Chapter 7 of *WebSphere Application Server V7 Administration and Configuration Guide,* SG24-7615

WebSphere Application Server for z/OS V7 Administration

In this chapter, we concentrate on the administration features unique to WebSphere® Application Server for z/OS® V7.

We cover the following topics:

WebSphere Application Server on z/OS Architecture

In this section we show the added value that the implementation for WebSphere Application Server for z/OS offers as compared to the distributed versions.

Architecture of a single application server on z/OS

This section gives a conceptual view of an application server inside WebSphere Application Server for z/OS.

Overview

WebSphere Application Server on distributed platforms is based on a single process model. This means that the entire application server runs in a single process, which contains the Java™ Virtual Machine (JVM™). If this process crashes for some reason, all applications that are deployed to this application server will be unavailable unless the application server is clustered.

With WebSphere Application Server for z/OS, a logical application server can consist of multiple JVMs, each executing in a different address space. These address spaces are called servant regions (SR), each containing one JVM. If a servant region abends, another servant region can take over the incoming requests in an multiple-servant environment.

In fact, each logical application server on z/OS has cluster capabilities through the use of multiple servants. These mini clusters benefit from cluster advantages such as availability and scalability without the overhead of a real cluster. This is a key differentiator against distributed platforms.

With regard to administration WebSphere Application Server for z/OS uses the same concepts as distributed environments to create and manage application servers. However, each application server is consists of multiple address spaces that represent a single logical application server:

- Control region (CR)
- Servant region (SR)
- Control region Adjunct

At minimum, one application server consists of one control region and one servant region. Additional servant regions can be added statically by defining a minimum amount of servant regions. Defining a maximum amount of servants, that is higher than the minimum amount allows the z/OS Workload Manager (WLM) to add more servants dynamically according to the demand of system resources. In practice the amount of servant regions is limited by the physical memory available on the system.

The main responsibility of the control region is to handle the incoming connections from the clients and dispatch the request to the WLM queues. Therefore the control region is equipped with its own JVM. Only IBM® written code runs In this JVM, as opposed to the servant region, where the application code runs. There is only one control region per application server. In the unlikely case that the servant region abends, the incoming requests cannot be handled anymore. In this case availability can only be assured with a real clustered server. Because the control region only runs IBM authorized code, this case is a lot more unlikely than a crashing servant region.

Figure 1 illustrates these concepts.

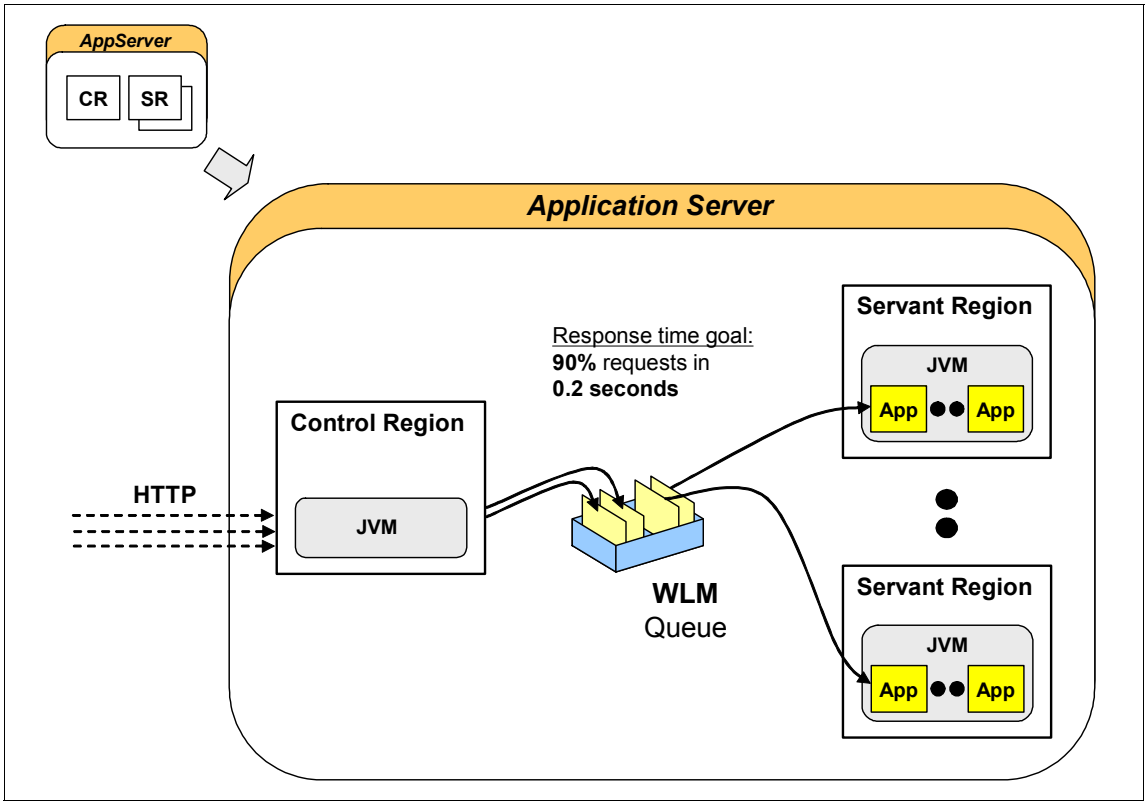

Figure 1 Architecture on a singe application server

The z/OS Workload Manager (WLM) allows you to prioritize work requests on a transaction granularity, compared to server granularity on a distributed environments. Therefore, a service class will be assigned to each work request. For instance, you can define a service class in WLM that has the goal to complete 90% of the requests within 0.2 seconds. The WLM tries to achieve this goal with all available resources and if response times of user transactions do not meet the defined goals, the WLM starts additional servant regions to process the incoming work requests. In other words, the WLM gives you the opportunity to define Service Level Agreements in the form of service classes. Because the WLM is necessary for all subsystems within z/OS this is a huge advantage compared to distributed environments.

A WLM queue is used to queue work for further processing. Each WLM queue uses a first-in-first-out mechanism and represents a service class. Servant regions are bound to a certain priority and therefore take work from the queue with the priority they are bound to. In other words one servant region can only serve one service class, but one service class can served by multiple servants. If additional service classes are defined, additional servant regions are required to process the workload.

All the possible profiles that can be instantiated on z/OS, are built using a control region and servant region:

- Application server
- Deployment manager
- Job manager
- Administrative agent

Although an application server is based on multiple components, it is still a single instance of a server from an application developer, system administrator or user perspective. This means that nearly all WebSphere variables are defined against the server, not each component of the server. However, some of the settings, such as heap sizes, have to be defined for each of component separately (control region, servant region and adjunct region) as shown in Figure 2. In the administrative console under **Application servers** → ***appserver*** → **Process definition**, you can define the heap size for each component.

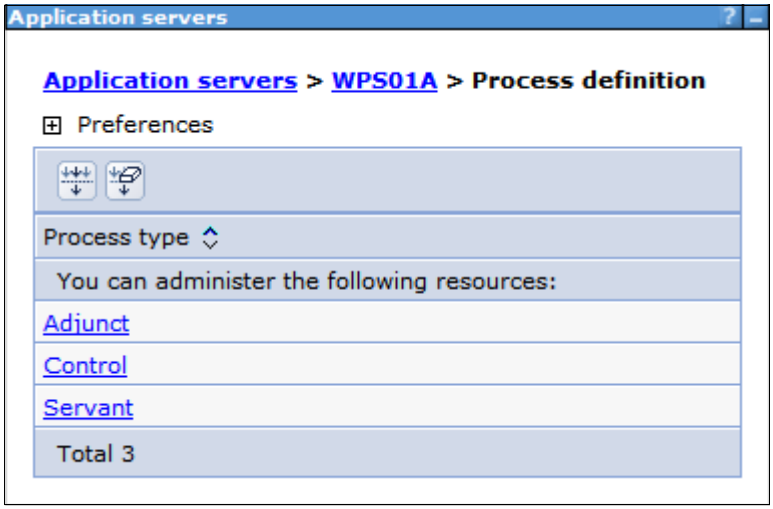

Figure 2 JVM settings of CR, SR, and Adjunct in the administrative console

Basically, the servant specific settings in the administrative console apply for all servants. For instance, if the servant minimum heap size is set to 256 megabyte in the administrative console, all servants will have a JVM with this minimum heap size. There is no possibility to differentiate between different servants.

Attention: This difference in platform settings also applies to **`wsadmin`** scripting. The major difference between scripting on WebSphere for z/OS versus distributed operating systems is that WebSphere for z/OS has multiple JVMs within one logical application server (control, servants, and adjunct). On distributed platforms, the **`wsadmin`** command to change the heap size of a particular application server is unique to the server JVM, whereas it is not on WebSphere for z/OS.

Control region

The control region is the face of the application server to the outside world. It is the only component that is reachable externally using standard protocols and port communication. For communication with the servant regions, where the application is executed, the control region is the end point for TCP transportation and switches to WLM queues.

Here are some facts to keep in mind about control regions:

- An application server can only have one control region.
- The control region contains a JVM.
- The control region will be the end point for communication with clients.

Control region adjunct

The control region adjunct is a specialized servant that interfaces with service integration buses to provide messaging services.

Servant region

The servant region is the component of an application server where the actual application runs and transactions are processed in a JVM. The EJB™ and Web container are in the servant region.

Through the use of multiple servants, it is possible to actually benefit from cluster advantages without the overhead of a real cluster. For additional robustness, 24x7 availability, and scalability, we recommend that you build an application server cluster that integrates these mini clusters. You still can use multiple servant regions for each cluster member.

Here are the most important things to note about servant regions:

- Each servant region contains its own, independent JVM.
- All servant regions are identical to each other.
- An application runs on all servant regions connected to an application server, because it is deployed at server scope.
- An application must be WebSphere Application Server cluster-ready to utilize the multi-servant concept.
- The number of servant regions is transparent to the user and the application.
- Servant regions can dynamically be started by the WLM component, if response times of user transactions do not meet the defined goals. However, the defined maximum is the limit.
- If a single servant fails, the others will still run, keeping the application alive. Only the transactions of the crashed servant region will fail and deliver errors to the user. The other servant regions will continue to work.
- Failed servant regions will be restarted automatically by the operating system providing a miniature automation.

> **Note:** When determining the maximum number of servant regions, make sure that the system has enough resources to utilize them all.

Changing the number of servants

The number of servants to be started initially and dynamically through WLM can be defined in the following path of the administrative console: **Application Servers** → ***app_server*** → **Java and Process Management** → **Server Instance**. See Figure 3.

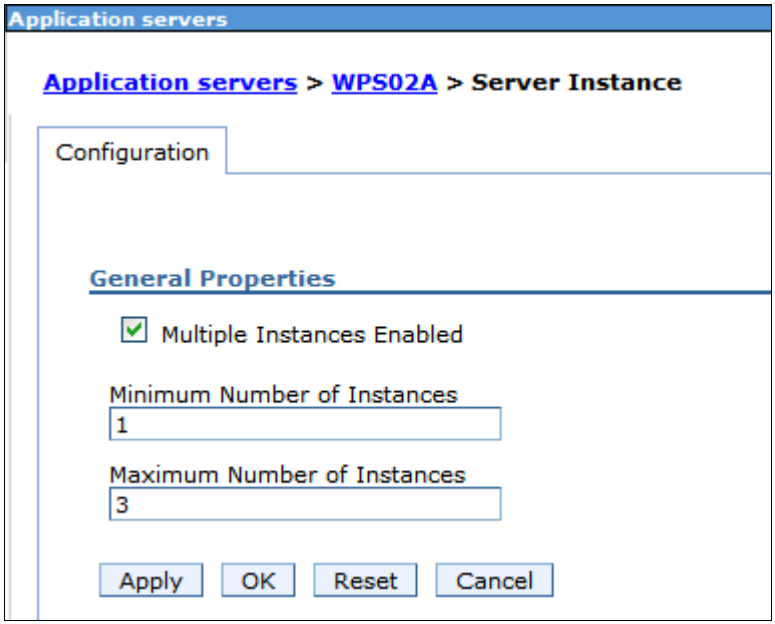

Figure 3 Setting the minimum and maximum amount of servants

In this panel, the minimum and maximum amount of servants can be defined. To use more that one servant region, the check box "**Multiple Instances Enabled**" must be checked.

The minimum amount of servants will be started initially during startup of the application server. If response times of user transactions do not meet the defined goals, the WLM can start additional servant regions. Therefore the maximum amount of servants defined must be higher than the minimum amount. If both numbers are the same, the WLM will not be able to start additional servant regions dynamically. Moreover, the WLM can only start additional servants dynamically until the maximum number of servants is achieved.

The default setting is minimum instances = 1 and maximum instances = 1 with "Multiple Instances Enable" unchecked. In this configuration the mini cluster will not be used.

Example 1 on page 8 shows an SDSF panel with all active started tasks (STCs) of a complete cell, including a deployment manager, daemon, node agent and

two application servers WPS01A and WPS02A. The jobname for all application server components (control region and servant region) starts with the same characters, but the servant region has an additional name suffix "**S**". Another way to distinguish the components is to look at the procedure step (ProcStep) which is BBOPACR for the control region and BBOPASR for the servant region. Furthermore, server WPS02A consists of two servants, which have the same STC name WPS02AS.

