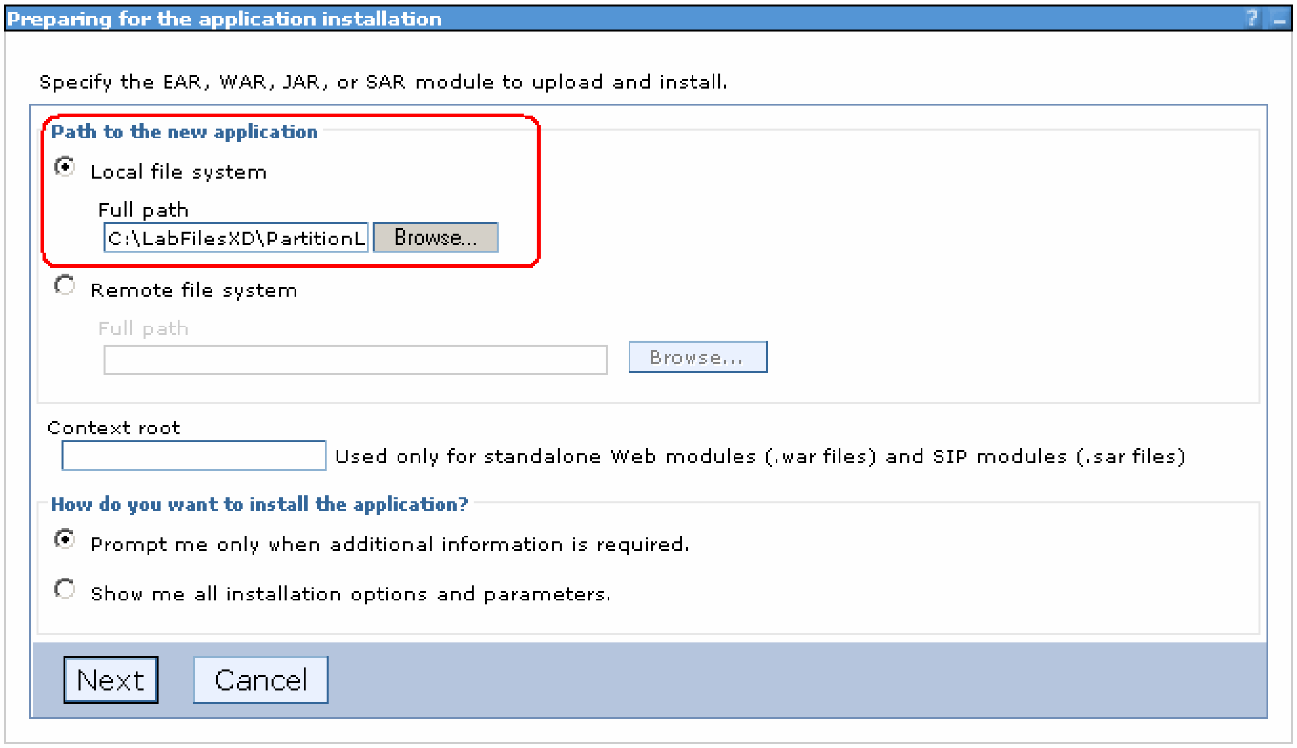
OPTIONAL: You may want to look at the **SystemOut.log** files for the servers to see what a normal server startup looks like before installing a partitioning application. This exercise will not describe anything about the log files, but they can be informative in normal operations to see how the partitioning is working and for debugging when problems occur. If you choose to, you can examine the log files during various stages of the exercise to get a better understanding of the partitioning behavior.

__ b. **Stop the Cluster** (see **Common Task 3: Stop the cluster on page 36**).

____ 10. Install the sample application.

__ a. In the Navigation panel, expand **Applications** and select **Install New Application**.

__ b. Click the **Browse** button for Local file system, and use the browse dialog to select **<LAB_FILES>\<LAB_NAME>\wpfsample1.ear**.


Preparing for the application installation

Specify the EAR, WAR, JAR, or SAR module to upload and install.

Path to the new application

(●) Local file system

Full path: C:\LabFilesXD\PartitionL

( ) Remote file system

Full path:

Context root: Used only for standalone Web modules (.war files) and SIP modules (.sar files)

How do you want to install the application?

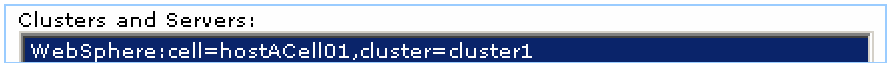
(●) Prompt me only when additional information is required.

( ) Show me all installation options and parameters.

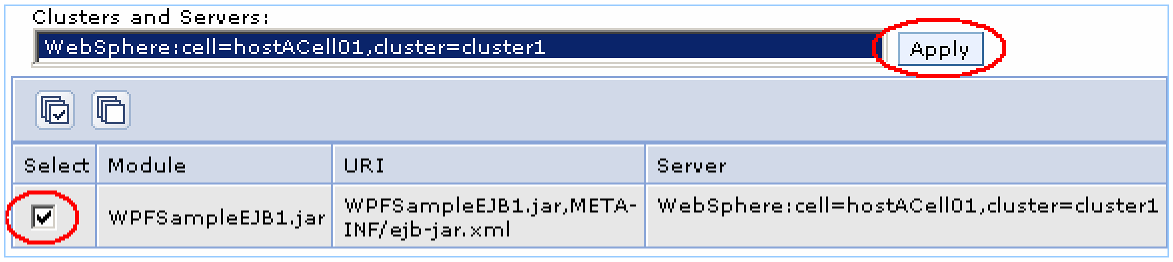
__ c. Click **Next.**

__ d. Click **Next.** You should now be on panel __**Step 2**__ **Map modules to servers.**

__ e. In the **Clusters and Servers** list, select **cluster1**.

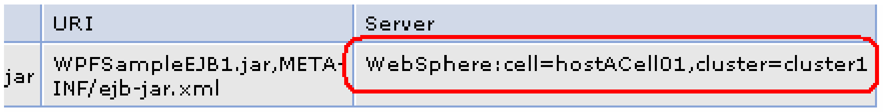

Clusters and Servers: WebSphere:cell=hostACell01,cluster=cluster1

__ f. In the **Module** list check **WPFSampleEJB1.jar,** and then click **Apply.**


Clusters and Servers: WebSphere:cell=hostACell01,cluster=cluster1

Select	Module	URI	Server
✓	WPFSampleEJB1.jar	WPFSampleEJB1.jar,META-INF/ejb-jar.xml	WebSphere:cell=hostACell01,cluster=cluster1

__ g. Ensure that the **Server** has been changed to **cluster1** for **WPFSampleEJB1.jar** (circled in the screen capture below).



	URI	Server
jar	WPFSampleEJB1.jar,META-INF/ejb-jar.xml	WebSphere:cell=hostACell01,cluster=cluster1

__ h. Click **Next** . You should now be on the **Summary** panel.

__ i. Click **Finish** and wait for the installation to complete.

____ 11. Create a name space binding used **only** by logic within the sample application to determine which partitioning scenario should be run.

__ a. In the Navigation panel, expand **Environment ➔ Naming** and select **Name Space Bindings.**

__ b. In the **Name Space Bindings** panel, ensure that the **Scope** is set as **Cell=hostACell01.**

__ c. Click **New**.

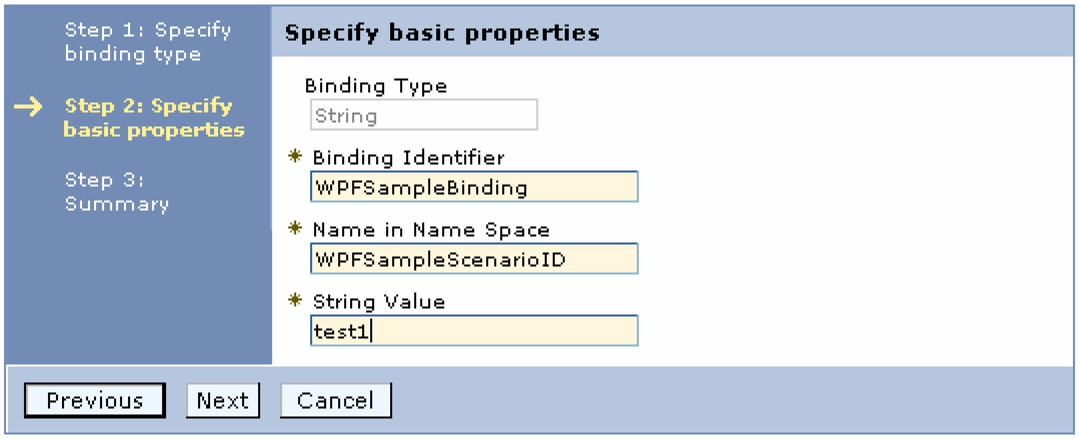
__ d. For **Binding Type,** select the **String** radio button.

__ e. Click **Next**.

__ f. Set **Binding identifier** to **WPFSampleBinding.**

__ g. Set **Name in name space** to **WPFSampleScenarioID.**

__ h. Set **String Value** to **test1.**


Step 1: Specify binding type
Step 2: Specify basic properties
Step 3: Summary
Previous
Next
Cancel

Specify basic properties

Binding Type: String

* Binding Identifier: WPFSampleBinding

* Name in Name Space: WPFSampleScenarioID

* String Value: test1

__ i. Click **Next.**

__ j. Click **Finish.**

____ **12. SAVE the Configuration**

Partition Facility

Part 2: Basic partition administration and default partition behavior

In this part you will be starting and stopping servers that are part of the cluster to see how the behavior of the partitions is affected with different combinations of servers running and with different sequencing of starting and stopping. You will use the wpfadmin command to examine the state of the running partitions and will also use it to make changes in where partitions are running. By the end of this part, you should have an understanding of the default partition behavior.

____ 1. **Start Server = c1server1** (see **Common Task 4: Start server = <servername> on page 36**).

____ **2.** Use the wpfadmin tool to examine the state of the partitions with one server started. Since the default policy requires a quorum of running servers (at least half of the cluster), you would expect that no partitions would be activated with only one server running.

__ a. Run **wpfadmin countActivePartitionsOnServers** (see **Common Task 6: Run wpfadmin countActivePartitionsOnServers on page 38**). This should only report on running servers (in this case *c1server1*) and it should report no partitions active.

```
WPFC0051I: Server hostaCell01\hostBnode01 \c1server1: 0
Total number of partitions is 0
```

__ b. Run **wpfadmin listActive** (see **Common Task 7: Run wpfadmin listActive on page 38**). This should only report on active partitions and therefore there should be no output from the command.

____ **3.** **Start Server = c1server2** (see **Common Task 4: Start server = <servername> on page 36**).

____ 4. Examine the state of the partitions with two servers started. Since there are three servers in the cluster, you now have a quorum. There are nine partitions in the application, so you should see those nine partitions allocated across the two servers.

__ a. Run **wpfadmin countActivePartitionsOnServers.** The expected output should look something like this:

```
WPFC0051I: Server hostaCell01\hostBnode01 \c1server2: 5
WPFC0051I: Server hostaCell01\hostBnode01 \c1server1: 4
Total number of partitions is 9
```

NOTE: Your result might have been reporting **Server hostaCell01\hostBnode01 \c1server1: 0** rather than what appears here. If that is the case, rerun the command. There is a window of time between a server becoming active and the partitioning facility recognizing it. Any time you a run wpfadmin command very close to a system status change and see an unexpected result, give the system a few moments to do its work and rerun the command.

__ b. Run **wpfadmin listActive**. The expected output should look something like the lines below, reporting the server where each specific partition is running. Note that your specific assignment of partitions to servers may be different than what is shown below.

```
WPFC0050I: Application WPFSample1, Partition par9: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par8: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition par7: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par6: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par5: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par4: Server
hostaCell01\hostBnode01 \c1server1
```

```
WPFC0050I: Application WPFSample1, Partition par3: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par2: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition par1: Server
hostaCell01\hostBnode01 \c1server1
```

____ **5.** **Start Server = c1server3** (see **Common Task 4: Start server = <servername> on page 36**).

____ 6. Examine the state of the partitions with three servers started. When an additional server is added after a quorum has been reached, the partitions are not redistributed across all of the servers

__ a. Run **wpfadmin countActivePartitionsOnServers**. The expected output should look something like this, with no new partitions being activated on the new server. Note that there are no new partitions on hostaCell01\hostCnode01 \c1server3.

```
WPFC0051I: Server hostaCell01\hostCnode01 \c1server3: 0
WPFC0051I: Server hostaCell01\hostBnode01 \c1server2: 5
WPFC0051I: Server hostaCell01\hostBnode01 \c1server1: 4
Total number of partitions is 9
```

__ b. Run **wpfadmin listActive**. The expected output should look something like the lines below, reporting the server where each specific partition is running. Notice that the allocation of partitions to servers is the same as in the previous step.

```
WPFC0050I: Application WPFSample1, Partition par9: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par8: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition par7: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par6: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par5: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par4: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition par3: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par2: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition par1: Server
hostaCell01\hostBnode01 \c1server1
```

____ 7. Partitions can be dynamically reassigned to balance the allocation of partitions across running servers. This is accomplished using the **wpfadmin balance --verify** command.