Example 1 SDSF panel

```
SDSF DA SC04  SC04     PAG    0  CPU/L/Z  81/ 40/  0   LINE 1-10 (10)
COMMAND INPUT ===>                                          SCROLL ===> PAGE
NP   JOBNAME  StepName ProcStep JobID    Owner    C Pos DP Real Paging    SIO
     WPDEMN   WPDEMN   BBODAEMN STC13150 WPDMGR     NS  FE 5332   0.00   0.00
     WPDMGR   WPDMGR   BBOPDCR  STC13145 WPDMGR     NS  F6  91T   0.00   0.00
     WPDMGRS  WPDMGRS  BBOPDSR  STC13152 WPDMGRS    IN  F4 175T   0.00   0.00
     WPAGNTA  WPAGNTA  BBOPACR  STC13151 WPACR      NS  F6  62T   0.00   0.00
     WPS01A   WPS01A   BBOPACR  STC13154 WPACR      NS  F6  58T   0.00   0.00
     WPS01AS  WPS01AS  BBOPASR  STC13155 WPASR      IN  F4  98T   0.00   0.00
     WPS02A   WPS02A   BBOPACR  STC13333 WPACR      NS  F6  86T   0.00   0.00
     WPS02AS  WPS02AS  BBOPASR  STC13334 WPASR      IN  F4  87T   0.00   0.00
     WPS02AS  WPS02AS  BBOPASR  STC13335 WPASR      IN  F4  88T   0.00   0.00
```

As a rule of thumb for problem determination, communication errors with the client should appear in the log of the control region, whereas application related errors should appear in the log of the servant region.

Availability

The concept of a separate servant and control region greatly enhances the availability of a user application:

- Multiple servant regions can form a kind of vertical cluster, running your application. In the case that one servant region goes down, users with in-flight transactions in that servant will receive an error. However, the other servant regions will continue to work and respond to requests. So the overall application is still available and new requests can enter the system. In addition, z/OS will automatically re-launch the failed servant.
- The control region might be identified as a single point of failure. Although the control region is unique per application server, the risk of failure is very low. Only WebSphere Application Server for z/OS product code is executed in this region. To make applications available, keep in mind that it is also possible to create a WebSphere Application Server cluster, as in an distributed environment.

Performance

From a performance point of view the concept of different regions and the usage of WLM greatly enhances performance and scalability.

- Performance improvements are achieved by creating multiple servant regions, because more requests can be processed in parallel, as long as the system has enough resources available.
- You can set very detailed performance targets on a transactional level for the response time. The system will automatically adjust resources on a 7x24x365 base trying to make sure that these goals are kept.

Through the usage of multiple JVMs with smaller heap sizes the penalty a single transaction has to pay during garbage collection, will decrease and therefore the general throughput will increase.

Cell architecture of WebSphere Application Server for z/OS

This section describes the cell architecture of the WebSphere Application Server for z/OS.

Overview

A cell can span multiple z/OS images and in a heterogeneous cell; even a mixture of z/OS images and distributed platforms are possible. In contrast, a node is always dedicated to a certain z/OS image, because each node has its own file system.

The deployment manager is a specialized application server in a distributed environment. It has its own deployment manager node and hosts the administrative console (administrative console). As with an application server, the same concept of CR and SR applies to the deployment manager as well, but the deployment manager is limited to one servant region, as illustrated in Figure 4. The administrative console J2EE™ application, itsclite, runs in the JVM of the servant region.

In a standalone application environment, the administrative console is deployed into the application server.

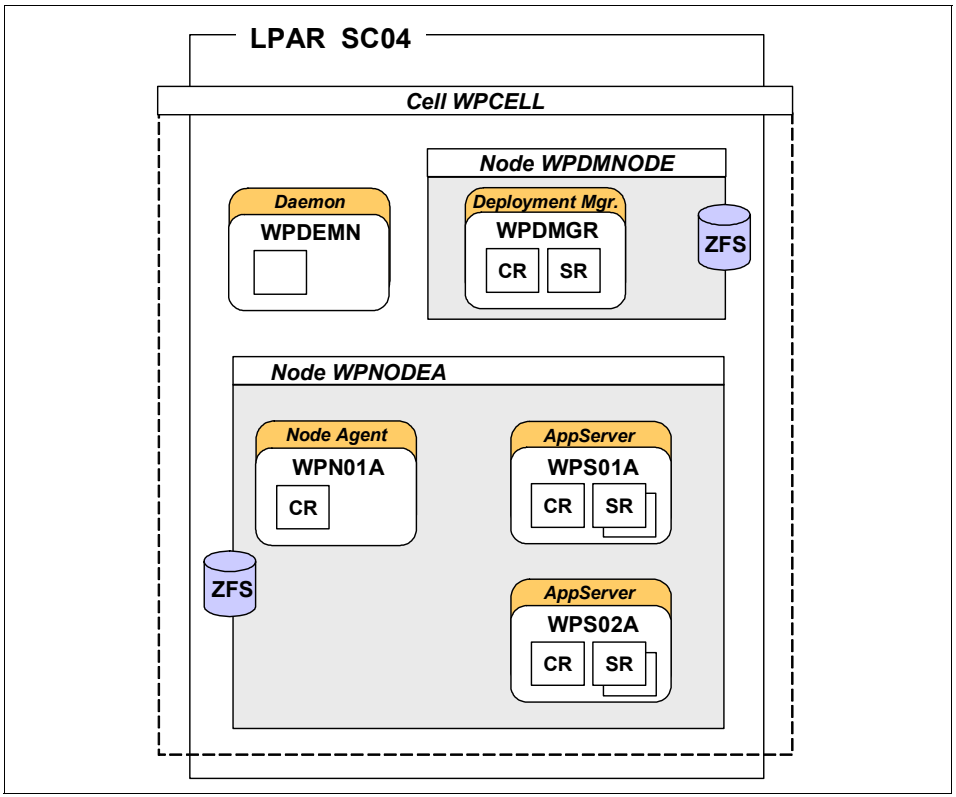

Figure 4 Architecture of a cell

WebSphere Application Server for z/OS introduces a WebSphere cell component exclusive to the z/OS platform: the location service daemon (LSD). In WebSphere Application Server for z/OS terminology the Location Service Agent is called daemon. It provides the location name service for external clients. For instance this service is used by clients to identify a target EJB within a cell. Another task of the daemon is to provide access to modules in storage for all servers within the cell on the same system.

There is one daemon per cell per z/OS image. If your cell consists of multiple z/OS images, a daemon will be created for each z/OS image where your cell exists. If there are two cells on the same z/OS image, one daemon will be created for each cell. The daemon is started automatically when the first control region for the cell on that z/OS image is started. If you terminate a daemon with a MVS STOP command, all the WebSphere Application Server components belonging to the cell on that z/OS image shut down. This is the easiest way to shut down all components of a cell on a specific system. The daemon is created

as a part of the customization process. Moreover, within a WebSphere z/OS cell the daemon in the only address space that is not equipped with its own JVM.

In a distributed environment, one node agent will be created for each node. It is the responsibility of the node agent to synchronize the node configuration with the master repository of the deployment manager. Moreover, the node agent administers the application servers belonging to the same node. A node agent consists of one control region; It does not require a servant region.

In a WebSphere Application Server for z/OS distributed environment with one application server, there will be a minimum of six address spaces:

- Deployment manager control region
- Deployment manager servant region
- Location Service daemon (daemon)
- Application server control region
- Application server servant region
- Node agent
- (Optional) Application server control region adjunct

A standalone server installation would consist of at least three address spaces:

- Location Service daemon (daemon)
- Application server control region
- Application server servant (assumed that one servant is used)
- (Optional) Application server control region adjunct

Making the deployment manager mobile

In a cell that spans multiple z/OS systems, the deployment manager is located in only one of those systems. In the case of an outage of the system where the deployment manager is located, it would be helpful if the deployment manager can be restarted on other systems of the same sysplex.

The white paper, *Making the DMGR Mobile*, provides step-by-step instructions for implementing a mobile deployment manager using the capabilities of the sysplex distributor. This paper is available at:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101140

Introducing job manager and secure proxy server for z/OS

A job manager is a specialized base application server on z/OS, which can efficiently administrate multiple cells on z/OS. The Job Manager has been introduced with WebSphere Application Server for z/OS V7. The white paper, *Introducing The WebSphere V7 Job Manager for z/OS*, from the Washington Systems Center (WSC) provides step-by-step instructions to implement a job manager on z/OS.

This paper is available at:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101341

In addition, the Secure Proxy Server for z/OS has been introduced with WebSphere Application Server for z/OS V7. The Secure Proxy Server is a special security hardened proxy server for use in a Demilitarized Zone (DMZ). To use the Secure Proxy Server on z/OS, a DMZ must implemented on z/OS using a firewall.

The white paper, *Introducing the WebSphere V7 Secure Proxy Server for z/OS*, provides the information required to set up a Secure Proxy for z/OS. This paper is available at:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101423

WebSphere Application Server for z/OS operations

In this section we discuss various aspects of operations in a WebSphere Application Server for z/OS environment that are unique to z/OS.

Structure of the configuration HFS

In WebSphere Application Server for z/OS, the product HFS is strictly separated from the configuration HFS. As the name indicates, the configuration HFS contains the complete configuration of one node, consisting of many XML configuration files. These files are encoded in ASCII. The configuration is mainly located in the *install_root*/profiles/default directory of the configuration HFS. Symbolic links in the configuration HFS, for instance in the *install_root*/bin directory, refer to files in the product HFS.

In contrast, the product HFS contains all necessary product files such as shell scripts, which can be shared across multiple nodes. The product HFS is maintained by System Modification Program/Extended (SMP/E).

During the customization process of a new node, you can either select an HFS or ZFS™ for the configuration file system. We recommend that you use ZFS for the configuration file system, because the performance of ZFS compared to HFS is significantly better and the possibilities to extend a ZFS file system are more flexible.

Attention: If you have a sysplex environment with multiple systems and you are using a shared USS file system, you need to be aware that the configuration file system is always mounted to the system where the corresponding started tasks are running. Otherwise this can have a major impact on performance, because the configuration file system always needs to be mounted in `RDWR` mode. If the configuration file system is mounted on a different system, the XCF traffic will increase dramatically in order to negotiate the writing permissions with the other systems.

The product file system can be shared across the sysplex. In order to avoid a performance impact, the product HFS / ZFS should be mounted in `READ` mode.

We recommend that you use one configuration dataset for each node, instead of using the same dataset for multiple nodes. There are several reasons for separating each node into a different dataset. Probably the most important one is, that the migration from one major version to another major version (for instance V6.1 to V7) is done by a node-by-node basis. For availability and performance reasons it is better to separate each node configuration by using different datasets.

Node synchronization

In a distributed environment the synchronization of nodes is important, especially if the security configuration has been changed.

The configuration file system of the deployment manager contains the master repository, which includes the XML configuration files of the complete cell. The master repository is located in `<DMGR HOME>/profiles/default/config`. In addition, each node has its own configuration file system where a local, reduced copy of the cell configuration and the complete copy of the node specific configuration files are stored. The master repository and configuration files for each node should always be in sync. For instance if important security changes are not propagated to the nodes, this can cause major communication problems between the deployment manager, node agents and application servers. It is the responsibility of the node agent to keep its own node file system in sync as illustrated in Figure 5.

By clicking "**Save**" after a change to an application server setting in the administrative console, the corresponding XML configuration files in the master repository will be changed first. In order to take effect, the adapted configuration files need to be copied to the configuration file system of the corresponding node. During the process of synchronization the node agent connects via TCPIP to the deployment manager and requests all configuration files that have changed since

the last synchronization. The updated configuration files will be copied into the local configuration file system by the node agent.

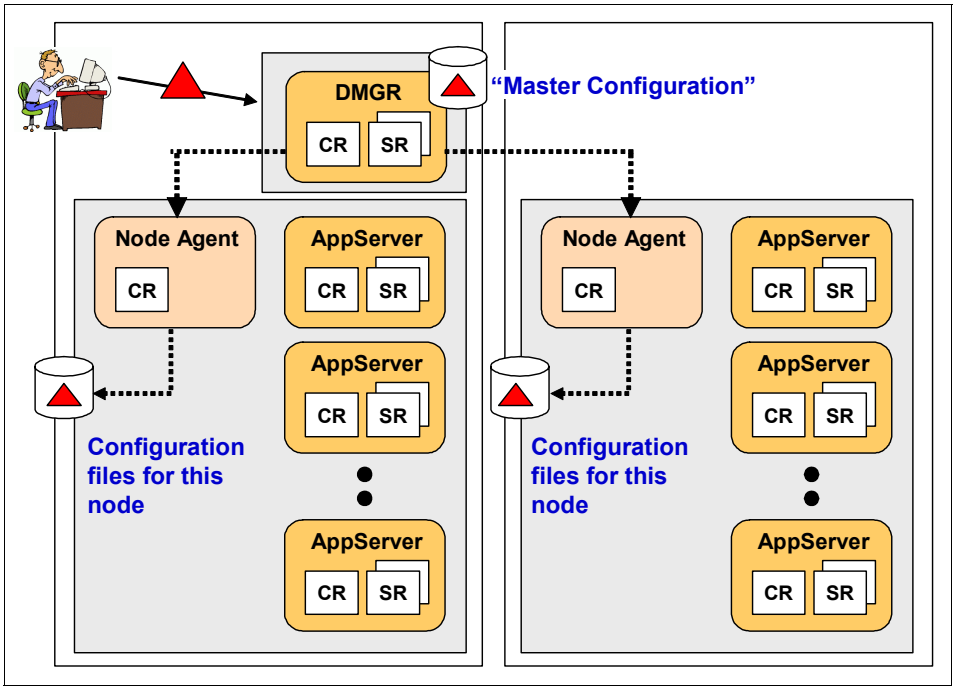

Figure 5 Node synchronization

Attention: In order to keep the consistency of the configuration file system, we recommend that you do not change any values directly in XML configuration files. A lot of settings require the change of multiple configuration files that are related to each other. If these dependencies are not taken into account, it is possible that the configuration settings are not in sync with each other anymore. Instead we recommend that you use the wsadmin shell with connection type "none", which requires no running process and changes the configuration files directly while considering all dependencies.

Load module libraries in the HFS

In WebSphere Application Server V7, the load libraries are now included in the product HFS instead of separate MVS datasets. Consequently the load module libraries do not have to be added to the STEPLIB, LNKLST or LPA anymore. This simplifies the maintenance of WebSphere z/OS dramatically compared to previous versions because it eliminates the possibility of an accidental mismatch

between the configuration HFS and the load module datasets. In previous versions, the configuration HFS always had to be in sync with the load module datasets.

> **Attention:** In order for this function to work properly, the fixes of the following UNIX® System Services APARs need to be applied:
>
> - OA22093
> - OA22094
> - OA25489
>
> These fixes provide an enhanced LOADHFS call to load the WebSphere modules directly from the HFS.

Within the configuration HFS, the load module libraries are now located in the directory *install_root*/lib/modules and begin the file name with “bbo”. In addition, the symbolic links in this directory point to <SMP/E_HOME>/lib/modules of the product HFS.

If you continue to use MVS datasets for the load modules, you can switch the configuration between using load modules in the file system and using load modules in a dataset with the **switchmodules** command, as described in the following Information Center article:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.installation.zseries.doc/info/zseries/ae/rins_switchmod.html

This major change also has implications for the start procedure JCLs, as described in the following section.

Changed start procedure JCLs with V7

In order to start a WebSphere Application Server component, such as a deployment manager, application server or node agent, different start procedures (JCLs) are used. As part of the customization process these procedures will be generated and copied into the PROCLIB. Some components such as the deployment manager and daemon have their own unique start procedure for each control region and servant region. It is common practice that all application servers belonging to the same node and the node agent share the same CR procedure and the same SR procedure as pointed out in Figure 6.

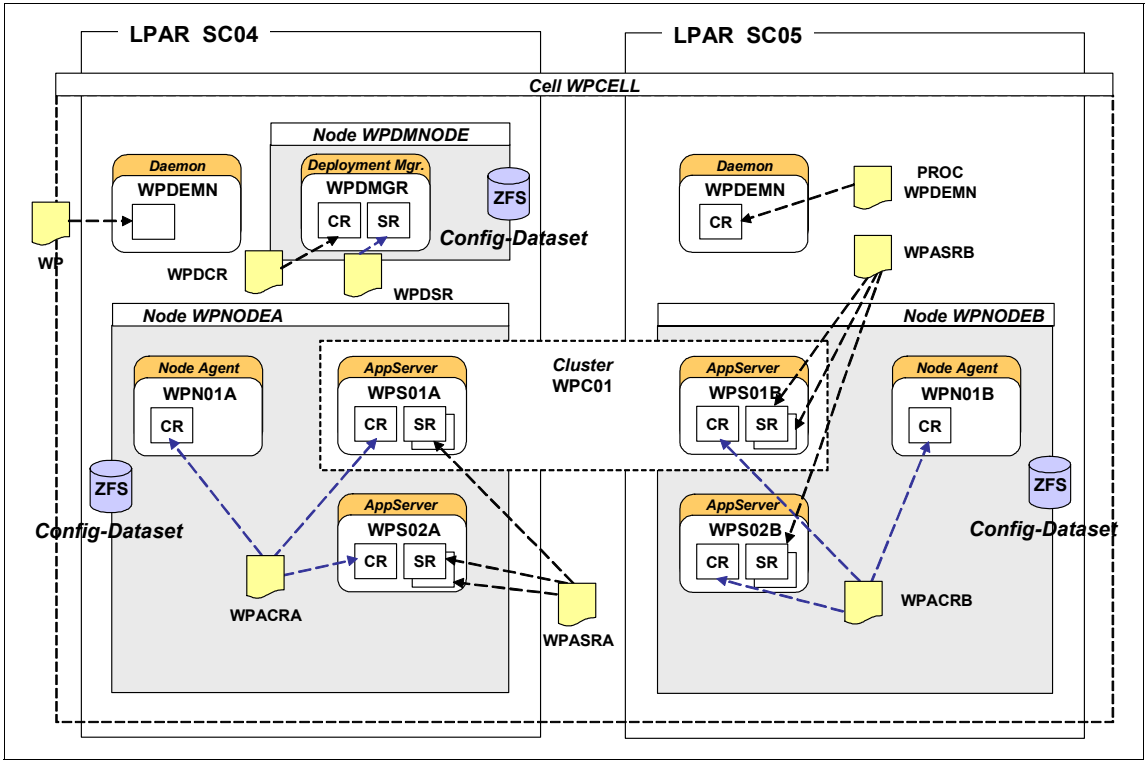

Figure 6 Reuse of PROCLIB members

The advantage of sharing the same procedures across all application servers of a node is that new application servers can be easily added using the administrative console without the need to define additional procedures. In addition, new start procedures usually require additional STARTED profiles in RACF®.

The reason why the start procedures usually cannot be shared across multiple nodes is that the USS path to the WebSphere configuration file system is part of the procedure, as shown in Example 2. Theoretically, this can be realized by making use of additional variables for the configuration path. Consequently the MVS start command has to be extended for that new variable.

With WebSphere Application Server for z/OS V7, all load libraries are located in the USS file system instead of MVS datasets. Consequently, all start procedures have to be adapted in V7. Also, the customization job for the start procedures does not create the z-Member in PROCLIB anymore. In the past, this z-Member has been used for including STEPLIBs. Example 2 illustrates JCL where the STEPLIB entries for DB2® can be placed. They are marked in bold.

Example 2 Start procedure of application server CR

```
//WPACRA  PROC ENV=,PARMS=' ',REC=N,AMODE=OO
// SET ROOT='/wasconfig/wpcell/wpnodea'
// SET FOUT='properties/service/logs/applyPTF.out'
// SET WSDIR='AppServer'
//*************************************************************
//* Test that OMVS can successfully launch a shell and return *
//*************************************************************
//TOMVS   EXEC PGM=BPXBATCH,REGION=OM,
// PARM='SH exit 13'
//STDERR   DD PATH='&ROOT./&ENV..HOME/&FOUT.',
// PATHOPTS=(OWRONLY,OCREAT,OAPPEND),PATHMODE=(SIRWXU,SIRWXG)
//STDOUT   DD PATH='&ROOT./&ENV..HOME/&FOUT.',
// PATHOPTS=(OWRONLY,OCREAT,OAPPEND),PATHMODE=(SIRWXU,SIRWXG)
//*********************************************************
//* If the shell RC code is as expected (13) - proceed    *
//*********************************************************
//IFTST   IF (RC = 13) THEN
//*********************************************************
//* Start the Multi-Product PTF Post-Installer            *
//*********************************************************
//APPLY   EXEC PGM=BPXBATCH,REGION=OM,
// PARM='SH &ROOT./&ENV..HOME/bin/applyPTF.sh inline'
//STDERR   DD PATH='&ROOT./&ENV..HOME/&FOUT.',
// PATHOPTS=(OWRONLY,OCREAT,OAPPEND),PATHMODE=(SIRWXU,SIRWXG)
//STDOUT   DD PATH='&ROOT./&ENV..HOME/&FOUT.',
// PATHOPTS=(OWRONLY,OCREAT,OAPPEND),PATHMODE=(SIRWXU,SIRWXG)
//       IF (APPLY.RC <= 4) THEN
//*********************************************************
//* If the RC from the Post-Installer is LE 4 then start  *
//* the WebSphere Application Server                      *
//*********************************************************
//BBOPACR    EXEC PGM=BPXBATA2,REGION=OM,TIME=MAXIMUM,MEMLIMIT=NOLIMIT,
// PARM='PGM &ROOT./&WSDIR./lib/bbooctlm &AMODE. &PARMS. REC=&REC'
//STEPLIB  DD DISP=SHR,DSN=DB8X8.SDSNEXIT
//         DD DISP=SHR,DSN=DB8X8.SDSNLOAD
//         DD DISP=SHR,DSN=DB8X8.SDSNLOD2
//STDENV DD PATH='&ROOT/&ENV/was.env'
//*
//* Output DDs
//*
//DEFALTDD  DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//HRDCPYDD  DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSOUT    DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
```

```
//CEEDUMP   DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//SYSPRINT  DD SYSOUT=*,SPIN=UNALLOC,FREE=CLOSE
//          ENDIF
//IFTSTEND ENDIF
//
```

The Addressing Mode (AMODE) is a JCL parameter that is used in the START command to determine whether the server shall be started in 64-bit or 31-bit mode. The `AMODE` parameter is still supported in V7; however, we recommend *not* to modify the default value. In the generated procedures during the installation, the value 00 is default. This means that the value defined inside the application server's XML files is used for the decision of running 64 or 31-bit mode.

If you start the server, for example, with `AMODE=64`, and the XML files reflect a 31-bit installation (or via versa), then the server will not start.

Note: We recommend that you use the default value for the AMODE (`AMODE=00`) in the startup JCL for the WebSphere Application Server components. However, keep in mind that you need to verify your automation settings.

Starting and stopping an application server

In order to start an application server, it is only necessary to issue the MVS start command for the control region (CR). The corresponding servant regions (SR) will be started by the WLM automatically.

An application server including the CR and all SRs can be stopped by executing the MVS stop command for the control region. Basically the same concept applies to the deployment manager.

Stopping a daemon address space will shut down all address spaces belonging to the same cell and z/OS system.

In Figure 7 the start order of the deployment manager is illustrated. The control region needs to be started first. If there is no daemon address space running, the CR will start the daemon address space. If the deployment manager CR is initialized, the WLM will start the minimum amount of servants. After the SRs have been started successfully, the "open for e-business" notification will appear in the job output of the CR. Next, the node agent CR can be started with the corresponding MVS start command.

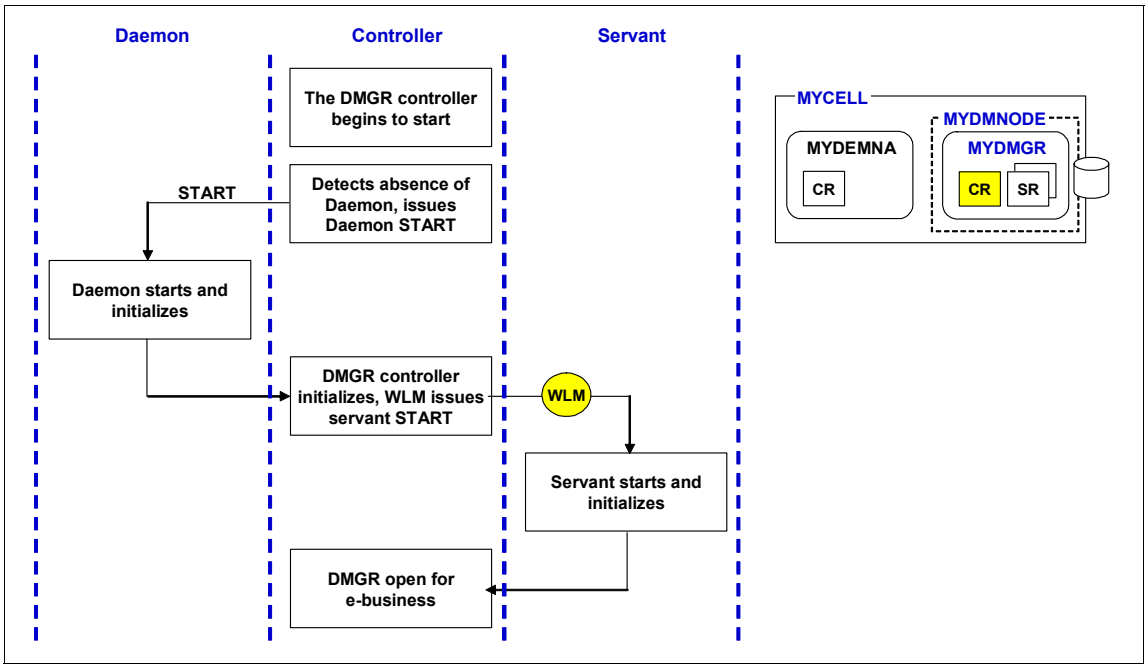

Figure 7 Address space start order

Start procedures can be shared between all application servers belonging to the same node. In order to determine which application server should be started, an `ENV` parameter needs to be specified in the MVS start command. This parameter is structured using the following qualifiers:

```
<cell_short_name>.<node_short_name>.<server_short_name>.
```

For most MVS people, this parameter seems to define an MVS dataset. But in this case, this parameter specifies a symbolic link that points to the configuration directory of the particular server. As illustrated in Figure 8, this symbolic link is located in the mount directory of the corresponding node. For every server belonging to the same node, a symbolic link will be created in this directory. During startup of a server, the `was.env` file in the server configuration directory is required, which contains the most important configurations for that server.

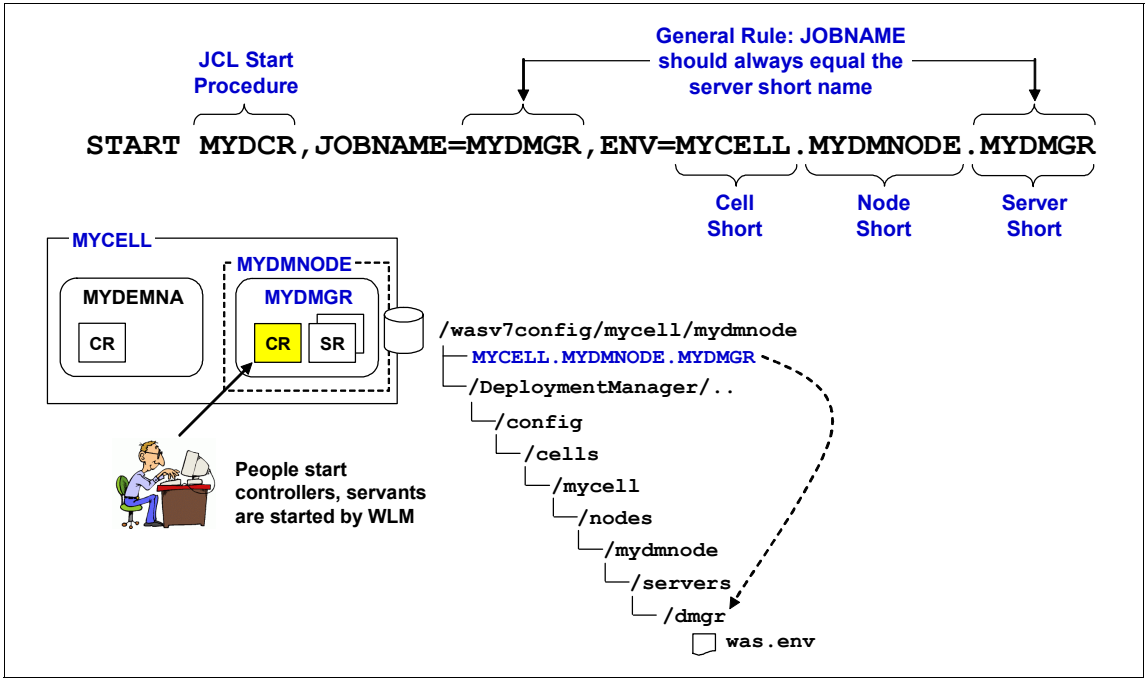

Figure 8 Start command for the deployment manager

Furthermore, this ENV parameter illustrates that WebSphere Application Server always uses the short names when facing the operating system. If a server short name is changed in the administrative console, the corresponding symbolic link will be recreated using the specified short name.