__ a. On hostA, open a Command Prompt window

__ b. Change directories to **C:\WebSphere\AppServer\profiles\dmgr\bin.**

__ c. Enter the command:

```
wpfadmin balance --verify
```

__ d. The expected output should look something like this, with partitions being moved from c1server1 and c1server2 to c1server3.

```
CWPFC0065I: Verify set to true
CWPFC0054I: Move command submitted successfully for partition par2
from Server hostACell01\hostBNode01\c1server2 to Server
hostACell01\hostCNode01\c1server3
CWPFC0084I: Partition has been moved successfully
CWPFC0054I: Move command submitted successfully for partition par9
from Server hostACell01\hostBNode01\c1server1 to Server
hostACell01\hostCNode01\c1server3
```

```
CWPFC0084I: Partition has been moved successfully
CWPFC0054I: Move command submitted successfully for partition par7
from Server hostACell01\hostBNode01\c1server1 to Server
hostACell01\hostCNode01\c1server3
CWPFC0084I: Partition has been moved successfully
CWPFC0089I: Balance command submissions completePartition has been
moved successfully
Balance command submissions complete
```

__ e. Examine the state of the partitions now that the partitions have been balanced across the running servers.

__ f. Run **wpfadmin countActivePartitionsOnServers**. The output should look something like what is shown below.

```
WPFC0051I: Server hostaCell01\hostCnode01 \c1server3: 3
WPFC0051I: Server hostaCell01\hostBnode01 \c1server2: 3
WPFC0051I: Server hostaCell01\hostBnode01 \c1server1: 3
Total number of partitions is 9
```

____ 8. Partitions can be explicitly moved from one server to another using the **wpfadmin move** command. Move one of the partitions from c1server1 to c1server3.

__ a. Run **wpfadmin listActive**. Examine the output and select a partition that is running on c1server1 (the lab will use ***parN*** to denote the chosen partition in the upcoming steps).

__ b. On **hostA**, open a Command Prompt window.

__ c. Change directories to **C:\WebSphere\AppServer\profiles\dmgr\bin**.

__ d. Enter the command (replacing ***parN*** with the selected partition):

```
wpfadmin move --p parN --d hostACell01/hostCnode01/c1server3 --verify
```

__ e. Examine the output, which should look similar to:

```
WASX7303I: The following options are passed to the scripting
environment and are available as argume
nt that is stored in the argv variable: "[move, --p, par6, --d,
hostACell01/hostCnode01/c1server3, -
-verify]"
CWPFC0065I: Partition set to parN
CWPFC0065I: Destination set to hostACell01/hostCnode01/c1server3
CWPFC0065I: Verify set to true
CWPFC0054I: Move command submitted successfully for partition parN
from Server hostACell01\hostBNode01\c1server1 to Server
hostACell01\hostCnode01\c1server3
CWPFC0084I: Partition has been moved successfully.
```

____ **9.** **Stop Server = c1server2** (see **Common Task 5: Stop server = <servername> on page 37**).

____ 10. Examine the state of the partitions now that a server has been stopped and there are only two servers running. The partitions that were on c1server2 should be reassigned across the other two servers.

__ a. Run **wpfadmin countActivePartitionsOnServers**. The output should look something like what is shown below. Note that the two remaining servers should now have all nine partitions between them.

```
WPFC0051I: Server hostaCell01\hostCnode01 \c1server3: 4
WPFC0051I: Server hostaCell01\hostBnode01 \c1server1: 5
Total number of partitions is 9
```

____ **11.** **Stop Server = c1server3** (see **Common Task 5: Stop server = <servername> on page 37**).

____ **12.** Examine the state of the partitions now that another server has been stopped and there is no longer a quorum. Because there is no longer a quorum, all the servers running the application will be stopped (in this case c1server1 will be stopped automatically).

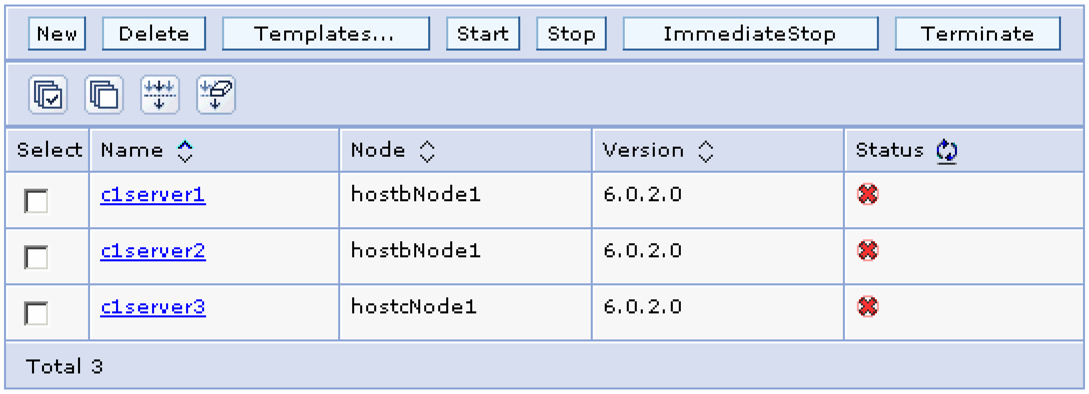
__ a. Run **wpfadmin countActivePartitionsOnServers**. The output should look something like what is shown below. Note that there are no longer any running servers.

```
Total number of partitions is 0
```

____ 13. You can also check the administrative console to see that there are no servers running.

__ a. In the Navigation panel, expand **Servers** and select **Application Servers.**

__ b. Examine the panel to see that all three servers are not running. This is indicated by a red **X** in the status column.



Select	Name	Node	Version	Status
☐	c1server1	hostbNode1	6.0.2.0	✖
☐	c1server2	hostbNode1	6.0.2.0	✖
☐	c1server3	hostcNode1	6.0.2.0	✖

Total 3

Part 3: The partitioned application session bean

An application is enabled for using partitions through use of a Stateless Session EJB which supports the PartitionHandlerLocal interface. In this part you will take a quick look at the implementation of the session bean used in the previous section of this exercise to understand how the partitions are defined. Then you will run a different scenario to see how changing the partitions defined in the session bean affect the runtime.

____ 1. A partitioned session bean must support the interface PartitionHandlerLocal and be associated with the home PartitionHandlerLocalHome. These are set in the deployment descriptor for the EJB.

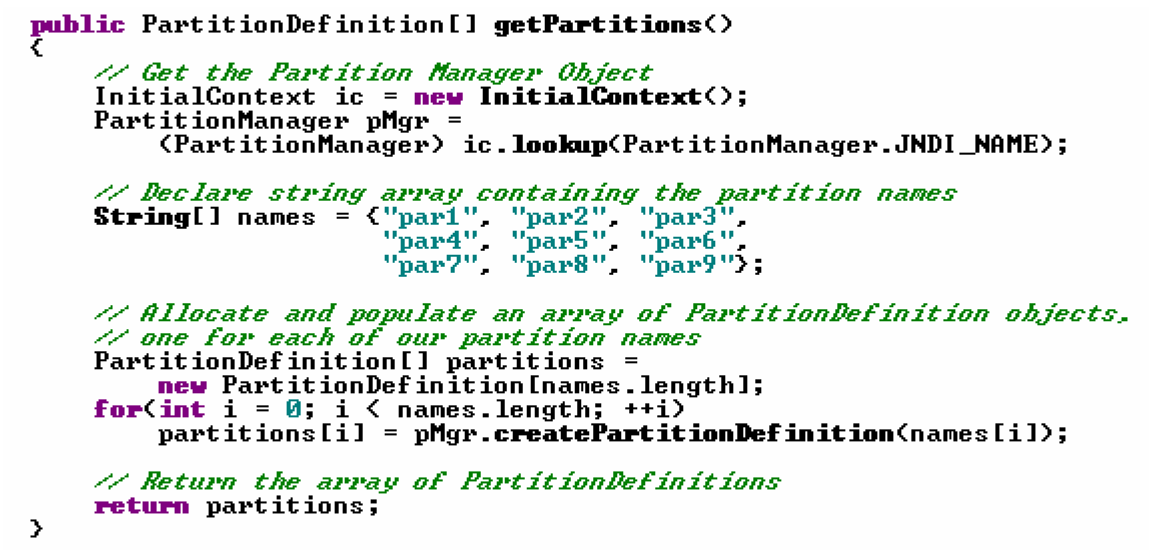
__ a. On **hostA**, open any editor, such as Notepad.

__ b. In the editor, open the file **<LAB_FILES>\<LAB_NAME>\ejb-jar.xml**.

__ c. Examine the contents of the deployment descriptor. Take note of the two following lines which are required for the Stateless Session EJB to be handled as a partitioning application.

```
<local-home>com.ibm.websphere.wpf.PartitionHandlerLocalHome</local-home>
<local>com.ibm.websphere.wpf.PartitionHandlerLocal</local>
```

____ 2. There are several methods that the Session bean must implement to support partitioning. The one that you will examine is the getPartitions() method, which returns an array of PartitionDefinition objects used to define the partitions for the application. These partition definition objects are created using the PartitionManager object. The following code shows the basics needed to implement the getPartitions() method.



```
public PartitionDefinition[] getPartitions()
{
    // Get the Partition Manager Object
    InitialContext ic = new InitialContext();
    PartitionManager pMgr =
        (PartitionManager) ic.lookup(PartitionManager.JNDI_NAME);

    // Declare string array containing the partition names
    String[] names = {"par1", "par2", "par3",
                      "par4", "par5", "par6",
                      "par7", "par8", "par9"};

    // Allocate and populate an array of PartitionDefinition objects.
    // one for each of our partition names
    PartitionDefinition[] partitions =
        new PartitionDefinition[names.length];
    for(int i = 0; i < names.length; ++i)
        partitions[i] = pMgr.createPartitionDefinition(names[i]);

    // Return the array of PartitionDefinitions
    return partitions;
}
```

__ a. You will see that the partition names defined in the above code are the same partition names used in the previous example. The actual session bean used in the exercise is structured slightly differently, but essentially implements the same logic as illustrated above for defining the partitions.

____ 3. View the Java source file for the LetterPartitions EJB that is used in this lab.

__ a. On hostA, open any editor, such as Notepad.

__ b. In the editor, open the file **<LAB_FILES>\<LAB_NAME>\LetterPartitionsBean.java**.

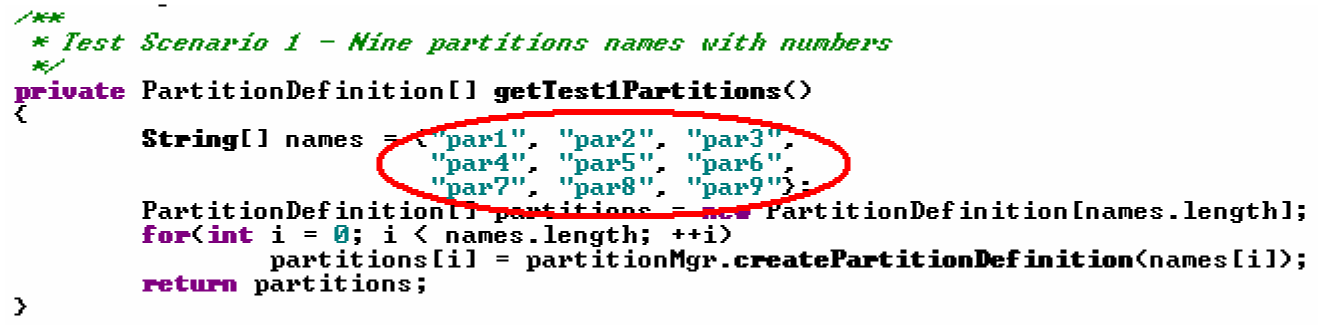
__ c. Find the method **setSessionContext().** You will see that this method obtains the **PartitionManager** object and caches it for use by the **getPartitions()** method.

__ d. Note that in the **setSessionContext()** method, the method performs a lookup on a string bound into the name space at the name **"cell/persistent/WPFSampleScenarioID."** (You may remember adding this binding in the configuration section of this exercise.) This value has nothing to do with normal partitioning implementation, but is used by this sample application as a way to control running different test scenarios. The value that is looked up is cached for future reference when determining which test scenario to run.

__ e. Finally, the setSessionContext() uses the AdminService to determine the name of the server on which it is running. This is cached and is used in the business logic example later in this exercise.