Logging and tracing

In order to enable a WebSphere trace or to change the trace level for a specific server, there are basically two options. As in WebSphere distributed environments, this can be done in the following path of the administrative console: **Logging and Tracing** → ***server_name*** → **Change Log Detail Levels**. Click the **Configuration** tab and add the WebSphere trace specification.

Another option is to use MVS modify commands to dynamically turn on WebSphere traces. Usually MVS modify commands are executed from the SYSLOG. For system programmers, this option is an efficient way to enable traces.

The WebSphere trace can be turned on dynamically from the MVS console using the following modify command:

```
F CR_short_name,TRACEJAVA='com.ibm.ws.security.*=all'
```

The following modify command resets the trace configuration to the original settings from the last startup:

```
F CR_short_name,TRACEINIT
```

After installation, the default log level for an application server is set to *=info, which means that all Java components are logged with trace level `INFO`. This can produce a massive trace overhead especially in an production environment. We recommend that you adjust the trace level of the application server in a production environment to `SEVERE` and in a development and test environment to `WARNING` for all components, in order to reduce the trace overhead. Basically it is not possible to distinguish between CR and SR for defining the log levels. These settings are always valid for the complete application server. The trace output will be written to the started tasks job outputs. These can be viewed for instance in SDSF. For application related problems the servant region should always be the starting point for investigation.

Customers sometimes require the job output to be directed to a file in the HFS file system. This can be done by modifying the corresponding start procedures as described in the white paper, *Directing SYSPRINT Output to an HFS File in WebSphere for z/OS*. It is available on the following Web site:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/TD101087

Because of the additional overhead of writing the log into an HFS file, we recommend that you do *not* redirect the job output.

Maintenance for the HFS

One of the basic concepts of rolling maintenance through an environment is that maintenance can only be applied to the configuration file system on a node-by-node basis. In fact, maintenance can be applied on each node of the cell independently. If the complete cell should be updated to a new service level, the process of maintenance is repeated for each node.

Moreover, it is possible to have each node of a cell on a different maintenance level as long as the deployment manager has the highest maintenance level. For instance, you can have a NodeA with service level 7.0.0.1 and a NodeB with service level 7.0.0.3 as long as the deployment manager itself has the service

level 7.0.0.3. Consequently, *a new service level should always be applied to the deployment manager first*.

The process of applying maintenance

Maintenance is applied first to the product datasets including the HFS dataset, using SMP/E. Each configuration HFS is linked to a dedicated product HFS. During the startup of a deployment manager, node agent, or application server, the applyPTF.sh shell script of the control region will notice a difference in the service level of the configuration HFS compared to the product HFS. (This script runs automatically as part of the server startup and should not be executed manually). If new maintenance has been introduced in the product HFS, applyPTF.sh updates the configuration HFS for the node. This shell script then recreates certain symbolic links within the configuration HFS. As a result, the configuration HFS and the product HFS have the same service level applied.

In WebSphere Application Server for z/OS V7, the load module libraries are located in the HFS instead of MVS datasets, which simplifies the process of maintenance. In previous versions, the load module datasets always had to be in sync with the HFS file system.

The concept of intermediate symbolic links

A lot of customers have isolated their test, acceptance test, development and production environment by using separate cells for each environment. Each cell usually has its own set of product datasets, which need to be maintained by SMP/E. In most cases, customers have fewer maintenance levels in use than different environments. With the use of intermediate symbolic links, you can reduce the number of product datasets by providing product datasets for each service level instead of each environment. These product datasets can be shared by nodes from different environments.

Because the product HFS is only mounted in READ mode, there is no opportunity for other environments to corrupt the product HFS. Moreover, intermediate symbolic links allow you to change the product HFS for each node independently. For instance, NodeA in the test environment and NodeD in the development can share the same product HFS, which has applied a certain maintenance level.

Figure 9 illustrates the use of one product HFS without intermediate symbolic links. In this case all symbolic links within a configuration HFS point directly to the product HFS. If a new maintenance level is applied to the product HFS, all configuration HFSs will be updated automatically. There is no possibility to distinguish different service levels between those nodes.

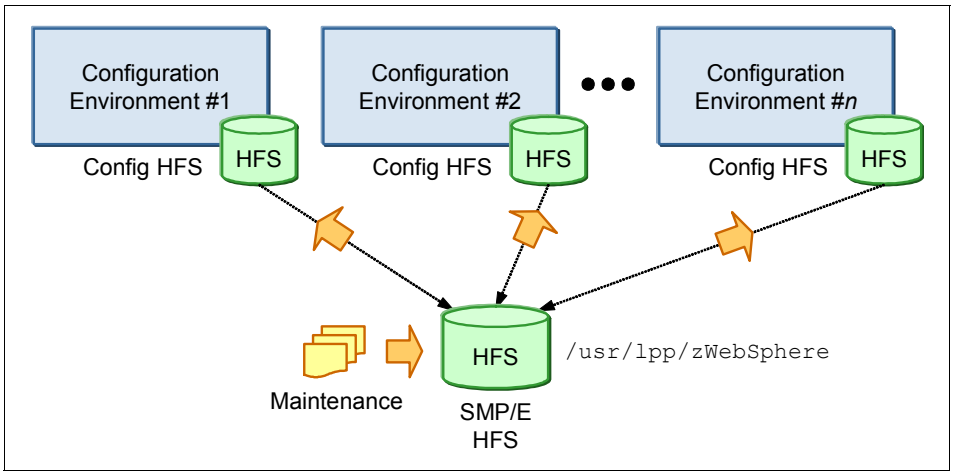

Figure 9 Applying maintenance without intermediate symbolic links

In contrast, intermediate symbolic links offer more flexibility for changing the maintenance level on node-basis. Instead of using a product HFS for each environment such as test, development or production, the concept of intermediate symbolic links accommodates a product HFS for each maintenance level, for instance one product HFS for version 7.0.0.1 and 7.0.0.3. An additional intermediate symbolic link is created, usually located in the mount directory of the node. All symbolic links within the configuration HFS point to the intermediate symbolic link, which refers to the product HFS.

As shown in Figure 10, the new maintenance level 700003 has been applied to a second product HFS. The intermediate symbolic link of node two is switched from the product HFS with version 700001 to the second product HFS with version 700003. During the next startup of a control region in node two, the configuration HFS will be updated to the new maintenance level 700003. Independently from node two, node one still points to version 700001.

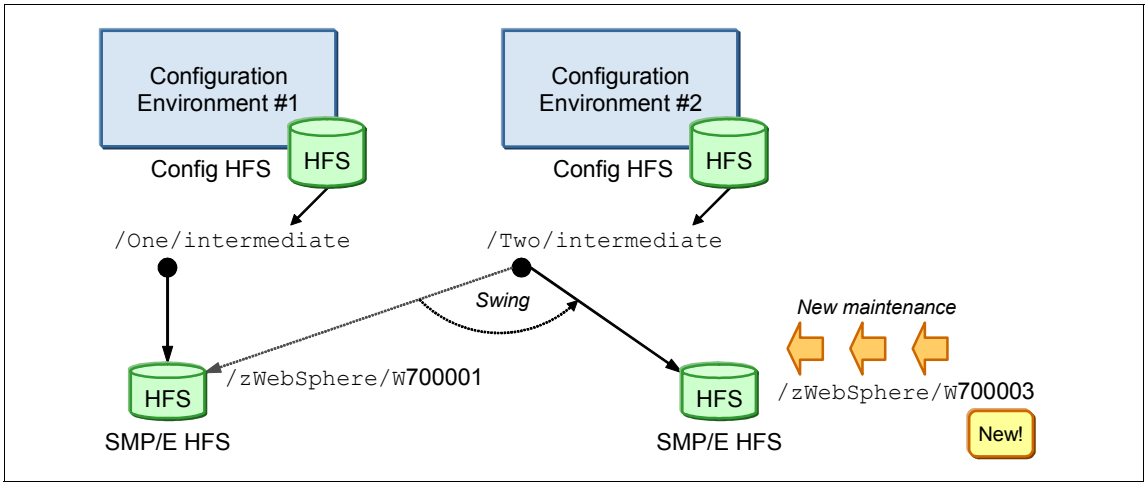

Figure 10 Applying maintenance with intermediate symbolic links

Although you might have only a few environments, we recommend that you use intermediate symbolic links to provide flexibility in changing maintenance levels on a node-by-node basis. During the customization of each WebSphere component in the PMT, you can specify a intermediate symbolic link as shown in Figure 11 on page 25. The check box, **Create intermediate symbolic link**, must be checked. The location of the intermediate symbolic link is usually in the mount point of the node as shown in this example.

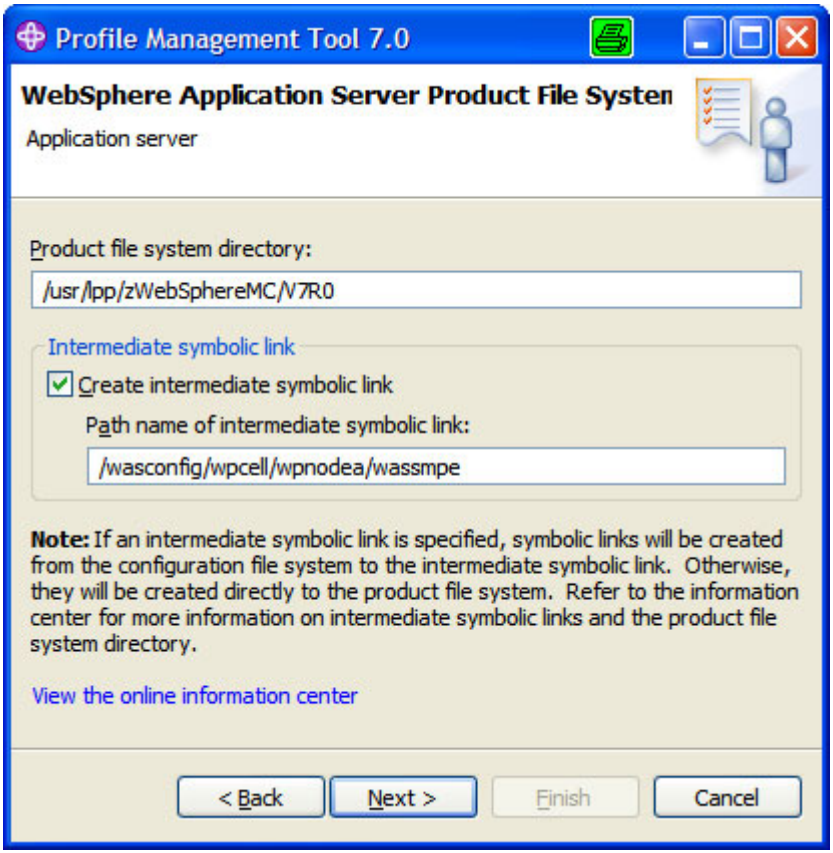

Figure 11 Create intermediate symbolic link during customization

More information about maintenance of WebSphere Application Server for z/OS is provided in the white paper, *WebSphere Application Server for z/OS - Planning for Test, Production and Maintenance*, which is available at the following Web site:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100396

Workload management

This section focuses on how WebSphere Application Server for z/OS exploits the WLM subsystem of z/OS.

Workload management overview

WebSphere Application Server for z/OS V7 can exploit the Workload Manager (WLM) subsystem of z/OS in the following ways:

- Workload classification:

  Coarse grain workload management on a server basis.
- Transaction classification:

  Fine grained workload management on transaction level.
- Servant activation:

  Starts additional servant regions for application processing.

Before we go into more detail on the enhancements that the WLM offers, let us briefly explain the concepts of service classes, reporting classes, and enclaves.

Service class

A service class is used to set performance goals for different work (like incoming requests, applications or operating system tasks). For example, a service class can be told to achieve a response time of 0.2 seconds 90% of the time for incoming requests. The WLM component of z/OS will then automatically assign resources (processor, memory and I/O) to achieve the goals. This is done by comparing the definitions of the service class to real-time data on how the system is currently performing.

You can have multiple service classes, with multiple goals. The mapping of work to a service class is set up by the system programmer and can be based on a variety of choices like user ID, application or external source.

In summary, a service class is the z/OS implementation of a service level agreement.

Enclaves

An enclave can be thought of as a container that has a service class and reporting class attached to it. A thread can connect to this enclave and execute its work with the priority of the enclave.

WebSphere Application Server for z/OS uses this technique to pass transactional work (the user application) from a servant to an enclave. The application then runs with the priority of the enclave and WLM can make sure that the performance goals for the application are achieved.

In summary, an enclave is used to assign the user application a service class during runtime.

Workload classification

WebSphere Application Server for z/OS V7 and its prior versions are capable of classifying incoming work on a server basis. To do this, the control region of an application server checks to see which application server the request belongs to. It will then assign the request accordingly to the WLM queue. Each servant will process work for one service class at any point in time.

As seen in Figure 12 incoming work is assigned a service class, based on information of the user-work request. The granularity is on application server level.