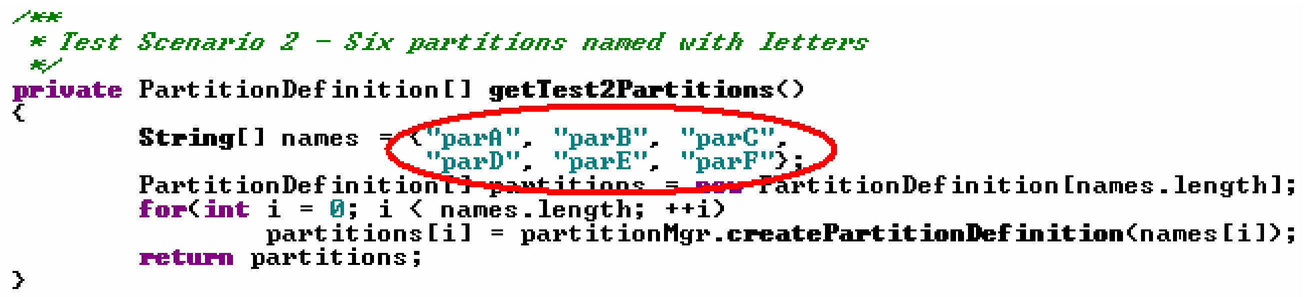
__ f. Find the method **getPartitions()**. Examine the code, and you will see that it uses the scenario value to determine which scenario to run, and calls the appropriate private method to define the partitions for the scenario.

__ g. Find the method **getTest1Partitions()**. You will see that this sets up the test you ran previously in this exercise, with nine partitions named par1 through par9.



```
/**
 * Test Scenario 1 - Nine partitions names with numbers
 */
private PartitionDefinition[] getTest1Partitions()
{
        String[] names = {"par1", "par2", "par3",
                          "par4", "par5", "par6",
                          "par7", "par8", "par9"};
        PartitionDefinition[] partitions = new PartitionDefinition[names.length];
        for(int i = 0; i < names.length; ++i)
                partitions[i] = partitionMgr.createPartitionDefinition(names[i]);
        return partitions;
}
```

__ h. Find the method **getTest2Partitions()**. You will see that this sets up six partitions named parA through parF. This is the scenario you will run next.



```
/**
 * Test Scenario 2 - Six partitions named with letters
 */
private PartitionDefinition[] getTest2Partitions()
{
        String[] names = {"parA", "parB", "parC",
                          "parD", "parE", "parF"};
        PartitionDefinition[] partitions = new PartitionDefinition[names.length];
        for(int i = 0; i < names.length; ++i)
                partitions[i] = partitionMgr.createPartitionDefinition(names[i]);
        return partitions;
}
```

____ 4. **Update Scenario Property = test2** (see **Common Task 8: Update scenario property = <testn> on page 39**).

____ 5. **SAVE the Configuration**.

____ 6. **Start the Cluster** (see **Common Task 2: Start the cluster on page 35**).

____ 7. Examine the state of the partitions with the second scenario running. You should now see six partitions named parA through parF (rather than nine partitions named par1 through par9) because this is what the Stateless Session EJB defined to the partitioning facility.

__ a. Run **wpfadmin countActivePartitionsOnServers**. The output should look something like what is shown below. Note that six partitions are now running on the active servers.

```
WPFC0051I: Server hostaCell01\hostCnode01 \c1server3: 0
WPFC0051I: Server hostaCell01\hostBnode01 \c1server2: 3
WPFC0051I: Server hostaCell01\hostBnode01 \c1server1: 3
Total number of partitions is 6
```

NOTE: It is possible to have 3 partitions on each of two servers and one server with no partitions. The result depends upon the window of time between the second and third server becoming active.

NOTE: On this release of the beta driver, you may encounter a situation where one server contains all six partitions with the other two servers do not contain any partitions. This appears to be a bug as it should not occur when quorum is required. If you see this, you can ignore it as it does not affect the next steps in the exercise.

__ b. Run **wpfadmin listActive**. The output should look something like what is shown below.

```
WPFC0050I: Application WPFSample1, Partition parF: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition parE: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition parD: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition parC: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition parB: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition parA: Server
hostaCell01\hostBnode01 \c1server1
```

____ 8. Stop the Cluster (see **Common Task 3: Stop the cluster on page 36**).

Part 4: Using cluster-wide policy with an application

Policies can be applied to a partitioned application at various levels of granularity. In the previous examples, the default policies were used for managing the partitions. In this part, you will be shown an example of a cluster wide policy applied to the application. The policy will be set up to not require a quorum for the partitions to run. This part will again use test scenario number two. This means that the application will work with six partitions named parA through parF.

____ 1. Examine the properties file used to define the policy to see how the quorum requirement is turned off.

__ a. On hostA, open any editor, such as Notepad.

__ b. In the editor, open the file **<LAB_FILES>\<LAB_NAME>\NoQuorum.properties**.

__ c. The property **QuorumEnabled = false** turns off the requirement for a quorum.

__ d. Policies are applied based on a set of matching criteria specified within the properties file. At runtime, the matching criteria are evaluated and if there is a policy with all matching criteria satisfied, it will be applied. If there is more than one policy where all matching criteria are satisfied, the one with the largest number of criteria is used. Examine the properties file for these matching criteria.

1) **NumOfMatchCriteria = 3** indicates that there are three matching criteria associated with this policy.

2) **Name_0 = -gt** and **Value_0 = -p** indicate that this policy should only be applied to **Partitioning facility** partitions. The HA Manager component which applies policies does so for more components than just the Partitioning Facility and therefore uses this as one of its criteria.

3) **Name_1 = IBM_hc** and **Value_1 = cluster1** indicate that this policy should only be applied to partitions which are running on servers which are members of the cluster named **cluster1**

4) **Name_2 = -pa** and **Value_2 = WPFSample1** indicate that this policy should only be applied to partitions that are part of the application named **WPFSample1**.

____ **2.** **Update Configuration Policies = NoQuorum.properties** (see **Common Task 9: Update configuration policies = <property file name> on page 39**).

____ **3.** **Start Server = c1server1**.

____ **4.** Use the wpfadmin commands to examine the state of the servers with one server running. Since there is no requirement for a quorum, you should see all six partitions (parA through parF) active.

__ a. Run **wpfadmin countActivePartitionsOnServers**.

__ b. Run **wpfadmin listActive**.

____ **5.** **Start Server = c1server2**.

____ **6.** **Start Server = c1server3**.

____ **7.** Use the wpfadmin commands to examine the state of the servers with all three servers running. Since all 6 partitions were already active on c1server1 before the other servers started, they should still all be running on c1server1 with no partitions on the other two servers.

__ a. Run **wpfadmin countActivePartitionsOnServers**.

__ b. Run **wpfadmin listActive**.

____ 8. Balance the partitions across the running servers.

__ a. On **hostA**, open a Command Prompt window.

__ b. Change directories to **C:\WebSphere\AppServer\profiles\dmgr\bin**.

__ c. Enter the command:

```
wpfadmin balance --verify
```

__ d. The expected output should look something like this

```
hostaCell01\hostBnode01 \c1server1 to Server
hostaCell01\hostXNode1\c1serverX
Partitions balanced
WPFC0065I: Verify set to true
Move command submitted successfully for partition parX from Server
hostACell01\hostBNode01\c1server1 to Server
hostACell01\hostXNode01\c1serverX
Partition has not been moved. Make sure the destination server is
correct and there is no policy that conflicts with the move.
Move command submitted successfully for partition parX from Server
hostACell01\hostBNode01\c1server1 to Server
hostACell01\hostXNode01\c1serverX
Partition has not been moved. Make sure the destination server is
correct and there is no policy that conflicts with the move.
Move command submitted successfully for partition parX from Server
hostACell01\hostBNode01\c1server1 to Server
hostACell01\hostXNode01\c1serverX
Partition has not been moved. Make sure the destination server is
correct and there is no policy that conflicts with the move.
Balance command submissions complete
```

____ **9.** Use the wpfadmin commands to examine the state of the servers (Before doing so, use your knowledge of the algorithm used by the balance command to determine the result you expect to see.)

__ a. Run **wpfadmin countActivePartitionsOnServers**.

__ b. Run **wpfadmin listActive**.

__ c. Since all 6 partitions were already active on c1server1 before the other servers started, they should still all be running on c1server1 with no partitions on the other two servers. This is because NoQuorum.properties specifies a list of preferred nodes and servers (listed order of preference). As long as the first server is running, the partitions will remain there, even after a “balance” command.

__ d. Also note that the **wpfadmin balance --verify** in the previous step caused certain “Move” commands to be submitted successfully, but policy settings overrode these commands.

____ **10.** **Stop Server = c1server1**.

____ **11.** Use the wpfadmin commands to examine the state of the servers

__ a. Run **wpfadmin countActivePartitionsOnServers**.

__ b. Run **wpfadmin listActive**.

__ c. Since c1server1 is terminated, you should see all the partitions moved to the second "preferred" server – c1server2.

____ **12.** **Stop Server = c1server2**.

____ **13.** Use the wpfadmin commands to examine the state of the servers. Since there is no quorum requirement and since c1server3 is listed in the list of preferred servers within NoQuorum.properties, the partitions should be reassigned to the last running server. Check c1server3 to see that all six partitions are active.

__ a. Run **wpfadmin countActivePartitionsOnServers**.

__ b. Run **wpfadmin listActive**.

____ **14.** **Stop Server = c1server3**.

Part 5: Using partition classification policy with an application

So far, you have looked at cluster wide policies for an application – both the default policies which require a quorum and a custom policy which did not require a quorum. In this part, you will look at using partition classifications to apply different policies to groups of partitions within the same application. Using these policies, you will assign groups of partitions to specific servers as the primary server for those partitions. In addition, you will assign another server to serve as the backup server for the partitions. This chart shows how you will allocate partitions between the servers.

Classification	Partitions	Primary Server	Backup Server
alpha	parA through parF	c1server1	c1server3
Numeric	par1 through par9	c1server2	c1server3

For example, partitions classified as alpha partitions will run only on c1server or c1server3 but never on c1server2. If both c1server1 and c1server3 are running, the partitions are assigned to c1server1. The classification of a partition occurs when the partition is defined by the session bean and the behavior for the classifications is defined to the configuration.

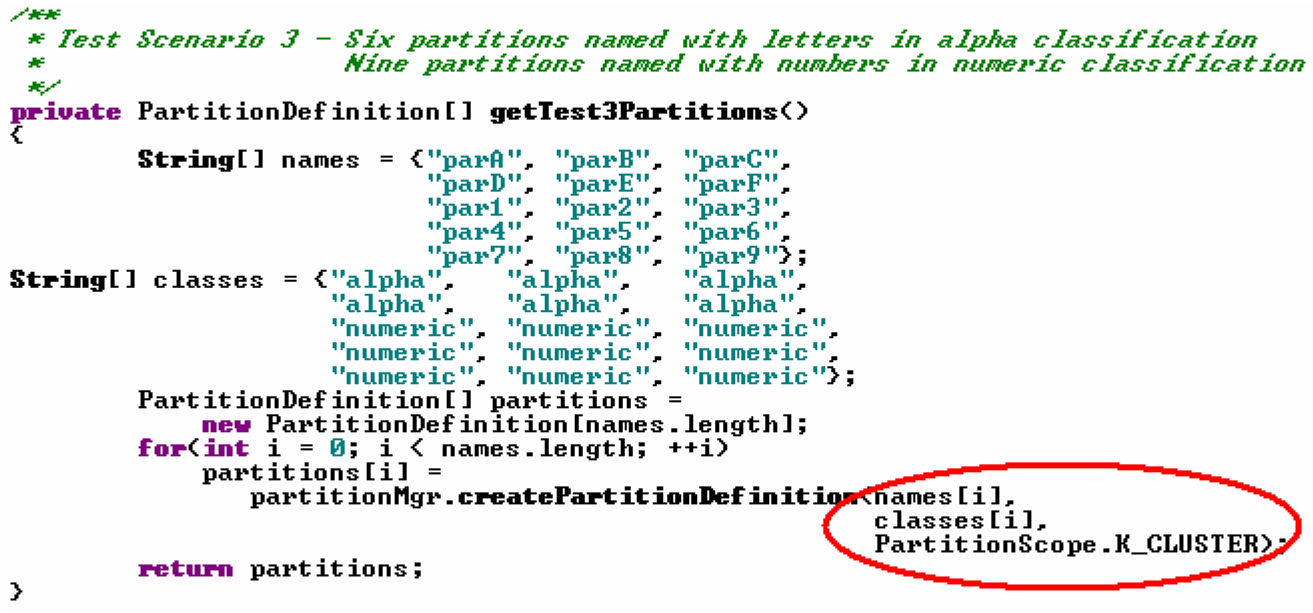
____ **1.** First you will look at the session bean to see how the code defines partitions with associated classifications.

__ a. On **hostA**, open any editor, such as Notepad.

__ b. In the editor, open the file **<LAB_FILES>\<LAB_NAME>\LetterPartitionsBean.java**.

__ c. Find the method named **getTest3Partitions()**.

__ d. Examine the code. Notice that the **createPartitionDefinition()** method takes three parameters rather than one as it did in the previous test scenarios. Associated with each partition name there is a partition classification which will be used to associate that partition with a specific classification of policy. (The partition scope parameter is not a factor in this exercise). Notice that the partitions parA through parF will be associated with the alpha classification and the partitions par1 through par9 will be associated with the numeric classification.



```
/**
 * Test Scenario 3 - Six partitions named with letters in alpha classification
 *                   Nine partitions named with numbers in numeric classification
 */
private PartitionDefinition[] getTest3Partitions()
{
        String[] names = {"parA", "parB", "parC",
                          "parD", "parE", "parF",
                          "par1", "par2", "par3",
                          "par4", "par5", "par6",
                          "par7", "par8", "par9"};
String[] classes = {"alpha",    "alpha",    "alpha",
                    "alpha",    "alpha",    "alpha",
                    "numeric", "numeric", "numeric",
                    "numeric", "numeric", "numeric",
                    "numeric", "numeric", "numeric"};
        PartitionDefinition[] partitions =
            new PartitionDefinition[names.length];
        for(int i = 0; i < names.length; ++i)
            partitions[i] =
               partitionMgr.createPartitionDefinition(names[i],
                                                      classes[i],
                                                      PartitionScope.K_CLUSTER);

        return partitions;
}
```

____ 2. Examine the properties file used to define the policy for the alpha partitions.

__ a. On hostA, open any editor, such as Notepad.

__ b. In the editor, open the file **<LAB_FILES>\<LAB_NAME>\AlphaClass.properties**.

__ c. Examine the properties file for these matching criteria.

__ d. **NumOfMatchCriteria = 4** -- notice that it is 4 rather than 3 as in the previous policy example because you need to specify a partition classification value.

__ e. **Name_3 = -pc** and **Value_3 = alpha -** the match criteria which indicates that this is used with the partition classification named alpha

__ f. **PreferredOnly = true** – indicates that partitions matching this policy should only be allowed to run on the preferred servers.

__ g. **NumOfPolicyServers = 2** – indicates that there are two servers to be used with this policy

__ h. **NodeName_0 = hostBnode01 and ServerName_0 = c1server1** – indicates that c1server1 on hostBnode01 is a preferred server.

__ i. **NodeName_1 = hostCnode01** and **ServerName_1 = c1server3** – indicates that c1server3 on hostCnode01 is a preferred server.

__ j. **Failback = true** – this indicates that the first server in the list of preferred servers should be used whenever possible, with partitions automatically moving to this server if it is running.

____ 3. Examine the properties file used to define the policy for the numeric partitions.

__ a. On **hostA**, open any editor, such as Notepad.

__ b. In the editor, open the file **<LAB_FILES>\<LAB_NAME>\NumericClass.properties**.

__ c. Compare this properties file with the properties file for the alpha policy. Notice that the only difference between the two files is the **value for –pc** (partition classification) matching criteria and the designation of the **first server in the primary server list** (excluding the PolicyName and PolicyDescription differences which do not affect the actual policy being defined).

____ **4. Update Configuration Policies = AlphaClass.properties** (see **Common Task 9: Update configuration policies = <property file name> on page 39**).

____ **5. Update Configuration Policies = NumericClass.properties** (see **Common Task 9: Update configuration policies = <property file name> on page 39**).

____ **6. Update Scenario Property = test3** (see **Common Task 8: Update scenario property = <testn> on page 39**).

____ **7. SAVE the Configuration**.

____ **8. Start the Cluster**.

____ **9.** Examine the state of the partitions with all three servers of the cluster running to see the effect of the defined policies. You should expect that all six of the alpha partitions are running on c1server1, all nine of the numeric partitions are running on c1server2 and that there are no partitions running c1server3.

__ a. Run **wpfadmin countActivePartitionsOnServers**. The output should look something like what is shown below.

```
WPFC0051I: Server hostaCell01\hostCnode01 \c1server3: 0
WPFC0051I: Server hostaCell01\hostBnode01 \c1server2: 9
WPFC0051I: Server hostaCell01\hostBnode01 \c1server1: 6
Total number of partitions is 15
```

__ b. Run **wpfadmin listActive**. The output should look something like what is shown below. Note that the alpha partitions (parA – parF) are located on c1server1 and the numeric partitions (par1 – par9) are running on c1server2.

```
WPFC0050I: Application WPFSample1, Partition par9: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par8: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par7: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par6: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par5: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par4: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par3: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par2: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition par1: Server
hostaCell01\hostBnode01 \c1server2
WPFC0050I: Application WPFSample1, Partition parF: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition parE: Server
hostaCell01\hostBnode01 \c1server1
```

```
WPFC0050I: Application WPFSample1, Partition parD: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition parC: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition parB: Server
hostaCell01\hostBnode01 \c1server1
WPFC0050I: Application WPFSample1, Partition parA: Server
hostaCell01\hostBnode01 \c1server1
```

____ **10.** **Stop Server = c1server2**.

____ **11.** Using the wpfadmin commands, examine the output to verify that the numeric partitions that had been running on c1server2 have now failed over to c1server3.

__ a. **Run wpfadmin countActivePartitionsOnServers**.

__ b. **Run wpfadmin listActive**.

____ **12.** **Start Server = c1server2**.

____ **13.** Using the wpfadmin commands, examine the output to verify that the numeric partitions have now returned to c1server2, and that c1server3 again has no partitions running on it.

__ a. Run **wpfadmin countActivePartitionsOnServers**.

__ b. Run **wpfadmin listActive**.

____ **14.** **Stop the Cluster**.

____ **15.** Using the wpfadmin commands, verify that none of the partitions are active

__ a. Run **wpfadmin countActivePartitionsOnServers**.

____ **16.** **Start Server = c1server3**.

____ **17.** Using the wpfadmin commands, verify that all 15 partitions have been activated on c1server3.

__ a. Run **wpfadmin countActivePartitionsOnServers**.

____ **18.** **Start Server = c1server2**.

____ **19.** **Start Server = c1server1**.

____ **20.** Using the wpfadmin commands, verify that the 6 alpha partitions have been moved to c1server1 and that the nine numeric partitions have been moved to c1server2.

__ a. Run **wpfadmin countActivePartitionsOnServers**.

__ b. Run **wpfadmin listActive**.

____ **21.** **Stop the Cluster**.

Part 6: Calling methods on the partitioned application

Up to this point you have looked at the basic structure of the Stateless Session EJB needed to enable a partitioned application. You have explored the various behaviors the system provides for allocating partitions to servers in a cluster. Now you are ready to look at how business logic is provided by a partitioned application and how that logic is called from client programs. There are actually three different models for client interaction with a partitioned application, using EJB method calls (IIOP protocol), using a Web application (HTTP protocol) and using messaging (JMS protocol). In this lab, you will only address using EJB method calls. Here is a basic outline of the steps needed:

- Add business logic methods to the Stateless Session EJB.
- Code a class used by the client side stub to determine which partition a request should be routed to.
- Generate and deploy code in an EAR file.
- Modify the EJB stubs in the EAR file to provide partition based routing.
- Install the application in a cluster.
- Run a client that uses the normal EJB programming model to make calls on the partitioned EJB application.

To illustrate these steps, you will continue using the partitioning scheme and policies from Part 6. In Part 6 you had partitions that were divided into classifications. The numeric classification (**par1** to **par9**) and the alpha classification (**parA** to **parF**) each had different behaviors defined regarding which servers they would run on. Your business logic will consist of methods that are passed a partition name (for example, "**par6**") or suffix (for example, "**6**") and return a string indicating if that partition is active on the server which received the method call. The following steps will lead you through the steps required to understand, build and run this application.

____ 1. The previous parts examined the **getPartitions()** method of the Stateless Session EJB but did not look at the other three required methods the bean must implement to participate as a partitioning application, specifically **partitionLoadEvent()**, **partitionUnloadEvent()** and **isPartitionAlive()**. These allow a partitioned application running in a specific server to know which partitions of the application are currently active on that server. It also allows the application to do any initialization or termination activities related to the partition. Look at the implementation of these methods in the sample application, **<LAB_FILES>\<LAB_NAME>\LetterPartitionsBean.java**..

__ a. A set is used to keep track of which partitions are active in the server. The **activePartitions** set is defined as follows:

```
private Set activePartitions = Collections.synchronizedSet(new HashSet());
```

__ b. The **partitionLoadEvent()** writes a message to indicate that the partition is being loaded and adds the name of the partition to the **activePartitions** set.

```
public boolean partitionLoadEvent(String partitionName)
{
    System.out.println(
         "*********** partitionLoadEvent entered, partitionName="
          + partitionName);
    activePartitions.add(partitionName);
    return true;
}
```

__ c. Similarly, the **partitionUnloadEvent()** writes a message to indicate that the partition is being unloaded and removes the name of the partition from the **activePartitions** set.

```
public void partitionUnloadEvent(String partitionName)
{
    System.out.println(
         "*********** partitionUnloadEvent entered, partitionName="
         + partitionName);
    activePartitions.remove(partitionName);
}
```

__ d. The **isPartitionAlive()** returns a boolean to indicate if the partition is currently loaded in this server.

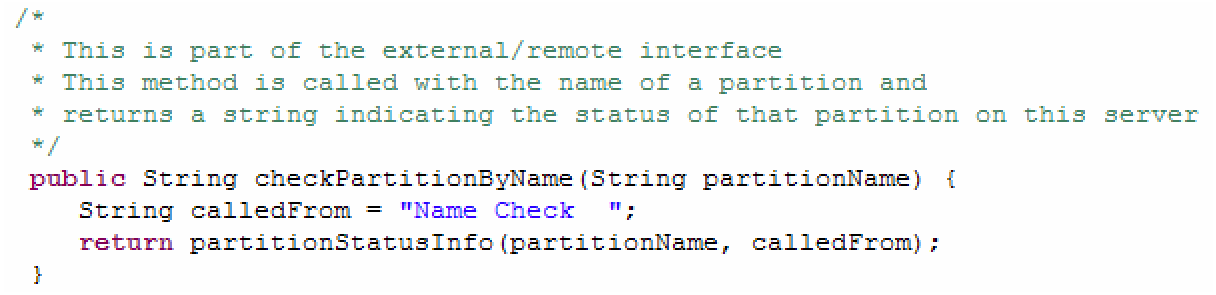
```
public boolean isPartitionAlive(String partitionName)
{
    return(activePartitions.contains(partitionName));
}
```

____ 2. The business logic for the sample application consists of two methods which are part of the remote interface of the EJB and a private method that does the real work for these methods. The parameter to the remote methods identifies a particular partition. The logic checks to make sure the specified partition is active on the server. A message indicating the status of the requested partition and the name of the server which handled the request is written to stdout and a returned to the caller as a string. Note that if the partitioning infrastructure is working correctly, a request should only reach a server where the requested partition is active. Therefore, if the partition is not active it indicates the existence of some problem in the setup or the partitioning infrastructure.

__ a. The private method which does the real work checks to see if the requested partition is active on this server by calling the **isPartitionAlive()** method and then writes an appropriate message and returns an appropriate status string.