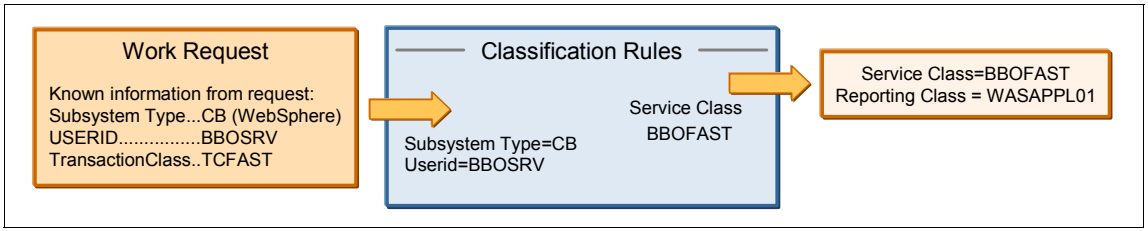

Figure 12 Workload classification for WebSphere Application Server for z/OS

Transaction classification

Transaction classification can be used to classify the transactions handled by your application. This technique can be used to prioritize special requests. A good example is a Web store that classifies its customers in gold and platinum terms, giving the platinum customers a better response time than the gold customers (see Figure 13 on page 28).

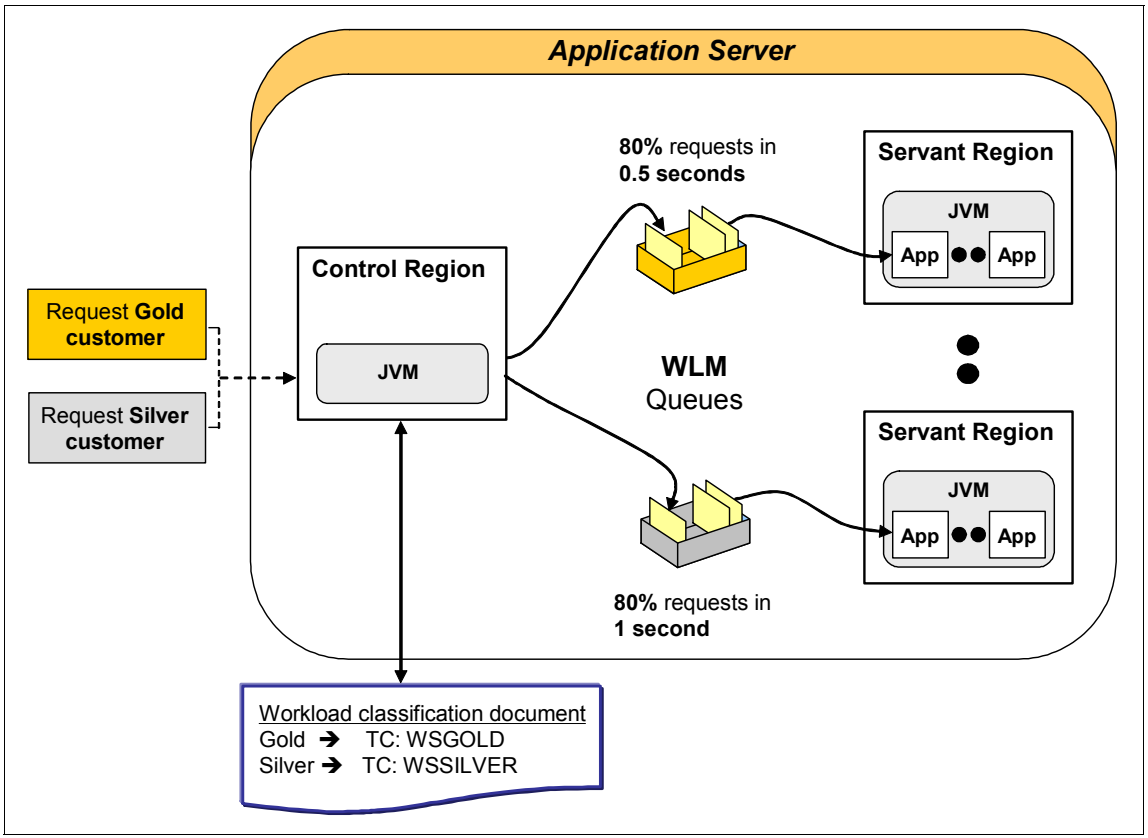

Figure 13 Transactional assignment of performance goals

A request that enters the system is assigned a *transaction class*, using the request details like utilized protocol, requested URI, or other metrics. The transaction class is then mapped to a service and reporting class inside the WLM subsystem, using a *workload classification document*. This file is an XML file that has to be populated with mapping rules.

Transaction classification can be used with the following protocols:

- Internal classification
- IIOP classification
- HTTP classification
- MDB classification
- SIP classification

In order to use transaction classification, you need to perform the following steps:

1. Talk with the application developers and the business functions to define what transactions need performance goals (Service Level Agreements).
2. Create a *workload classification document*.

   In Example 3, we have created a workload classification document for HTTP requests. The document includes transaction classes for multiple applications deployed to different application servers within the same cell.

   The main advantage of the workload classification document is that the filter can be nested. For instance, the host name wtsc04.itso.ibm.com and port 12067 apply to the applications MyIVT, cachemonitor, and PlantsByWebSphere. In this case, we have not distinguished between different JSPs. By using the wildcard /PlantsByWebSphere/* all requests containing this context root are assigned to transaction class PLANTSWS.

Example 3 Example of a workload classification document for HTTP requests

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Classification SYSTEM "Classification.dtd" >
<Classification schema_version="1.0">
<InboundClassification  type="http"
                        schema_version="1.0"
                        default_transaction_class="WPOTHER">
   <http_classification_info  transaction_class="WPOTHER"
                              host="wtsc04.itso.ibm.com"
                              port="12067"
                              description="WPS01A HTTP">
         <http_classification_info  transaction_class="FRCAAPP"
                                    uri="/MyIVT/*"
                                    description = "FRCA Demo Application" />
         <http_classification_info  transaction_class="CACHEMON"
                                    uri="/cachemonitor/*"
                                    description="Cache Monitor" />
         <http_classification_info  transaction_class="PLANTSWS"
                                    uri="/PlantsByWebSphere/*"
                                    description="PlantsByWebSphere" />
   <http_classification_info  transaction_class="DAYTRADE"
                              host="wtsc04.itso.ibm.com"
                              port="12087"
                              uri="/trade/*"
                              description = "Trade Application" />
   <http_classification_info  transaction_class="DAYTRADE"
                              host="wtsc04.itso.ibm.com"
                              port="12088"
                              uri="/daytrader/*"
```

```
                              description = "Trade Application" />
   <http_classification_info  transaction_class="ADMINC"
                              host="wtsc04.itso.ibm.com"
                              port="12006"
                                uri="/ibm/console/*"
                              description = "AdminConsole HTTPS" />
</InboundClassification>
</Classification>
```

This final workload classification document needs to be copied to a path where the application server CR can access it. It is also necessary to create a DTD file in the same directory, which is referenced in the header of the workload classification document. Here is an example of that reference:

```
<!DOCTYPE Classification SYSTEM "Classification.dtd" >
```

You can use the DTD from the Information Center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rrun_wlm_tclass_sample.html

3. Configure the server to use the classification document:
   a. In order to activate the workload classification document, a new WebSphere variable needs to be defined with the path to the document. Select **Environment** → **WebSphere variables** in the administrative console.
   b. Select the scope for the workload classification document. If the WebSphere variable is defined at the cell scope, the classification document is valid for the complete cell.
   c. Click **New.**
   d. Create a new variable named **wlm_classification_file**. The corresponding value should point to the complete USS path of the workload classification document.
   e. Save and synchronize the changes with nodes.
   f. In order for the classification file to take effect in a particular application server, it has to be restarted.
4. Modify the WLM settings to use transaction classes.

   The same transaction classes specified in the workload classification document need to be defined in the Classification Rules of Subsystem CB in WLM. A classification rule maps a transaction class to a defined service class and reporting class. If you want to distinguish between multiple service classes within one application server, one servant region is required for each

service class. Consequently the minimum and maximum amount of servant regions must be adjusted in order to handle the incoming workload.

According to the workload classification document in Example 3 on page 29, we have defined the transaction classes with the corresponding service classes and reporting classes in the following example. In this case the transaction classes should only apply to all started task names with the qualifier WP*. Therefore we defined a main rule (1) with the type CN and name WP*.

All transaction classes have been defined as sub rules (2) of the WP* filter. A transaction class is indicated by the classification type TC. The name of the transaction class must be equal to the `transaction_class` value in the workload classification document. In this example, a reporting class has been defined for each application, and all applications deployed on the same application server share the same service class.

For each report class, a separate workload activity report can be generated with the details about the CPU consumption. In Example 4, we have chosen service class WASHI, which has a response time goal of 80% requests in 0.5 seconds.

Example 4 Defining classification rules in WLM

```
Modify Rules for the Subsystem Type    Row 34 to 40 of 41
Command ===> ____________________________________________ SCROLL ===>
CSR

Subsystem Type . : CB         Fold qualifier names?   N  (Y or N)
Description  . . . ________________________________

Action codes:   A=After     C=Copy       M=Move     I=Insert rule
               B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                          More ===>
         --------Qualifier--------              -------Class--------
Action    Type       Name     Start               Service     Report
                                          DEFAULTS: WASHI       ________
 ____  1  CN         WP*      ___                  WASHI       ________
 ____  2    TC         PLANTSWS ___                WASHI       WPPLANTS
 ____  2    TC         DAYTRADE ___                WASHI       WPDAYTRD
 ____  2    TC         FRCAAPP  ___                WASHI       WPFRCAAP
 ____  2    TC         FRCAENC  ___                WASHI       WPFRCA
 ____  2    TC         CACHEMON ___                WASHI       WPCACHEM
 ____  2    TC         ADMINC   ___                WASDM       WPADMINC
 ____  2    TC         WPOTHER  ___                WASDM       WPOTHER
```

The changes of the WLM configuration must be activated in order to take effect.

For detailed information about transaction classifications, refer to the Information Center article, *Using transaction classes to classify workload for WLM*. This article contains links to all information sources needed, as well as samples. It can be found at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0//index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rweb_classservers.html

5. In order to verify the defined workload classification, we have invoked several applications at the same time using a load generator and have monitored the classified enclaves in SDSF. All the active enclaves with the assigned service class and report class can be viewed in the SDSF.ENC panel (Example 5).

Example 5 SDSF Display of Enclaves

```
SDSF ENCLAVE DISPLAY  SC04     ALL
COMMAND INPUT ===>
NP   NAME             SSType Status   SrvClass Per PGN RptClass
     5400022F1B       CB     ACTIVE   WASDM      1     WPADMINC
     5400022F45       CB     ACTIVE   WASHI      1     WPCACHEM
     2C00022F38       CB     ACTIVE   WASHI      1     WPFRCAAP
     3800022838       CB     ACTIVE   WASHI      1     WPFRCA
     3000022F65       CB     ACTIVE   WASHI      1     WPPLANTS
     2000000001       STC    INACTIVE SYSSTC     1
     2400000002       TCP    INACTIVE SYSOTHER   1
```

Servant activation

An application server can have multiple servant regions that process the user application. If the response time goals defined for the applications cannot be kept, WLM can start additional servant regions to process incoming, or queued up requests faster.

The minimum and maximum number of the servant regions can be defined as described in “Servant region” on page 6.

Basic WLM classifications

The usage of WLM classification for the control and servant region address spaces is a basic z/OS approach. It is part of the installation process of the WebSphere Application Server for z/OS V7.

The following recommendations apply:

- Control regions should be assigned a service class with high priority in the system, whether it is the `SYSSTC` service class or a high importance or velocity goal. This is because controllers perform processing that is required to receive work into the system, manage the HTTP transport handler, classify the work, and do other housekeeping tasks.
- The servant classification should not be higher in the service class hierarchy than more important work, such as the controller and CICS®, or IMS™ transaction servers. We recommend that you use a high velocity goal.

In Example 6, the WLM classification rules for all started tasks (STCs) that belong to a distributed environment have been defined, including a deployment manager, daemon, node agents and application servers. In order to use the STC name for classification, the type TN must be selected. The qualifier name specifies the STC name. In this example, wildcards have been used to define one classification rule that is valid for all application server CRs. We have assigned service classes with higher velocity goals to the application server CR and daemon. We have assigned service classes with lower velocity goals to the deployment manager, node agent, and application server SR.

Example 6 WLM classification of the CR and SR started tasks

```
Modify Rules for the Subsystem Type   Row 16 to 22 of 106
Command ===> ____________________________________________ SCROLL ===>
CSR

Subsystem Type . : STC        Fold qualifier names?   N  (Y or N)
Description  . . . ______________________________

Action codes:   A=After     C=Copy       M=Move     I=Insert rule
               B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                          More ===>
         --------Qualifier--------              -------Class--------
Action    Type       Name     Start               Service     Report
                                          DEFAULTS: STC_HIGH    ________
 ____  1  TN         WPDMGR   ___                  STC         WPDMGR
 ____  1  TN         WPDMGRS  ___                  STC         WPDMGRS
 ____  1  TN         WPDEMN   ___                  STC_HIGH    WPDEMN
 ____  1  TN         WPAGNT%  ___                  STC         WPAGNT
 ____  1  TN         WPS%%%   ___                  STC_HIGH    WPACR
 ____  1  TN          WPS%%%S  ___                   STC         WPASR
```

Refer to the "*Controller and Servant WLM classifications*" article in the Information Center found at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0//index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rweb_classervers.htm

- Enclaves for WebSphere Application Server for z/OS are classified using the `Subsystem CB`. The performance goals set here depend on your application and the environment, therefore no quantitative recommendation can be made here. However, usually a percentile response time goal is advisable.
- OMVS components of WebSphere Application Server for z/OS need to be classified as well. Some OMVS scripts are executed during server startup, therefore if these are not classified in the WLM, the server startup time will be increased.

Attention: WLM classification for OMVS components:

A step in the control region start-up procedure invokes the applyPTF.sh script, using BPXBATCH. Because the BPXBATCH program is classified according the OMVS rules, on a busy system, several minutes might pass before this step is completed.

You can minimize the impact of the BPXBATCH step by changing the WLM Workload Classification Rules for OMVS work to a higher service objective (see Example 7).

Example 7 Service class definition for OMVS components