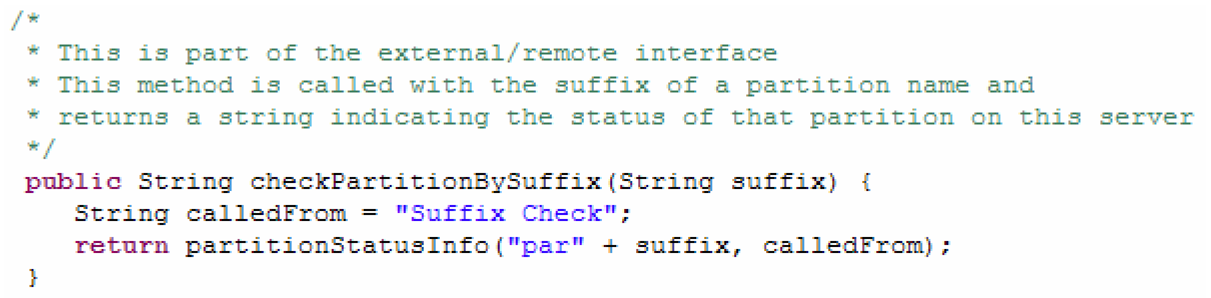
```
/*
 * Called with a partition name and responds indicating if that partition
 * is active on this server, and also provides the name of the server.
 */
private String partitionStatusInfo(String partitionName, String calledBy)
{
    if (isPartitionAlive(partitionName)) {
        System.out.println("*********** " + calledBy + ": Name="
                           + partitionName + " is Active");
        return(calledBy + ": Name=" + partitionName
              + ",is active on Server=" + servername);
    } else {
        System.out.println("*********** " + calledBy + ": Name="
                           + partitionName + " called but NOT Active <<--- error");
        return(calledBy + ": Name=" + partitionName + ",is NOT active on Server="
              + servername + ",,--- error");
    }
}
```

__ b. The first method which is part of the external interface is the **checkPartitionByName()** method. It is passed the full name of a partition (for example, “**parB**”) and it calls the private method which builds the appropriate status string for it to return.



```
/*
 * This is part of the external/remote interface
 * This method is called with the name of a partition and
 * returns a string indicating the status of that partition on this server
 */
public String checkPartitionByName(String partitionName) {
    String calledFrom = "Name Check  ";
    return partitionStatusInfo(partitionName, calledFrom);
}
```

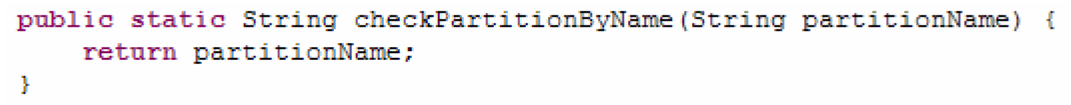
__ c. The second method which is part of the external interface is the **checkPartitionBySuffix()** method. It is passed the suffix of the partition name (for example, “**B**”) and it calls the private method which builds the appropriate status string for it to return.



```
/*
 * This is part of the external/remote interface
 * This method is called with the suffix of a partition name and
 * returns a string indicating the status of that partition on this server
 */
public String checkPartitionBySuffix(String suffix) {
    String calledFrom = "Suffix Check";
    return partitionStatusInfo("par" + suffix, calledFrom);
}
```

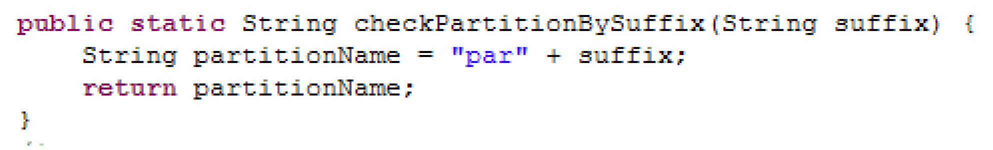
____ 3. At runtime, the decision of which server to route a request to is made on the client side. Application specific code must be written that can examine the parameters of a method request and make a determination as to which partition the request should be routed to. This application specific code is then used by the client side EJB stub when determining how to route the request. The naming convention for the class that must be written is **<stateless-session-ejb-name>_PartitionKey**. This class must have methods that match the methods of the remote EJB interface (method name and parameter types must match) and return the name of the partition the request should be routed to. The sample application has two external methods on the remote interface, and therefore the **<stateless-session-ejb-name>_PartitionKey** class must have code for each of these methods.

__ a. The **checkPartitionByName()** method takes the name of the partition as a parameter, and therefore it can be returned as is.



```
public static String checkPartitionByName(String partitionName) {
    return partitionName;
}
```

__ b. The **checkPartitionBySuffix()** method must construct the name of the partition as the parameter to it only contains the suffix of the partition name.



```
public static String checkPartitionBySuffix(String suffix) {
    String partitionName = "par" + suffix;
    return partitionName;
}
```

____ 4. The next step is to build an EAR file where the EJB stub has been modified so that the **<stateless-session-ejb-name>_PartitionKey** class participates in the client side routing of requests. The **wpfStubUtil** command line tool is provided to do this.

__ a. Normally the first step of this process would be to use Rational Application Developer to create an EAR file for the application that already contains the deployment code. For the purposes of this lab, these steps have already been done for you. The wpfsample1.ear file in the <LAB_FILES>\<LAB_NAME> directory contains the results of deploying and exporting the EAR file from within Application Developer.

NOTE: The Partition Facility wpfStubUtil command annotates existing deployed code. Therefore when preparing an EAR file for partitioning support, you cannot export an EAR without deployed EJBs and then select "deploy EJBs" during application installation. The ear file in <LAB_FILES>\<LAB_NAME>\wpfsample1.ear was created with deployed EBJs.

__ b. To generate the stubs, use the **wpfStubUtil** command. The command requires parameters specifying the EAR file, the name of the EJB jar file, the path to the Stateless Session EJB class and a temporary directory to use.

1) On **hostA**, open a Command Prompt window.

2) Change directories to **C:\WebSphere\AppServer\bin**.

3) Enter the command (all on one line) :

Windows version:

```
wpfStubUtil -ear <LAB_FILES>\<LAB_NAME>\wpfsample1.ear
      -jar WPFSampleEJB1.jar
      -class wpf/sample/simple/ejb/LetterPartitions.class
      -temp <LAB_FILES>\<LAB_NAME>\temp
```

UNIX version:

```
wpfStubUtil -ear <LAB_FILES>/<LAB_NAME>/wpfsample1.ear
      -jar WPFSampleEJB1.jar
      -class wpf/sample/simple/ejb/LetterPartitions.class
      -temp <LAB_FILES>/<LAB_NAME>/temp
```

4) Verify that the update was successful by finding these lines in your output.

```
WPFC0072I: Updating stub source
WPFC0073I: Compiling modified stub source
WPFC0074I: Rejaring jar file WPFSampleEJB1.jar
WPFC0075I: Rejaring ear file <LAB_FILES>/<LAB_NAME>/wpfsample1.ear
Cleaning up
```

Note: If you receive unexpected errors (for example, as a result of incorrect command operands) delete the contents of the <LAB_FILES>/<LAB_NAME>/temp directory before submitting the command again.

____ 5. Install the application in the cluster. Since the previous version of the EAR is currently installed, it will need to be uninstalled first.

__ a. In the Navigation panel, expand **Applications** and select **Enterprise Applications.**

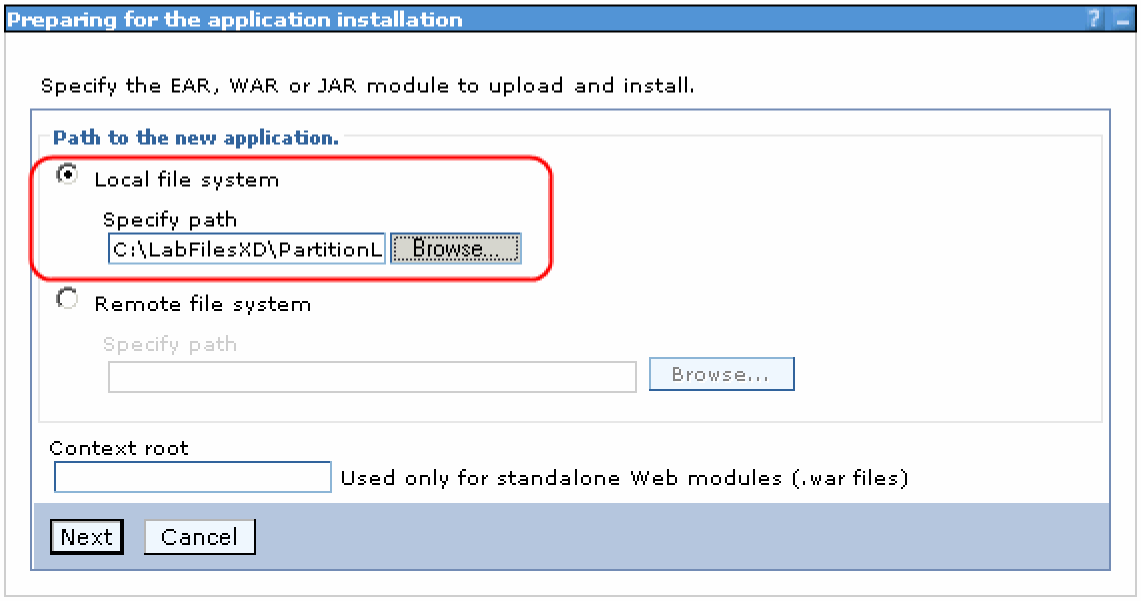
__ b. Select the check box in front of **WPFSample1**.

__ c. Click the **Uninstall** button.

__ d. **SAVE the Configuration**.

__ e. In the Navigation panel, expand **Applications** and select **Install New Application.**

__ f. Click the **Browse** button for Local file system, and use the browse dialog to select **<LAB_FILES>\<LAB_NAME>\wpfsample1.ear**.



Specify the EAR, WAR or JAR module to upload and install.

Path to the new application.

(●) Local file system

Specify path: C:\LabFilesXD\PartitionL

( ) Remote file system

Specify path

Context root: Used only for standalone Web modules (.war files)

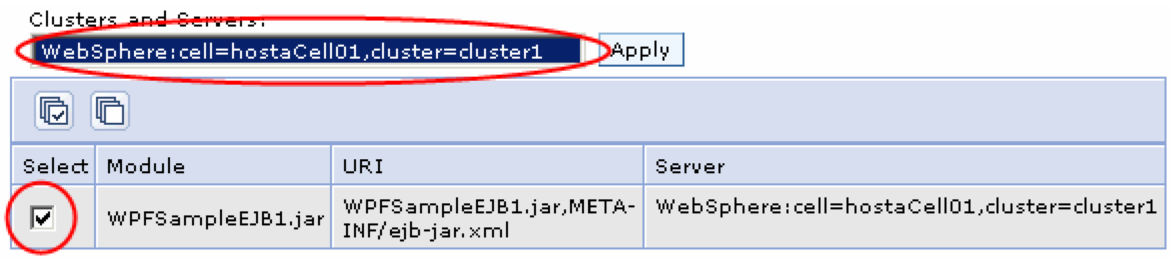
__ g. Click **Next**.

__ h. Click **Next**.

__ i. Click **Step 2 Map modules to servers.**

__ j. In the **Clusters and Servers** list, select **cluster1** (see screen capture below).

__ k. In the **Module** list check **WPFSampleEJB1.jar.**

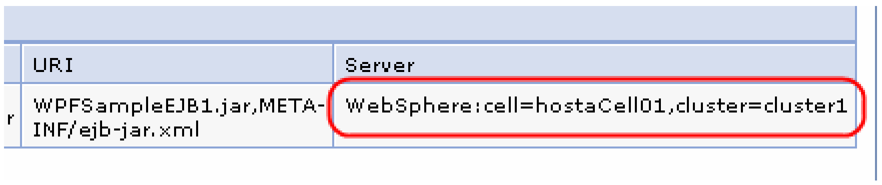

Clusters and Servers:

WebSphere:cell=hostaCell01,cluster=cluster1

Select	Module	URI	Server
✓	WPFSampleEJB1.jar	WPFSampleEJB1.jar,META-INF/ejb-jar.xml	WebSphere:cell=hostaCell01,cluster=cluster1

__ l. Click **Apply**.

__ m. Ensure that the **Server** has been changed to **cluster1** for **WPFSampleEJB1.jar** (circled in the screen capture below).



URI	Server
WPFSampleEJB1.jar,META-INF/ejb-jar.xml	WebSphere:cell=hostaCell01,cluster=cluster1

__ n. Click **Step 3 Summary**.

__ o. Click **Finish** and wait for the installation to complete.

__ p. **SAVE the Configuration**

____ 6. **Start the Cluster**.

____ **7.** Using the wpfadmin commands, verify that the 6 alpha partitions have been started on c1server1 and that the nine numeric partitions have been started on c1server2.

__ a. Run **wpfadmin countActivePartitionsOnServers**.

__ b. Run **wpfadmin listActive**.

____ 8. Start the client application and learn about how the client side program works.

__ a. On **hostA**, open a Command Prompt window.

__ b. Change directories to **C:\WebSphere\AppServer\bin**.

__ c. Enter the command (all on one line):

```
launchClient <LAB_FILES>/<LAB_NAME>/wpfsample1.ear
      -CCBootstrapHost=hostA –CCBootstrapPort=9809
```

where 9809 is the bootstrap port for hostB

Note: the bootstrap port can be determined by examining the ports assignment for the hostB nodeagent.

- click on **System administration ➔ Node agents**
- click on nodeagent for hostBNode01
- expand Ports and take note of the port assignment for **BOOTSTRAP_ADDRESS**

__ d. Examine the help that is displayed from the client application. The application has two modes of operation, and the input is interpreted based on current mode. In name mode, you can enter the name of a partition and a method call will be made to that partition. In suffix mode, you can enter any number of partition suffixes (1-9, A-F) and a method call will be made to every partition whose suffix is specified. In addition, there are keywords that can be entered to change modes, display the help panel again or exit the program.

____ 9. Run the client program a couple of times to see the kind of output is produced by the client.

__ a. The program comes up in “suffix” mode. Enter the string **123456789ABCDEF** to invoke a method call on every partition. This may take quite a long time to complete, probably between 1-2 minutes.

Note: The first call to a partition from a client side JVM takes a long time to resolve. This should improve in future fixpacks, and is not a problem when making calls between application servers. There are also design patterns, beyond the scope of this lab, which will bypass this issue.

__ b. Examine the output to ensure all the numeric partitions were routed to c1server2 and that all the alpha partitions were routed to c1server1. You output should look something like this:

```
Current Mode: suffix
Enter ===> 123456789ABCDEF
Suffix Check: Name=par1,is active on Server=c1server2, elapsed
milliseconds=2000
Suffix Check: Name=par2,is active on Server=c1server2, elapsed
milliseconds=1594
Suffix Check: Name=par3,is active on Server=c1server2, elapsed
milliseconds=1562
```

```
Suffix Check: Name=par4,is active on Server=c1server2, elapsed
milliseconds=2750
Suffix Check: Name=par5,is active on Server=c1server2, elapsed
milliseconds=2078
Suffix Check: Name=par6,is active on Server=c1server2, elapsed
milliseconds=2282
Suffix Check: Name=par7,is active on Server=c1server2, elapsed
milliseconds=2171
Suffix Check: Name=par8,is active on Server=c1server2, elapsed
milliseconds=2047
Suffix Check: Name=par9,is active on Server=c1server2, elapsed
milliseconds=1547
Suffix Check: Name=parA,is active on Server=c1server1, elapsed
milliseconds=1594
Suffix Check: Name=parB,is active on Server=c1server1, elapsed
milliseconds=2062
Suffix Check: Name=parC,is active on Server=c1server1, elapsed
milliseconds=1610
Suffix Check: Name=parD,is active on Server=c1server1, elapsed
milliseconds=1547
Suffix Check: Name=parE,is active on Server=c1server1, elapsed
milliseconds=2828
Suffix Check: Name=parF,is active on Server=c1server1, elapsed
milliseconds=2063
```

__ c. Enter the string **123456789ABCDEF** to invoke a method call on each partition a second time. The performance should be much improved and your output should look something like this:

```
Current Mode: suffix
Enter ===> 123456789ABCDEF
Suffix Check: Name=par1,is active on Server=c1server2, elapsed
milliseconds=31
Suffix Check: Name=par2,is active on Server=c1server2, elapsed
milliseconds=16
Suffix Check: Name=par3,is active on Server=c1server2, elapsed
milliseconds=15
Suffix Check: Name=par4,is active on Server=c1server2, elapsed
milliseconds=47
Suffix Check: Name=par5,is active on Server=c1server2, elapsed
milliseconds=0
Suffix Check: Name=par6,is active on Server=c1server2, elapsed
milliseconds=15
Suffix Check: Name=par7,is active on Server=c1server2, elapsed
milliseconds=16
Suffix Check: Name=par8,is active on Server=c1server2, elapsed
milliseconds=16
Suffix Check: Name=par9,is active on Server=c1server2, elapsed
milliseconds=15
Suffix Check: Name=parA,is active on Server=c1server1, elapsed
milliseconds=16
Suffix Check: Name=parB,is active on Server=c1server1, elapsed
milliseconds=16
Suffix Check: Name=parC,is active on Server=c1server1, elapsed
milliseconds=62
Suffix Check: Name=parD,is active on Server=c1server1, elapsed
milliseconds=16
Suffix Check: Name=parE,is active on Server=c1server1, elapsed
milliseconds=0
Suffix Check: Name=parF,is active on Server=c1server1, elapsed
milliseconds=31
```

Partition Facility

____ 10. Examine the SystemOut.log file for c1server1 to see the type of information the Stateless Session EJB puts into the log.

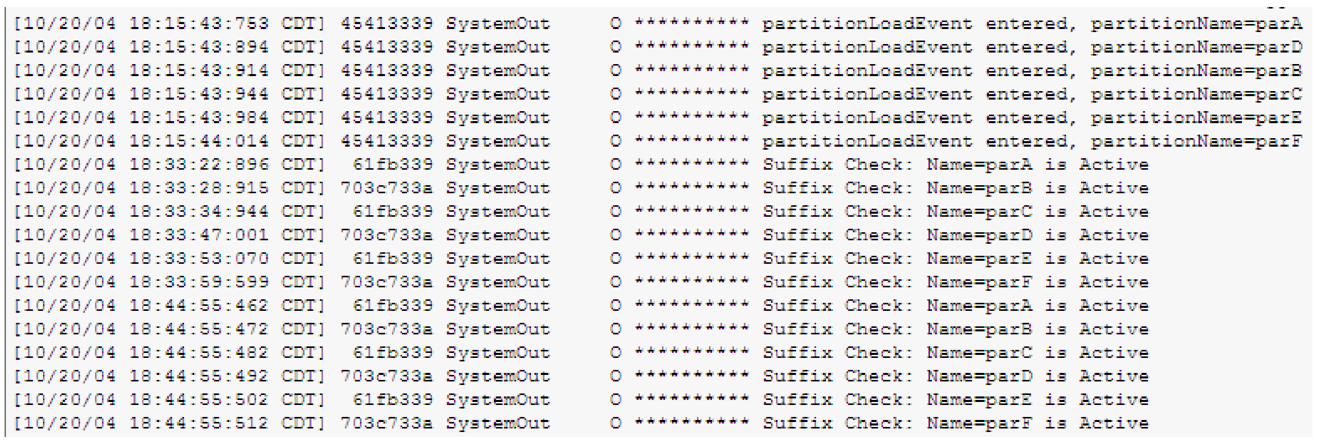
__ a. In the Navigation panel, expand **Troubleshooting** and select **Logs and Trace.**

__ b. In the right panel, select **c1server1 → JVM Logs** and then click on the **Runtime** tab.

__ c. Press the **View** button for the **SystemOut.log** file.

__ d. The panel displaying the log file will tell you the total number of lines in the file. In the **Retrieve Lines** entry field, enter the last 20 lines (for example, if 678 total lines, enter 658-678) and click the **Refresh** button.

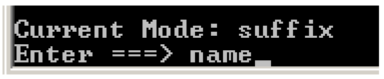
__ e. You should see these messages in your SystemOut.log file. They are the messages from the partition load events for each of the alpha partitions and the messages from the method calls to the alpha partitions. If you were to look in the log for c1server2, you would see similar entries for the numeric partitions.



```
[10/20/04 18:15:43:753 CDT] 45413339 SystemOut     O ********** partitionLoadEvent entered, partitionName=parA
[10/20/04 18:15:43:894 CDT] 45413339 SystemOut     O ********** partitionLoadEvent entered, partitionName=parD
[10/20/04 18:15:43:914 CDT] 45413339 SystemOut     O ********** partitionLoadEvent entered, partitionName=parB
[10/20/04 18:15:43:944 CDT] 45413339 SystemOut     O ********** partitionLoadEvent entered, partitionName=parC
[10/20/04 18:15:43:984 CDT] 45413339 SystemOut     O ********** partitionLoadEvent entered, partitionName=parE
[10/20/04 18:15:44:014 CDT] 45413339 SystemOut     O ********** partitionLoadEvent entered, partitionName=parF
[10/20/04 18:33:22:896 CDT]  61fb339 SystemOut     O ********** Suffix Check: Name=parA is Active
[10/20/04 18:33:28:915 CDT] 703c733a SystemOut     O ********** Suffix Check: Name=parB is Active
[10/20/04 18:33:34:944 CDT]  61fb339 SystemOut     O ********** Suffix Check: Name=parC is Active
[10/20/04 18:33:47:001 CDT] 703c733a SystemOut     O ********** Suffix Check: Name=parD is Active
[10/20/04 18:33:53:070 CDT]  61fb339 SystemOut     O ********** Suffix Check: Name=parE is Active
[10/20/04 18:33:59:599 CDT] 703c733a SystemOut     O ********** Suffix Check: Name=parF is Active
[10/20/04 18:44:55:462 CDT]  61fb339 SystemOut     O ********** Suffix Check: Name=parA is Active
[10/20/04 18:44:55:472 CDT] 703c733a SystemOut     O ********** Suffix Check: Name=parB is Active
[10/20/04 18:44:55:482 CDT]  61fb339 SystemOut     O ********** Suffix Check: Name=parC is Active
[10/20/04 18:44:55:492 CDT] 703c733a SystemOut     O ********** Suffix Check: Name=parD is Active
[10/20/04 18:44:55:502 CDT]  61fb339 SystemOut     O ********** Suffix Check: Name=parE is Active
[10/20/04 18:44:55:512 CDT] 703c733a SystemOut     O ********** Suffix Check: Name=parF is Active
```

____ 11. Change the client program to "name" mode and invoke a request to familiarize yourself with this mode of operation.

__ a. At the client program's **Enter** prompt, enter **name**.



```
Current Mode: suffix
Enter ===> name
```

__ b. Enter **par3** and examine the output.

```
Current Mode: name
Enter ===> par3
Name Check  : Name=par3,is active on Server=c1server2, elapsed
milliseconds=15
```

__ c. Enter **parA** and examine the output.

```
Current Mode: name
Enter ===> parA
Name Check  : Name=parA,is active on Server=c1server1, elapsed
milliseconds=15
```

____ 12. See how the client program reacts when a partition is moved to another server.

__ a. **Stop Server = c1server1**.

__ b. At the client program's **Enter** prompt, enter **parA**. Notice that it again takes a long time to resolve, but that the output indicates that the request was handled by c1server3.

```
Current Mode: name
Enter ===> parA
```

```
Name Check  : Name=parA,is active on Server=c1server3, elapsed
milliseconds=2094
```

__ c. Again, enter **parA**. Notice that the method call returns to being handled quickly.

```
Current Mode: name
Enter ===> parA
Name Check  : Name=parA,is active on Server=c1server3, elapsed
milliseconds=31
```

____ 13. Now it is time to experiment on your own. Try different scenarios of starting and stopping servers and running requests from the client. Examine the client side output and the SystemOut.log to gain a better understanding of the partitioning behavior. Type **quit** toexit the facility.

____ 14. You can now stop the cluster, uninstall the wpfsample1 application, and delete the cluster.

Appendix A. Common tasks used throughout the exercise

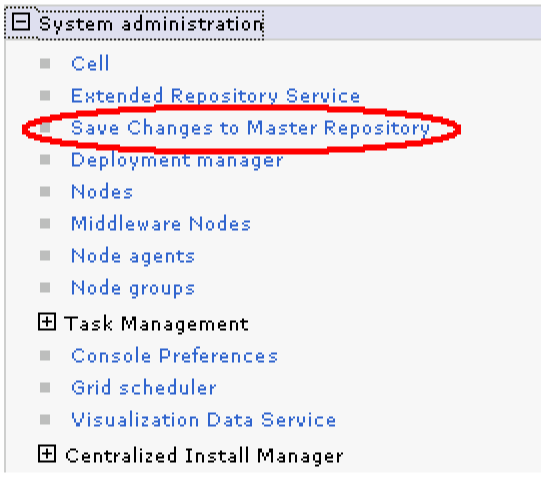
This part contains instructions for simple tasks that are performed multiple times during this lab exercise. Other parts of the exercise reference these tasks when they need to be performed.

Common Task 1: SAVE the configuration

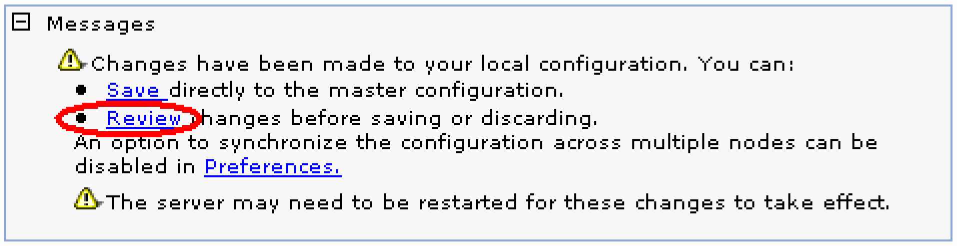
____ 15. The task of saving configuration changes to the Master Configuration is performed from the **Save to Master Configuration Panel.** This panel can be reached in a several different ways, depending upon what configuration activities you have been doing. These instructions will document the most common ways to access the panel.

__ a. Access the Save to Master Configuration Panel

1) Method 1 – In the Navigation panel, expand **System Administration** and select **Save Changes to Master Repository**



2) Method 2 – Click the **Review** link in Messages panel

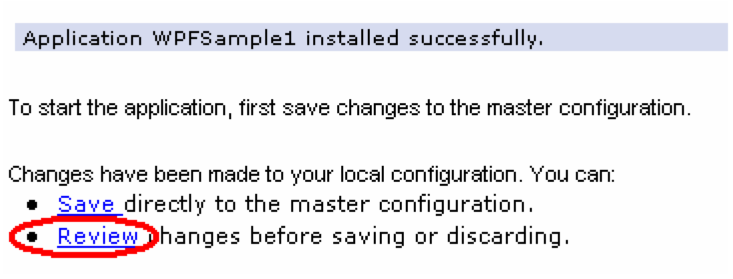

Messages

Changes have been made to your local configuration. You can:
- Save directly to the master configuration.
- Review changes before saving or discarding.