```
Subsystem Type . : OMVS    Fold qualifier names?   Y  (Y or N)
Description  . . . OMVS subsystem rules mod for WAS

Action codes: A=After   C=Copy        M=Move    I=Insert rule
              B=Before  D=Delete row  R=Repeat  IS=Insert Sub-rule
                                                      More ===>
      --------Qualifier--------            -------Class--------
Action   Type   Name     Start            Service     Report
                                 DEFAULTS: OMVS____    OMVSREP
____  1  TN     FTPSERVE ___              EBIZ_HI     FTPSERV
____  1  UI     OMVSKERN ___              SYSSTC      ________
____  1  TN     WSSRV*   ___              EBIZ_HI     WAS70
```

What is new in V7 for z/OS

WebSphere Application Server V7 in general offers some new concepts, functions, and features. The following are new additions are specific to WebSphere Application Server V7.0 for z/OS:

- z/OS Fast Response Cache Accelerator (FRCA)
- Thread hang recovery
- SMF 120 subtype 9

In this section, we discuss each of these new features in more detail.

z/OS Fast Response Cache Accelerator (FRCA)

WebSphere Application Server for z/OS V7 can be configured to use the Fast Response Cache Accelerator facility of the z/OS Communications Server TCP/IP. FRCA has been used for years inside the IBM HTTP Server to cache static contents like pictures or HTML files.

The high speed-cache can now be used to cache static and dynamic contents, such as servlets and JavaServer™ Pages (JSP™) files, instead of using the WebSphere Application Server Dynamic Cache.

Figure 14 on page 36 shows the changed flow of a request for a JSP that can be answered from the cache, assuming that the IBM HTTP server also resides on z/OS:

- Without FRCA exploitation, a request has to be processed by TCP/IP, then by the IBM HTTP Server on z/OS until WebSphere Application Server itself can answer the request from its dynacache.
- With FRCA exploitation, a request to a cached JSP is recognized in the TCP/IP processing and gets answered directly.

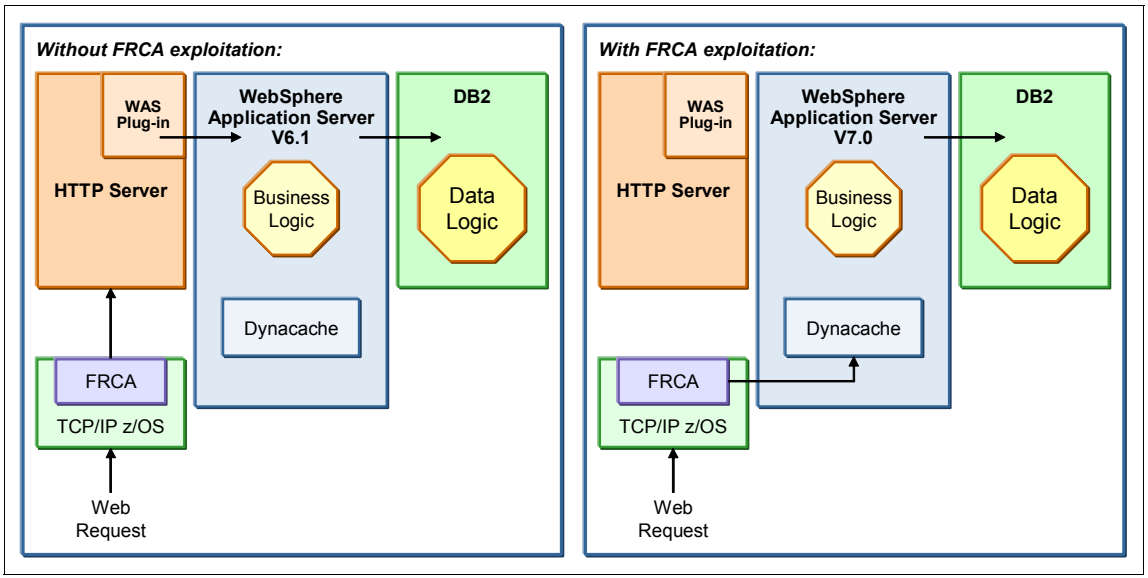

Figure 14 Overview of z/OS FRCA and WebSphere Application Server for z/OS

The benefits of using the FRCA are a reduced response time and a reduced CPU cost for the serving of requests, compared to the dynamic cache. Tests have shown that a request served from the FRCA uses approximately 8% of the processor time that the same request consumed in a dynamic cache environment uses. These advantages come from its structure, because the FRCA cache can directly serve incoming TCP/IP requests.

Attention: This functionality needs z/OS 1.9 or higher to be used.

In order for FRCA to work properly, the fix for APAR PK72551 (UK42691) has to be applied to the Communications Server TCP/IP on z/OS Version 1.9. If this fix is not applied, the server will issue error message BBOO0347E or BBOO0348E. TCP/IP utilizes CSM storage to maintain the cache.

To use FRCA in an application server with 31-bit mode, the fix for APAR PK80838 must be applied. This fix is planned for inclusion in service level 7.0.0.3.

Restriction: Currently FRCA cache is only supported for non-SSL connections.

Configuring FRCA

FRCA support needs to be configured in the administrative console as an external cache group and in `cachespec.xml`. This XML file should exist for each application in the corresponding `WEB-INF` directory of that application.

To configure FRCA, the following major steps need to be performed on an application server basis. To enable FRCA for a complete cell, it can be enabled on a server-by-server basis using wsadmin scripts:

1. Create an external cache group.
2. Populate the cache group with your server.
3. Modify the cachespec.xml.
4. [Optional] configure logging for the cache.
5. [Optional] If objects larger than 10MB are used, modify user variable `protocol_http_large_data_response_buffer` to a size larger than the largest object to be cached.

Best-practice: We recommend that you configure the dynamic cache disk offload. This will prevent objects from being removed from the dynamic cache and hence being removed from the FRCA cache. Refer to the "*Configuring dynamic cache disk offload"* article in the information center for further information, found at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.express.doc/info/exp/ae/tdyn_diskoffload.html

Create external cache group

Follow these steps to create the cache group:

1. Click **Servers** → **Server Types** → **WebSphere application servers.**
2. Click on the server name that should benefit from the FRCA to open the configuration page.
3. Select **Container Services** → **Dynamic cache service.**
4. Select **External cache groups** under the Additional Properties section.
5. Click **New.**
6. Enter a name for the external cache group (`frca` in Figure 15 on page 38), select **OK**, and save the changes to the master configuration. This name must be defined in the corresponding `cachespec.xml` file.

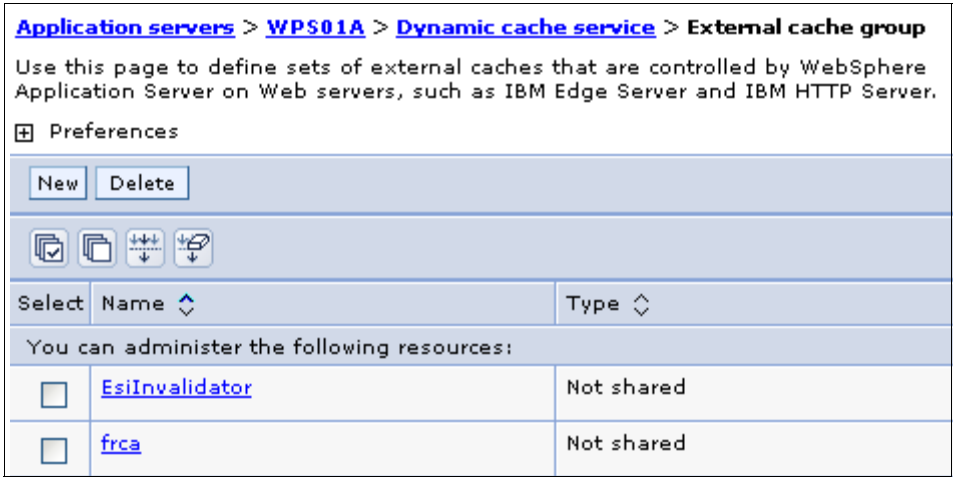

Figure 15 New external cache group

Add a member to the cache group

In this step, a new member has to be defined to the external cache group:

1. Open the new cache group by clicking on the name in the list of external cache groups (Figure 15).
2. Click on **External cache group members** in the Additional Properties section.
3. Select **New** to open the configuration page (Figure 16 on page 39).

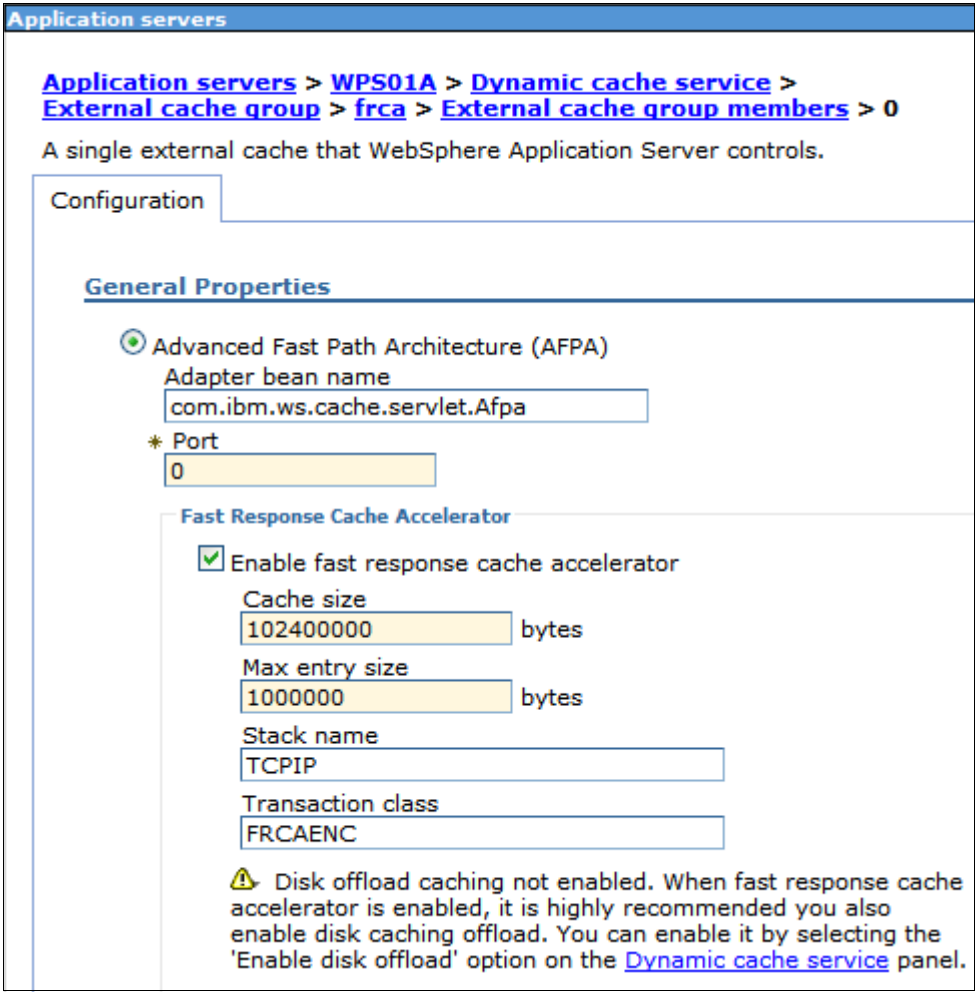

Figure 16 Enable FRCA cache

4. In the configuration tab, enter:
   - Select **Advanced Fast Path Architecture (AFPA).**
   - Check the **Enable fast response cache accelerator** option.
   - In the Port field, select 0 as the port number.

   For FRCA configuration, the following fields will need to be set as well:

   - **Cache size**: The cache size is a value that specifies the size of the FRCA cache. The maximum size is limited by the amount of available CSM memory managed by the z/OS Communications Server. The value is rounded up to a 4K(4096) interval. The default is 102400000.

- **Max entry size**: The max entry size value specifies the maximum size in bytes of a single object that can be placed in the FRCA cache. The default is 1,000,000.
- **Stack name**: The stack name specifies the name of the Open Edition Physical File system supporting the TCPIP stack containing the FRCA cache. The stack name specified must match the name on the `SubFileSysType` statement in the Open Edition `BPXPRMxx` parmlib member. (This directive is only needed if the Open Edition Common Inet function is being used. Contact your system programmer to determine if Common Inet is in use, and if so, the name of the FRCA-enabled TCPIP stack.) The default is none.
- **Transaction Class**: The transaction class name, which is eight characters or less, specifies the transaction class name that is used to classify the work done by FRCA. If the transaction class is specified, the FRCA processing is classified under WLM. If it is not specified, no classification will occur. The default is none.

  Usually the TCPIP address space runs under a high WLM priority such as SYSSTC within z/OS. Because the FRCA cache is physically located in TCPIP address space, the FRCA processing runs under the same priority as the TCPIP address space, if no special CB transaction class for FRCA is defined with a lower service goal.

  If no FRCA specific transaction class with a lower service goal is defined, the FRCA processing can theoretical cause the TCPIP address space to reject any additional connections. This can happen if especially large FRCA cached objects are invoked in a high frequency so that it consumes all available CPU resources. That the reason why we recommend to define this additional transaction class.

  Furthermore, a FRCA reporting class can be used to estimate the savings in CPU consumption compared to classical WebSphere workload without FRCA caching.

  Currently it is not possible to distinguish between different transaction classes within the FRCA workload as opposed to the classical WebSphere workload, where a workload classification can be utilized.

Note: By default, the FRCA cache is active on all channel chains that contain a Web container channel and do not contain an SSL channel.

You can disable FRCA for specific channel chains and listener ports, using the configuration tab for transport channels. Select **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Web Container Settings** → **Web container transport chains** → ***transport_chain***, and select the **Disable FRCA caching** option.

Update cachespec.xml

When updating the XML file, remember that the names used for the external cache in the XML file and the administrative console must match. Otherwise the cache cannot be used. A sample `cachespec.xml` is shown in Figure 17:

```
<property name="ExternalCache">frca</property>
```

An assembly tool is used to place the cachespec.xml file in the deployment module and to define the cacheable objects. You can also place a global cachespec.xml file in the application server properties directory.

For more information about the usage of a `cachespec.xml` refer to the WebSphere Application Server V7 Information Center, available at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rdyn_cachespec.html

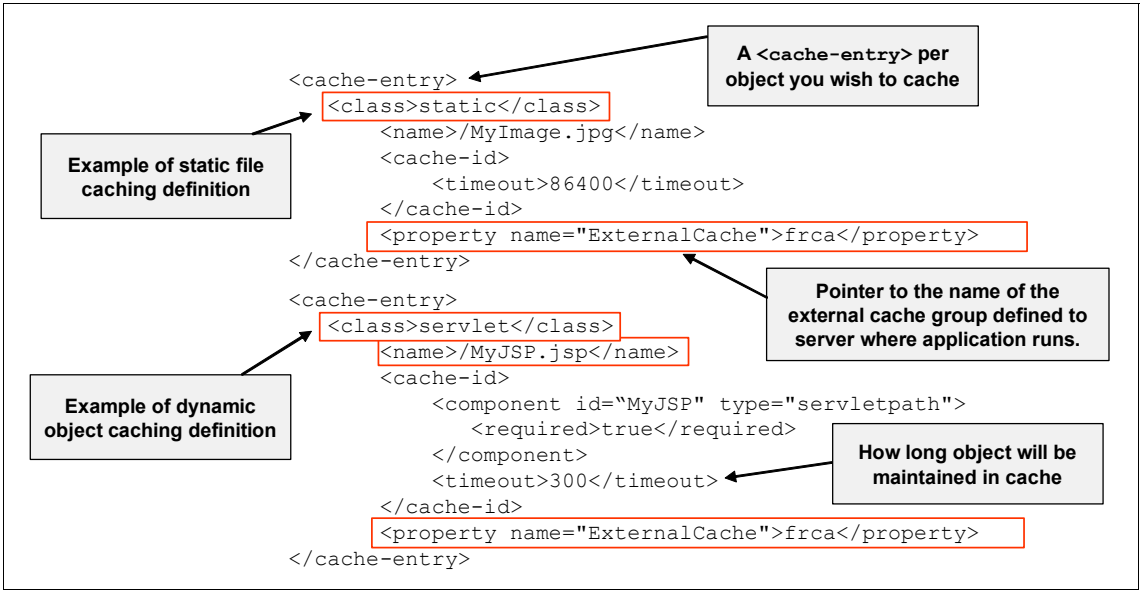

Figure 17 Sample cachespec.xml for usage with FRCA

[Optional] Enable FRCA logging

If you want to enable logging for FRCA, use the administrative console and execute the following steps:

1. Select **Servers** → **Server Types** → **WebSphere application servers** → ***server_name***.
2. Under Troubleshooting, select **NCSA access and HTTP error logging.**
3. Select the **Enable logging service at server start-up** option.

You can modify the other settings or keep the defaults.

Large object caching

If objects larger then 10MB should be cached, then you need to set the custom property `protocol_http_large_data_response_buffer`. The value for this property should be higher than the maximum size that should be cached.

For information about how to set custom properties, refer to the IBM Information Center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp

Monitoring FRCA

You can use the WebSphere Application Server modify display console command to display statistics and to list the contents of the FRCA cache. Refer to the *Configuring high-speed caching using FRCA with the WebSphere Application Server on z/OS* article in the Information Center for more detail:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/tdyn_httpserverz.html

To display cache statistics, the following commands can be issued:

- From z/OS console: `f <serverName>,display,frca`
- `f <serverName>,display,frca,content`
- From z/OS console: `display tcpip,,netstat,cach`
- From TSO: `netstat cach`

To monitor the activity of object caching in the dynamic cache, you can use the cache monitor. This installable Web application provides a real-time view of the dynamic cache state. In addition, it is the only way of manipulating data inside the cache. For more information about how to set up the cache monitor, go to the *Displaying cache information* article in the Information Center at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/tdyn_servletmonitor.html

Resource Access Control Facility (RACF) integration

FRCA services can be restricted. If access is restricted (the SERVAUTH class and the FRCA resource are defined) in your environment, then WebSphere Application Server must be granted access.

If the access is restricted, the message seen in Example 8 on page 43 will be issued.

Example 8 FRCA access denied message

```
BBOOnnnnE FRCA INITIALIZATION FAILED. SERVER NOT AUTHORIZED TO
USE FRCA SERVICES. IOCTL RV=%d, RC=%d, RSN=%08X
```

In order to use FRCA, the following RACF definitions are required:

- First, a new SERVAUTH profile needs to be defined with the corresponding system name and TCP/IP procedure name:

  ```
  RDEFINE SERVAUTH EZB.FRCAACCESS.<system_name>.<TCPIP_procname>
  UACC(NONE)
  ```

- Then, the application server control region must be given READ access to this SERVAUTH profile:

  ```
  PERMIT EZB.FRCAACCESS.<system_name>.<TCPIP_procname> CLASS
  (SERVAUTH) ID (UID_CR) ACCESS (READ)
  ```

- Finally, the SERVAUTH class has to be refreshed, to activate the changes:

  ```
  SETRPOTS RACLIST (SERVAUTH) REFRESH
  ```

Thread hang recovery

This section describes the new thread hang recovery option available on z/OS.

Overview

WebSphere Application Server for z/OS V7 contains a new technique called thread hang recovery. A hung thread will end up with one of the following situations:

- It simply hangs around, only blocking threads and application environment resources, such as connections, tables, and so forth.
- It ends in a loop state, not only blocking other resources but in addition consuming CP or IBM System z® Application Assist Processors (zAAP) resources (what kind of processor is being used depends on whether a zAAP is available at all and in what step of the application the error occurs).

Thread hang recovery directly addresses both of these issues. First, it allows you to define actions that should be started if a timeout occurs. It allows you to specify thresholds for processor usage and actions that should be performed if a single request exceeds this value. This is of real value if your environment uses high timeout values, due to some long running transactions, but with few processor resources per request. If such a transaction would suddenly consume a high amount of CPU due to an error, this situation would not be detected by prior versions unless the normal timeout occurs. Until the timeout occurs, there is a performance impact to the whole environment.

Pre-V7 technique

In releases prior to V7, when a request times out, the server assumes that the request is hung and will start to resolve the situation. Depending on the recovery setting for your installation, the server has two choices of processing:

- Terminate the servant with ABEND EC3:

  If `protocol_http_timeout_output_recovery=SERVANT`, then the servant will be terminated and WLM will restart a new one. All active threads in that particular servant are lost. A multi-servant architecture in this case can prevent total outages, because new work requests can be dispatched to another servant. A dump of the servant can be generated and all work that was running in the servant is terminated. This option could end up penalizing work that was not having any problems. In addition, server throughput is affected while the a dump is being taken and a new servant is started, which can take a long time. Figure 18 illustrates a hanging thread in the servant region.

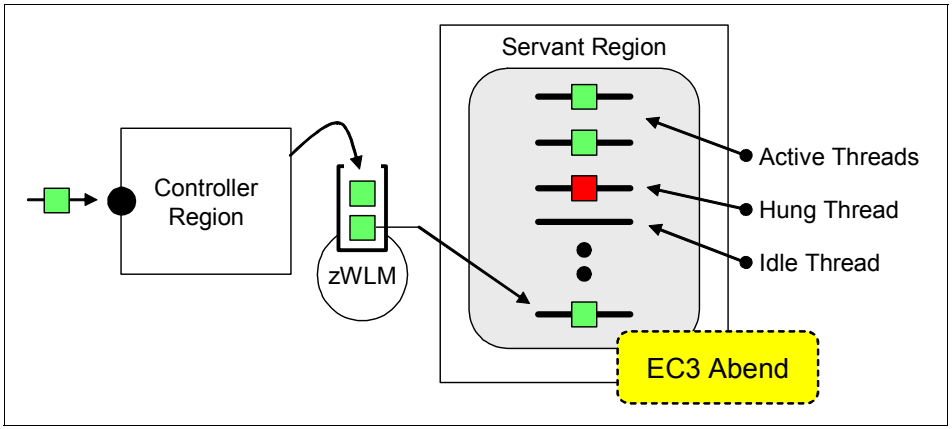

Figure 18 EC3 Abend caused by a hung thread prior to V7

- Respond to the client and continue working:

  If `protocol_http_timeout_output_recovery=SESSION`, then it is assumed that there was an unusual event that caused the timeout and that the request will eventually complete successfully. If this assumption is wrong, and the request is truly hung, the servant is left with one less thread for processing incoming work. In addition, by allowing the request to continue, deadlocks could occur if the request is holding locks or other resources. If this problem continues to happen on subsequent requests, multiple threads become tied up and the throughput of the servant is affected; possibly to the point where it has no threads available to process work.

V7 technique

In V7, if a hung thread is detected, then the servant can try to interrupt the thread (shake it loose). To allow the servant to do so, a new registry of *interruptible objects* is introduced. Certain blocking code can register so that if too much time passes, the servant can call the interruptible object in order for it to attempt to unblock the thread. A Java interruptible object will always be registered so the servant will try to have Java help interrupt the thread if all else fails.

> **Note:** The code that is used to unblock a thread is provided by WebSphere Application Server V7. You do not have to implement code for the Interruptable Objects registry.

The results of this situation can be:

- The thread can be freed:

  The user whose request is hung receives an exception. The administrator can define what dump action should be taken (none, svcdump, javacore or traceback).

- The thread cannot be freed:

  If the thread cannot be freed, the system action depends on the administrator settings. Basically the options are:

  – Abend the servant.

  – Keep the servant up and running.

  – Take a dump (defined by new variables, see Table 1 on page 47)

Although the basic options if a thread cannot be freed are still the same as in prior versions of the WebSphere Application Server for z/OS product, the decision on whether a servant should be abended or kept alive now depends on:

- How much CPU time is consumed by the thread? (Looping or really just hanging?)
- How many threads are already in a hung state, within this servant?
- Is the servant the last servant?

The recovery process of a hanging thread is illustrated in Figure 19.

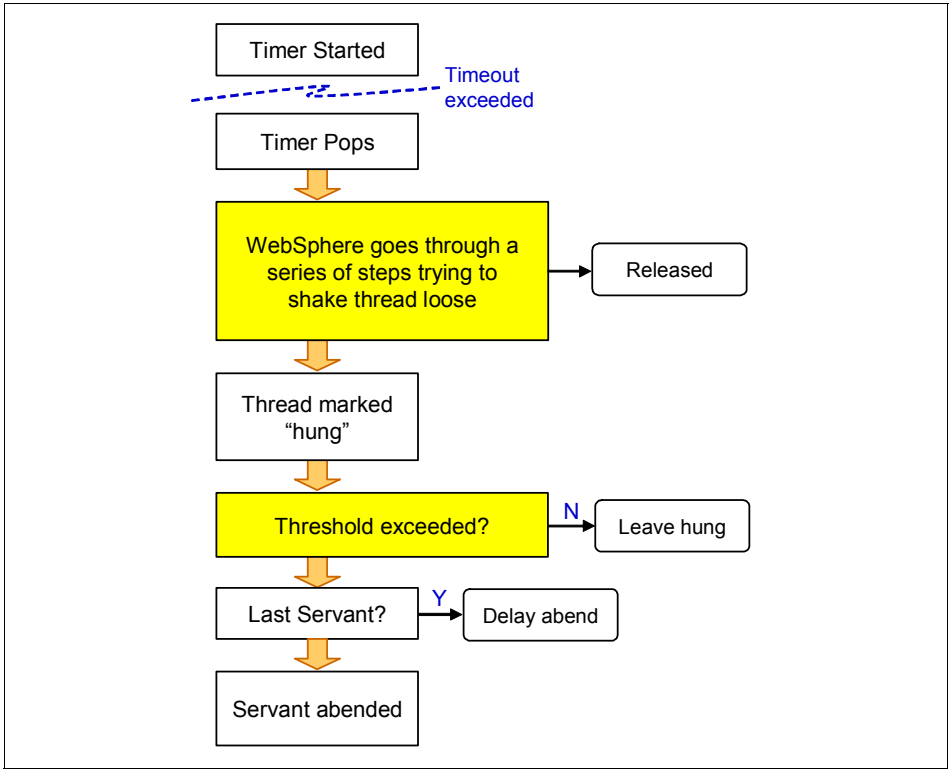

Figure 19 Hanging thread recovery process

With a new environment variable `server_region_stalled_thread_threshold_percent`, a percentage of threads that can be marked as "hung" can be specified, before the control region terminates the servant.

New properties

V7 introduces a set of new variables that allow the administrator to configure the behavior of the application server if a hung thread cannot be freed.

The new properties will be listed by default in the system log, although they must first be created as Custom properties to benefit from them. To create the custom properties, log on to the administrative console and navigate to **Server** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Server Infrastructure** → **Administration** → **Custom properties** → **New**. The variables are listed in Table 1.

Note: When modifying one of the new parameters, make sure to have these additional parameters configured (available in prior versions):

- control_region_timeout_delay
- control_region_timeout_save_last_servant

Table 1 Variables for hung thread related actions

Variable name	Values	Meaning
server_region_stalled_thread_threshold_percent	0 -100	This variable specifies the percentage of threads that can be marked as “hung” before the controller terminates the servant. Default value of 0 means that it behaves as in prior versions.
server_region_<type>_stalled_thread_dump_action	NONE | SVCDUMP | JAVACORE | JAVATDUMP | HEAPDUMP | TRACEBACK	Specifies the dump action after a stalled thread cannot be interrupted. <type> is the access method, which can be specified as http(s), iiop, mdb, or sip(s).
server_region_request_cputimeused_limit	variable	Amount of processor milliseconds a request can consume before servant will take an action.
server_region_cputimeused_dump_action	NONE | SVCDUMP | JAVACORE | JAVATDUMP | HEAPDUMP | TRACEBACK	Type of documentation to take when a request has consumed too much processor time.
control_region_timeout_save_last_servant	0 | 1	Indicates whether the last available servant should be abended if a timeout situation occurs on that servant, or the last available servant should remain active until a new servant is initialized.

Note: If the request exceeds the specified amount of time, UNIX Systems Services generates a signal that might or might not get delivered to the rogue request.

The signal might not get delivered immediately if the thread has invoked a native PC routine, for instance. In that case, the signal will not get delivered until the PC routine returns to the thread. When and if the signal gets delivered, a `BB000327` message is output, documentation is gathered according to what is specified as documentation action in the `server_region_cputimeused_dump_action` property, and the controller is notified that a thread is hung.

After the signal is delivered on the dispatch thread, the WLM enclave that is associated with the dispatched request is quiesced. This situation lowers the dispatch priority of this request, and this request should now only get CPU resources when the system is experiencing a light work load.

Display command

There is a new command to display the dispatch threads that are currently active. The `DISPLAY,THREADS` command will display every dispatch thread in every servant region associated with the specified controller. By default, it will give you `SUMMARY` information but you can also specify that you want `DETAILS`. In the case of the `REQUEST=<value>` parameter, the default is `DETAILS`.