An option to synchronize the configuration across multiple nodes can be disabled in Preferences.

The server may need to be restarted for these changes to take effect.

3) Method 3 – Click **Review changes before saving or discarding** after installing an application.



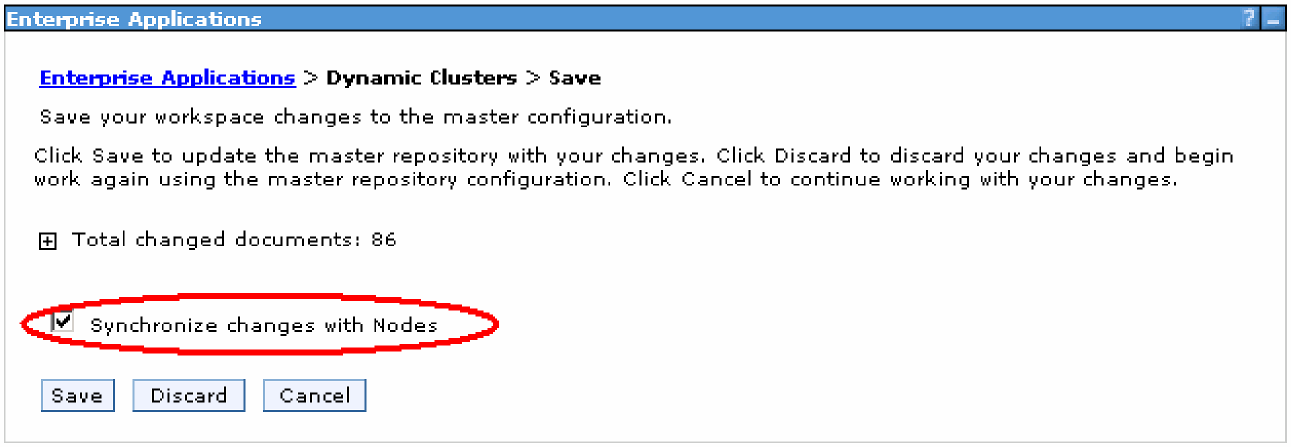
Application WPFSample1 installed successfully.

To start the application, first save changes to the master configuration.

Changes have been made to your local configuration. You can:

- Save directly to the master configuration.
- Review changes before saving or discarding.

__ b. For all three methods, ensure that the **Synchronize changes with Nodes** check box is checked.



Enterprise Applications > Dynamic Clusters > Save

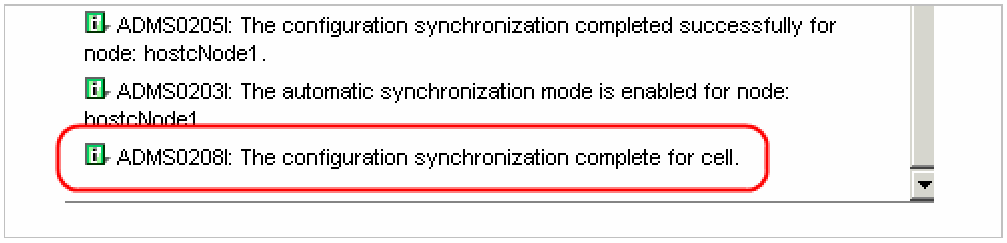
Save your workspace changes to the master configuration.

Click Save to update the master repository with your changes. Click Discard to discard your changes and begin work again using the master repository configuration. Click Cancel to continue working with your changes.

⊞ Total changed documents: 86

☑ Synchronize changes with Nodes

__ c. Click **Save** and wait for the message circled in the screen capture below appears in the node synchronization box.



ADMS0205I: The configuration synchronization completed successfully for node: hostcNode1 .

ADMS0203I: The automatic synchronization mode is enabled for node: hostcNode1

ADMS0208I: The configuration synchronization complete for cell.

Common Task 2: Start the cluster

____ 1. This task is used to start all the servers in the cluster at the same time.

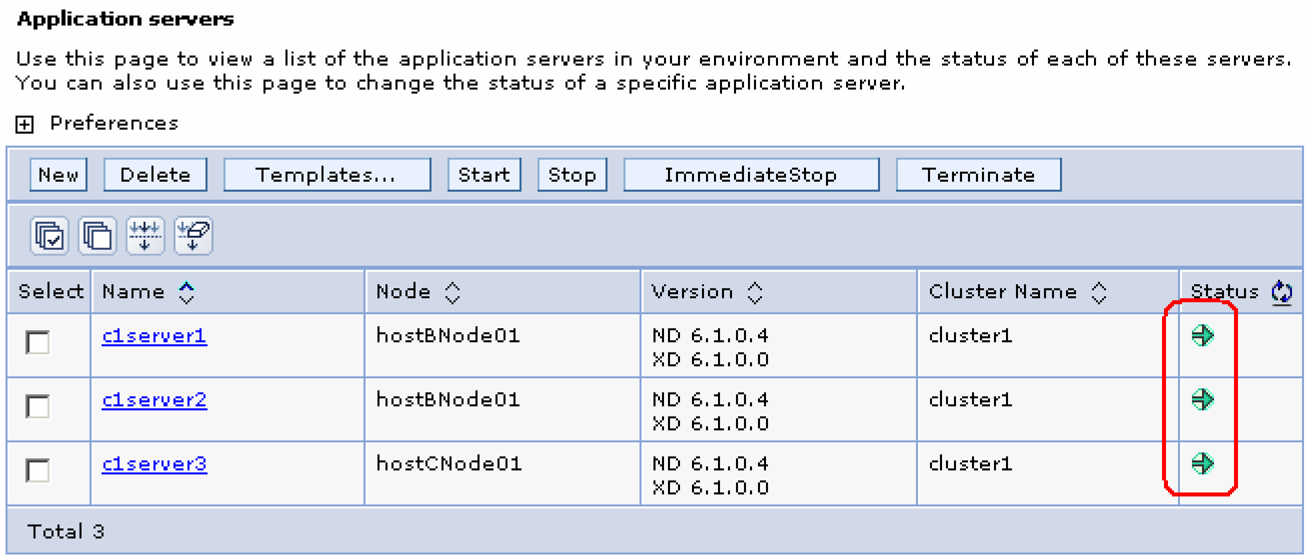
__ a. In Navigation panel, expand **Servers** and select **Clusters**.

__ b. Select the check box to the left of **cluster1** in the clusters list.

__ c. Click **Start**.

__ d. In the Navigation panel, select **Application servers**.

__ e. Wait until the three servers that are in the cluster show a status of Started (indicated by the green arrow). This may take about a minute. You may have to click the **Refresh View** icon ( ) if the status does not update automatically.


Application servers

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

⊞ Preferences

Select	Name	Node	Version	Cluster Name	Status
☐	c1server1	hostBNode01	ND 6.1.0.4 XD 6.1.0.0	cluster1	➡
☐	c1server2	hostBNode01	ND 6.1.0.4 XD 6.1.0.0	cluster1	➡
☐	c1server3	hostCNode01	ND 6.1.0.4 XD 6.1.0.0	cluster1	➡

Total 3

NOTE: If one or more of the servers in the cluster do not start, you will need to debug why this occurred. Checking the **SystemOut.log** file for the server is typically the best place to start. It can be found in the directory **c:\WebSphere\AppServer\profiles\<PROFILE_NAME>\logs\<SERVER_NAME>**.

Common Task 3: Stop the cluster

____ 1. This task is used to stop all the servers in the cluster at the same time.

__ a. In the Navigation panel, expand **Servers** and select **Clusters**.

__ b. Select the check box to the left of **cluster1** in the clusters list.

__ c. Click **Stop**.

__ d. In the Navigation panel, select **Application Servers**.

__ e. Wait until the three servers that are in the cluster show a status of Stopped (indicated by the red **X**). This may take about a minute. You may have to click the **Refresh View** icon ( ) if the status does not update automatically.

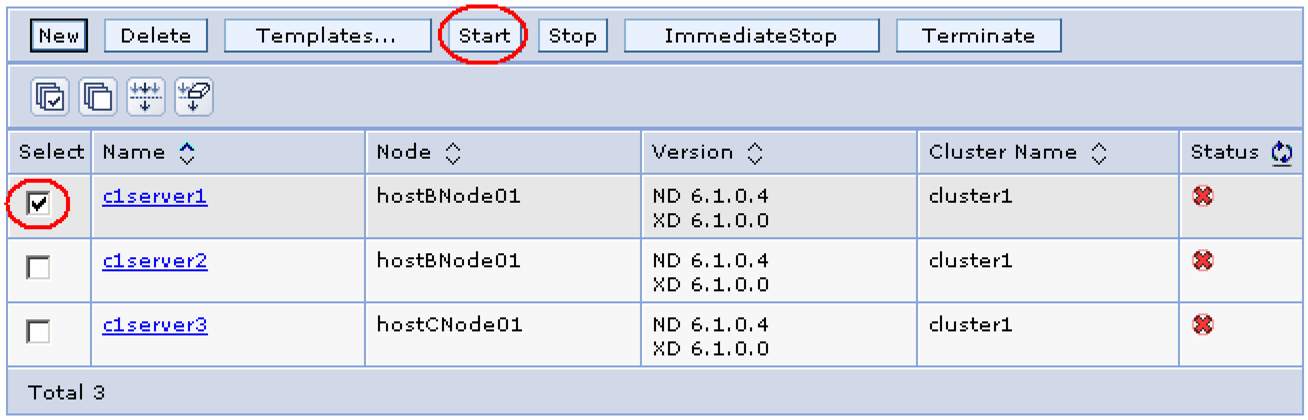
Common Task 4: Start server = <servername>

____ 1. This task is used to start a single server named <servername>.

__ a. In the Navigation panel, expand **Servers** and select **Application Servers.**

__ b. Select the check box to the left of **<servername>** in the servers list.

__ c. Click **Start.**



New | Delete | Templates... | Start | Stop | ImmediateStop | Terminate

Select	Name	Node	Version	Cluster Name	Status
☑	c1server1	hostBNode01	ND 6.1.0.4 XD 6.1.0.0	cluster1	✖
☐	c1server2	hostBNode01	ND 6.1.0.4 XD 6.1.0.0	cluster1	✖
☐	c1server3	hostCNode01	ND 6.1.0.4 XD 6.1.0.0	cluster1	✖
Total 3					

__ d. Wait until <servername> shows a status of Started (indicated by the green arrow). This may take about a minute. You may have to click the **Refresh View** icon ( ) if the status does not update automatically.

Common Task 5: Stop server = <servername>

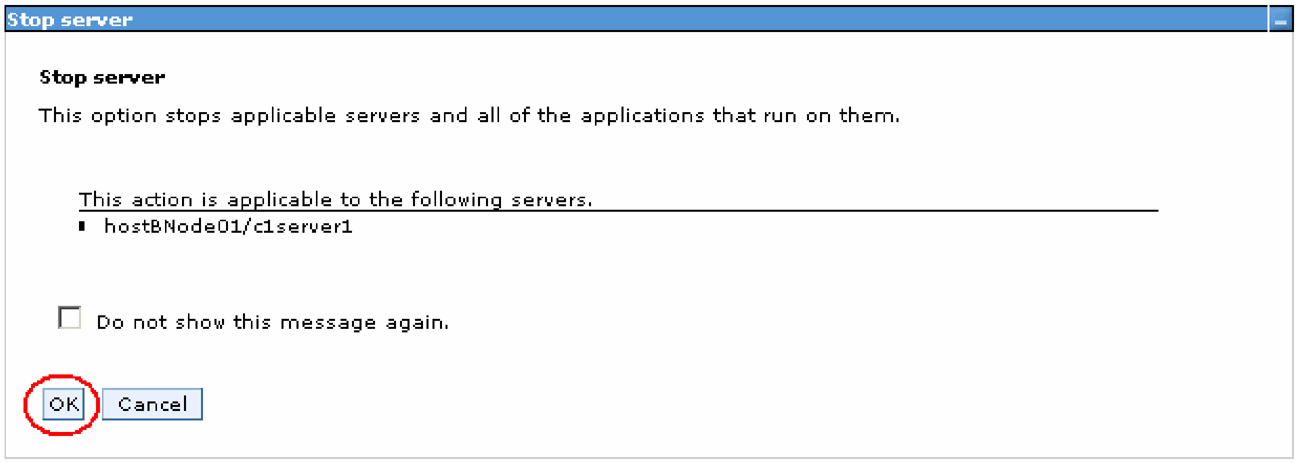
____ 1. This task is used to stop a single server named <servername>.

__ a. In the Navigation panel, expand **Servers** and select **Application Servers.**

__ b. Select the check box to the left of **<servername>** in the servers list.

__ c. Click **Stop.**

__ d. In the Stop server panel, read the message and click **OK**.



Stop server

Stop server

This option stops applicable servers and all of the applications that run on them.

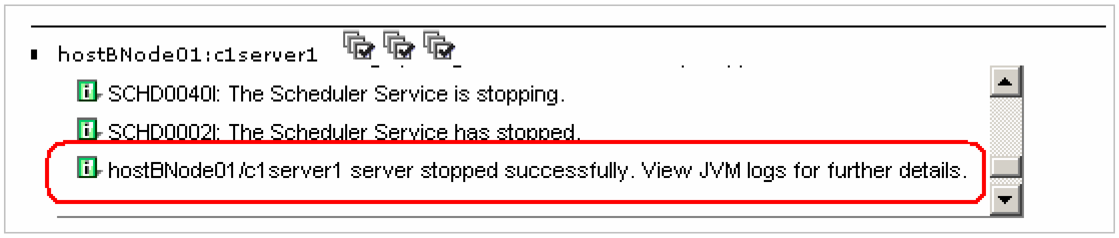
This action is applicable to the following servers.

- hostBNode01/c1server1

☐ Do not show this message again.

OK | Cancel

__ e. Wait for the message circled in the screen capture below to appear. This may take up to one to two minutes to appear.



```
hostBNode01:c1server1
SCHD0040I: The Scheduler Service is stopping.
SCHD0002I: The Scheduler Service has stopped.
hostBNode01/c1server1 server stopped successfully. View JVM logs for further details.
```

Common Task 6: Run wpfadmin countActivePartitionsOnServers

____ 1. This task is used to provide a count of the number of partitions that are active on each of the running servers containing a partitioned application.

__ a. On **hostA**, open a Command Prompt window.

__ b. Change directories to **C:\WebSphere\AppServer\profiles\dmgr\bin.**

__ c. Enter the command:

```
wpfadmin countActivePartitionsOnServers
```

__ d. Examine the output shown below from this command. Each of the running servers is listed along with a count of the active partitions. In this case, each of the three servers contains 9 active partitions.

```
CWPFC0051I: Server hostACell01\hostCNode01\c1server3: 0
CWPFC0051I: Server hostACell01\hostBNode01\c1server2: 5
CWPFC0051I: Server hostACell01\hostBNode01\c1server1: 4
CWPFC0081I: Total number of partitions is 9
```

Common Task 7: Run wpfadmin listActive

____ 1. This task is used to provide a count of the number of partitions that are active on each of the running servers containing a partitioned application.

__ a. On **hostA**, open a Command Prompt window.

__ b. Change directories to **C:\WebSphere\AppServer\profiles\dmgr\bin**.

__ c. Enter this command

```
wpfadmin listActive
```

__ d. Examine the output shown below from this command. There is line produced for each of the active partitions, providing the Application name, the Partition name and the name of the server on which it is running.

```
CWPFC0050I: Application WPFSample1, Partition par9: Server
hostACell01\hostBNode01\c1server1
CWPFC0050I: Application WPFSample1, Partition par8: Server
hostACell01\hostBNode01\c1server2
CWPFC0050I: Application WPFSample1, Partition par7: Server
hostACell01\hostBNode01\c1server1
CWPFC0050I: Application WPFSample1, Partition par6: Server
hostACell01\hostBNode01\c1server2
CWPFC0050I: Application WPFSample1, Partition par5: Server
hostACell01\hostBNode01\c1server1
```

```
CWPFC0050I: Application WPFSample1, Partition par4: Server
hostACell01\hostBNode01\c1server2
CWPFC0050I: Application WPFSample1, Partition par3: Server
hostACell01\hostBNode01\c1server2
CWPFC0050I: Application WPFSample1, Partition par2: Server
hostACell01\hostBNode01\c1server1
CWPFC0050I: Application WPFSample1, Partition par1: Server
hostACell01\hostBNode01\c1server2
```

Common Task 8: Update scenario property = <testn>

____ 1. This task is used to update the property used to control which scenario will be run by the partitioning application.

__ a. In the Navigation panel, expand **Environment > Naming** and select **Name space bindings.**

__ b. Click on **WPFSampleBinding**.

__ c. Change the String Value to **<testn>** (where <testn> is the string “test1”, “test2”, and so on)

__ d. Click **OK**.

Common Task 9: Update configuration policies = <property file name>

____ 1. This task is used to take a properties file and update the configuration files with the policy information defined by those properties.

__ a. On **hostA**, open a Command Prompt window.

__ b. Change directories to **C:\WebSphere\AppServer\profiles\dmgr\bin**.

__ c. At the command prompt, enter this command (ensure that the slashes are forward slashes and not backward slashes as wpfadmin will not work with backward slashes).

```
wpfadmin createPolicy <LAB_FILES>/<LAB_NAME>/<PROPERTY_FILE_NAME>
```

__ d. The expected output from the command above should look something like this:

```
The policy <POLICY_NAME> has been created
```

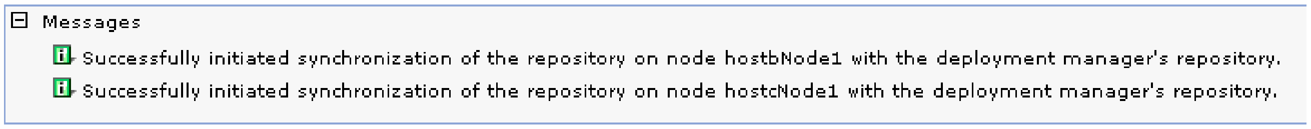
____ 2. Ensure that the updates are distributed to all the nodes by forcing a resynchronization. (This step is technically not needed, as the updated configuration files will be sent to the nodes at the time of the next scheduled file sync. However, this step ensures the files are synchronized before proceeding.)

__ a. In the Navigation panel, expand **System Administration** and click on **Nodes**.

__ b. Check the check box in front of **hostBnode01** and **hostCnode01.**

__ c. Click **Full Resynchronize**.

__ d. Wait for the messages at the top of the panel indicating “Successfully initiated synchronization…” (see the screen capture below) and for the Status column to show the synchronized icon (⇔).


Messages

Successfully initiated synchronization of the repository on node hostbNode1 with the deployment manager's repository.

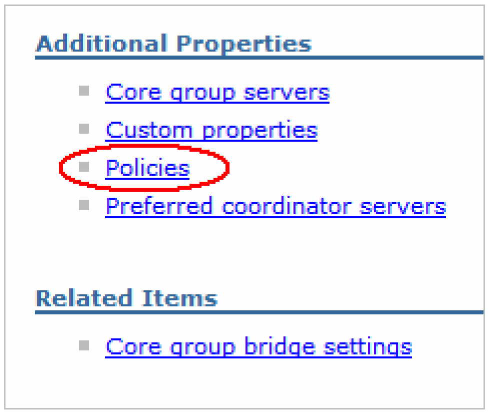
Successfully initiated synchronization of the repository on node hostcNode1 with the deployment manager's repository.

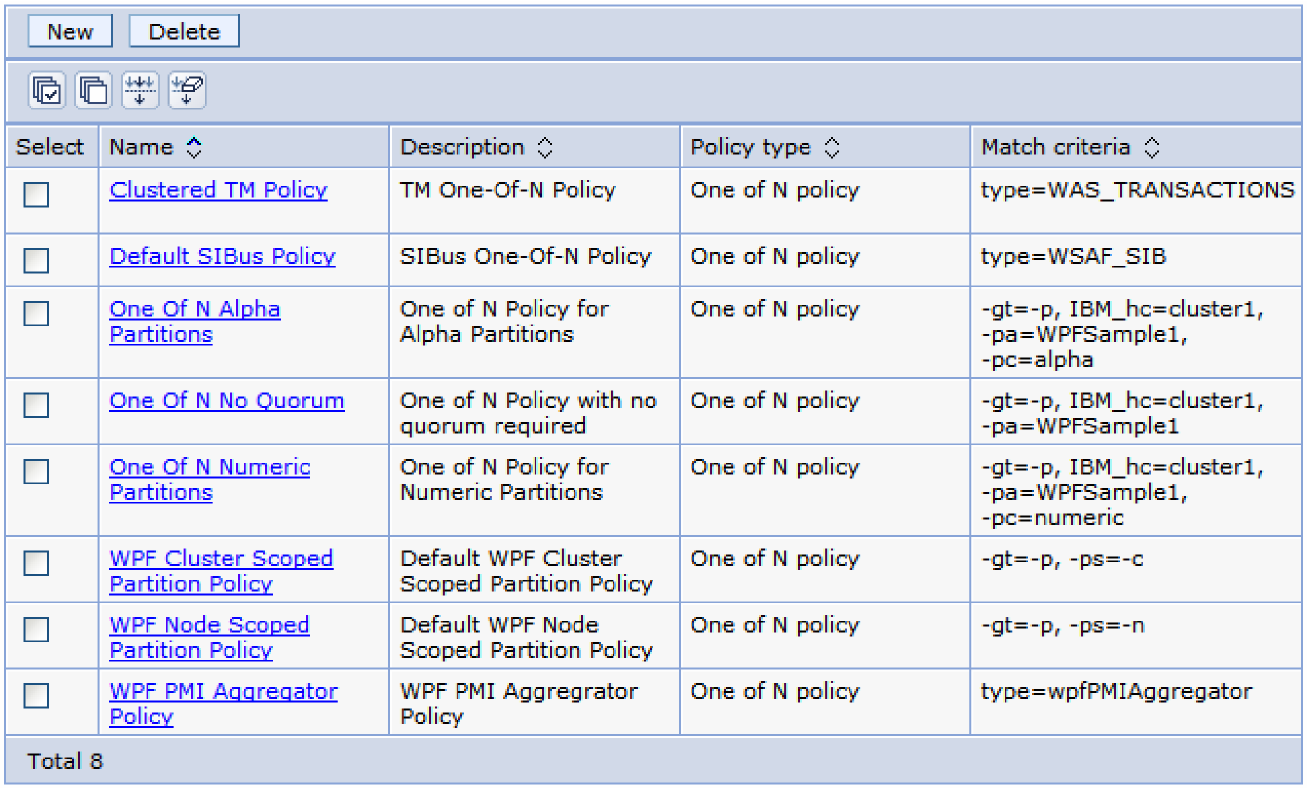
Partition Facility

____ 3. To view the policies you created in the administrative console expand Servers➔ Core Groups and select “Core Group Settings.”

__ a. In the Core Groups list, click DefaultCoreGroup

__ b. On the DefaultCoreGroup General Properties panel, click Policies under Additional Properties.



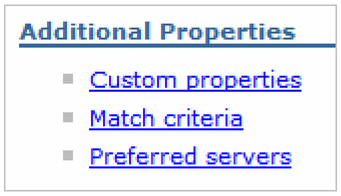


Select	Name	Description	Policy type	Match criteria
☐	Clustered TM Policy	TM One-Of-N Policy	One of N policy	type=WAS_TRANSACTIONS
☐	Default SIBus Policy	SIBus One-Of-N Policy	One of N policy	type=WSAF_SIB
☐	One Of N Alpha Partitions	One of N Policy for Alpha Partitions	One of N policy	-gt=-p, IBM_hc=cluster1, -pa=WPFSample1, -pc=alpha
☐	One Of N No Quorum	One of N Policy with no quorum required	One of N policy	-gt=-p, IBM_hc=cluster1, -pa=WPFSample1
☐	One Of N Numeric Partitions	One of N Policy for Numeric Partitions	One of N policy	-gt=-p, IBM_hc=cluster1, -pa=WPFSample1, -pc=numeric
☐	WPF Cluster Scoped Partition Policy	Default WPF Cluster Scoped Partition Policy	One of N policy	-gt=-p, -ps=-c
☐	WPF Node Scoped Partition Policy	Default WPF Node Scoped Partition Policy	One of N policy	-gt=-p, -ps=-n
☐	WPF PMI Aggregator Policy	WPF PMI Aggregrator Policy	One of N policy	type=wpfPMIAggregator

Total 8

__ c. Click on the PolicyName for the policy you just created.

__ d. On the Policy General Properties tab select “Match criteria” under “Additional Properties.” To view the details of the match criteria, or “Preferred servers” to see the list of preferred servers specified in the policy.



What you did in this exercise

In this exercise, you learned about:

- The basic requirements for implementing a Stateless Session EJB required for implementing a partitioning application.
- The default runtime handling of partitions and their allocation to servers in a cluster.
- The usage of a few of the commands provided by the wpfadmin command line tool.
- The definition and application of special policies to modify the default handling of partitions.
- How to deploy a partitioning application that can be used by EJB clients.