```
f <server_name>,display,threads,[ALL | TIMEDOUT | REQUEST=<value> |
ASID=<value> | AGE=<value>]
```

See Figure 20 for an illustration of this situation.

F <server>,DISPLAY,THREADS,ALL

One servant example:

```
BBOJ0111I:  REQUEST ASID JW TO RE DISPATCH TIME
BBOJ0112I: ffffb35f 0176  N  N  N 2009/03/26 18:21:17.423648
BBOJ0112I: ffffb360 0176  Y  N  N 2009/03/26 18:21:15.569834
BBOJ0112I: ffffb361 0176  Y  Y  N 2009/03/26 18:21:12.790693
BBOJ0112I: ffffb362 0176  Y  Y  Y 2009/03/26 18:21:10.720461
BBOO0188I END OF OUTPUT FOR COMMAND DISPLAY,THREADS,ALL
```

- **Healthy thread**
- **In a Java Wait but not yet Timed Out**
- **Java Wait and Timed Out, but ITI has not yet marked it "hung"**
- **JW, TO and retried -- Hung**

In this example Threshold > 0 otherwise servant would be recycled because one thread is hung

Figure 20 Output of display thread modify command

The information is also available via a new `InterruptibleThreadInfrastructure` MBean.

More information about the thread hang recovery and dispatch timeouts is provided in the white paper, *WebSphere Application Server for z/OS V7 - Dispatch Timeout Improvements*, available at:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101374

Systems Management Facility (SMF) Subtype 9

WebSphere Application Server for z/OS is capable of writing z/OS Systems Management Facility (SMF) records for performance tuning and charge-back.

Prior to WebSphere Application Server for z/OS V7

Since Version 4 of WebSphere Application Server, SMF record 120 has been used to log WebSphere usage data. Record 120 includes a number of subtypes, each of which contains a subset of the data. This fragmented view of the data is due to internal divisions in the product. Some record 120 subtypes are created by the WebSphere Application Server runtime (subtypes 1 and 3), while others are created by the Web containers and EJB containers (subtypes 5, 6, 7, 8). Because each subtype provides only a partial view of the activity, you need to correlate several subtypes to get a more complete picture of what the server is doing. This will, however, increase the overhead of SMF usage.

New in WebSphere Application Server for z/OS V7

In V7 a new subtype 9 record has been added. The name is *Request Activity Record* and it can be used to create/write resource usage records without unreasonable overhead. Any data collection that adds substantially to the cost of acquiring that data is optional. Additionally you can activate or deactivate this record dynamically.

Recommendation: Because the new SMF 120-9 subrecord consumes less processing time than the collection of multiple subrecords, we recommend the usage of the new subrecord. The old subrecords remain valid and can be used with WebSphere Application Server for z/OS V7.

The subtype 9 gives you the option to monitor which requests are associated with which applications, how many requests occur, and how much resource each request uses. You can also use this record to identify the application involved, and the CPU time that the request consumes. Because a new record is created for each request, it is possible to determine the number of requests that arrive at the system over a specific period of time.

Table 2 shows the new variables, possible values and their meaning. A browser to display the contents of the new SMF records is provided.

Table 2 New SMF record 120-9 variables

Variable name	Values	Meaning
server_SMF_request_activity_enabled	0 | 1	Turn record off/ on
server_SMF_request_activity_CPU_detail	0 | 1	Processor usage details
server_SMF_request_activity_timestamps	0 | 1	Formatted timestamps
server_SMF_request_activity_security	0 | 1	Security information

Modify commands

The following modify commands can be used to activate/ disable the new variables:

- `F <server>,SMF,REQUEST,[ON | OFF]`
- `F <server>,SMF,REQUEST,CPU,[ON | OFF]`
- `F <server>,SMF,REQUEST,TIMESTAMPS,[ON | OFF]`
- `F <server>,SMF,REQUEST,SECURITY,[ON | OFF]`

To show the current settings in your environment, as well as the number of records written since the last *Initial Program Load* (IPL), and the last non-zero *Return Code* (RC), use the command:

```
F <server>,DISPLAY,SMF
```

For more information about the SMF enhancements refer to White paper *Understanding SMF Record Type 120, Subtype 9*, available at:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101342

Thread management using the workload profile

The workload profile setting defines the number of available application threads for each servant. This setting applies only to worker threads which are managed by the application server.

To change the value via the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Container services** → **ORB service** → **z/OS additional settings**.

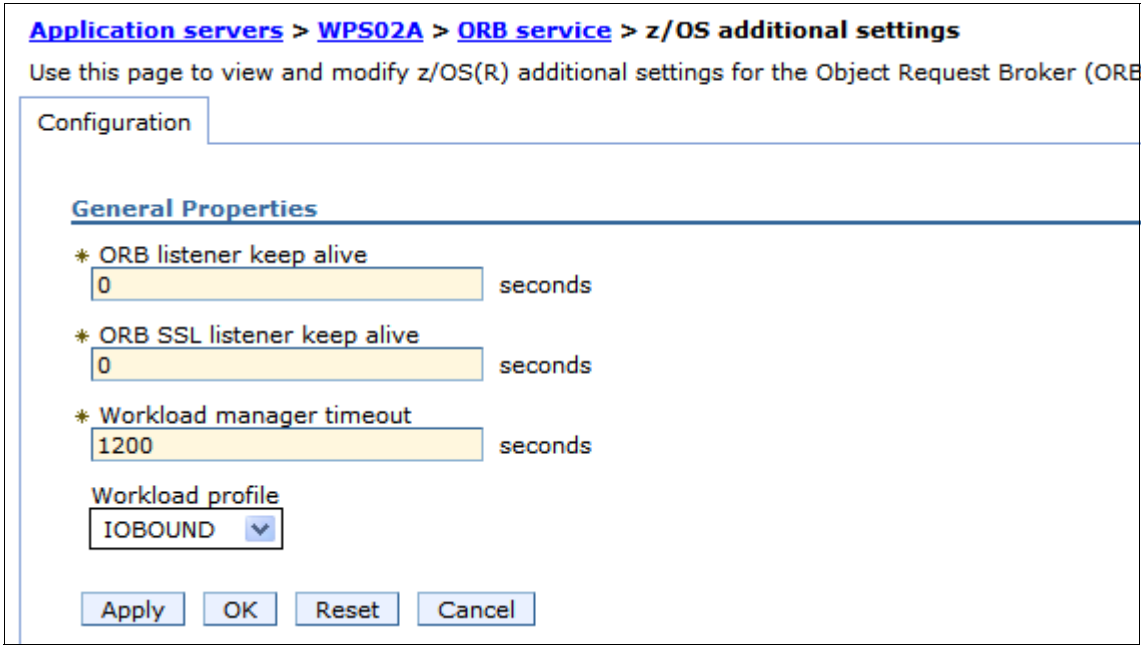

Figure 21 Thread Management - Workload profile

The optimal number of parallel threads depends on the number of available CPU resources and on the workload characteristics of the application. We recommend that you use the Tivoli® Performance Viewer in the administrative console to determine the number of parallel threads that are currently occupied by a representative production workload. In most customer cases, the IOBOUND profile is selected.

WebSphere Application Server for z/OS V7 introduces a new value for the workload profile setting in the Object Request Broker (ORB) services advanced settings. It is now possible to make a user defined selection for the number of threads, using the CUSTOM setting.

Table 3 explains all possible workload profile settings.

Table 3 Workload Profile settings for z/OS ORB service

Value	# Threads	Description
ISOLATE	1	Specifies that the servants are restricted to a single application thread. Use ISOLATE to ensure that concurrently dispatched applications do not run in the same servant. Two requests processed in the same servant could cause one request to corrupt another.
IOBOUND	MIN(30, MAX(5,(Number of CPUs*3)))	Specifies more threads in applications that perform I/O-intensive processing on the z/OS operating system. The calculation of the thread number is based on the number of CPUs. IOBOUND is used by most applications that have a balance of CPU intensive and remote operation calls. A gateway or protocol converter are two examples of applications that use the IOBOUND profile.
CPUBOUND	MAX((Number of CPUs-1),3)	Specifies that the application performs processor-intensive operations on the z/OS operating system, and therefore would not benefit from more threads than the number of CPUs. The calculation of the thread number is based on the number of CPUs. Use the CPUBOUND profile setting in CPU intensive applications, like XML parsing and XML document construction, where the vast majority of the application response time is spent using the CPU.
LONGWAIT	40	Specifies more threads than IOBOUND for application processing. LONGWAIT spends most of its time waiting for network or remote operations to complete. Use this setting when the application makes frequent calls to another application system, like CICS screen scraper applications, but does not do much of its own processing.
CUSTOM	User defined	**This option is new in V7!** Specifies that the number of servant application threads is determined by the value that is specified for the `servant_region_custom_thread_count` server custom property. The minimum number of application threads that can be defined for this custom property is 1; the maximum number of application threads that can be specified is 100.

Local connectivity to DB2

If the DB2 resides on the same system as WebSphere Application Server, we recommend that you use the JDBC™ type 2 driver. This driver establishes a cross-memory connection to DB2, as opposed to a JDBC type 4 connection, which is always a TCP/IP connection with the corresponding overhead.

Prerequisites for implementing a JDBC type 2 driver

The following prerequisites must be fulfilled in order to get the JDBC type 2 driver to work:

1. Ensure that the following DB2 load libraries are in the STEPLIB or LNKLST concatenation of the application server control region and servant region:
   a. SDSNEXIT: Depending on your individual DB2 setup, the SDSNEXIT module should be included.
   b. SDSNLOAD
   c. SDSNLOD2

   See Example 2 on page 17.
2. When using the JDBC type 2 driver, WebSphere Application Server uses the DB2 subsystem identifier provided in the DSNHDECP load module. If another DB2 subsystem should be used, a db2.jcc.propertiesFile needs to be specified in the JVM custom properties, containing the subsystem's SSID. More information about this properties file, see:

   http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/tdat_jdbcdb2cfg.html
3. The JDBC driver must first be bound with the DB2 packages that represent the SQL statements to be executed. The specific details of the bind utility and bind process are described by the README provided with the installed DB2 Universal JDBC Driver. Refer to this README for details on how to set up and perform the required binding. Example 9 shows the necessary OMVS commands for the binding process.

Example 9 JDBC Driver binding

```
export PATH=/usr/lpp/db2/d8xg/jcc3/bin:$PATH
export
CLASSPATH=/usr/lpp/db2/d8xg/jcc3/classes/db2jcc.jar:/usr/lpp/db2/d8xg/j
cc3/classes/db2jcc_javax.jar:/usr/lpp/db2/d8xg/jcc3/classes/db2jcc_lice
nse_cisuz.jar:/usr/lpp/db2/d8xg/jcc3/classes/sqlj.zip
```

```
java com.ibm.db2.jcc.DB2Binder -url
jdbc:db2://<host_name>:<tcpport>/<location_name> -user <sysadm>
-password xxxxx
```

We recommend JDBC Driver V3.51 or higher, because the multi-row fetch feature has been introduced which can significantly improve the performance of a type 2 connection.

The JDBC Driver (JCC) V3.51+ driver is available to both DB2 for z/OS Version 8 and Version 9 customers. The initial deliveries of JCC 3.51 for DB2 z/OS were made under PK63584 for DB2 z/OS V8 and PK68428 for DB2 z/OS V9. These APARs are available for immediate customer install and include all of the following major JDBC type-2 for z/OS features/enhancements:

- Ability to concurrently connect to multiple, local DB2 subsystems
- Type 2 failover support using DB2 z/OS Group Attach Names
- Support for multiple row INSERT using when using JDBC batching
- Support for multiple row FETCH

The following command can be used to check the installed version of the JDBC Driver:

```
java -cp <path_to_jdbc_classes>/db2jcc.jar com.ibm.db2.jcc.DB2Jcc
-configuration
```

WebSphere Application Server V7 is based on JDK™ 1.6, which includes the new JDBC 4.0 specification level. This specification is implemented in the JDBC Driver V4.3+ on z/OS. This version of the driver also supports the new multi-row fetch feature as with V3.51+ and is only delivered with DB2 for z/OS V9.

Creating a JDBC provider

To create a JDBC provider for type 2 connections:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources** → **JDBC** → **JDBC Providers**.
2. Select the scope at which applications can use the JDBC provider. This scope becomes the scope of any data source that you associate with this provider. You can choose a cell, node, cluster, or server.
3. Click **New** to start the wizard for creating a new JDBC provider.
4. Specify the following parameters:
   - **Database type:** DB2
   - **Provider type:** DB2 Universal JDBC Provider
   - **Implementation type:** Connection pool data source

If you use the Connection pool data source implementation type with type 2 connectivity, WebSphere Application Server on z/OS uses Resource Recovery Services (RRS) to process both one-phase and two-phase transactions. In contrast, if the application does not require that connections support two-phase commit transactions, and you plan to use type 4 connectivity, choose Connection Pool Data Source.

Choose XA Data Source if you plan to use driver type 4, and your application requires connections that support two-phase commit transactions. Use only driver type 4 connectivity for the XA data source.

– Give a name to the new JDBC provider.

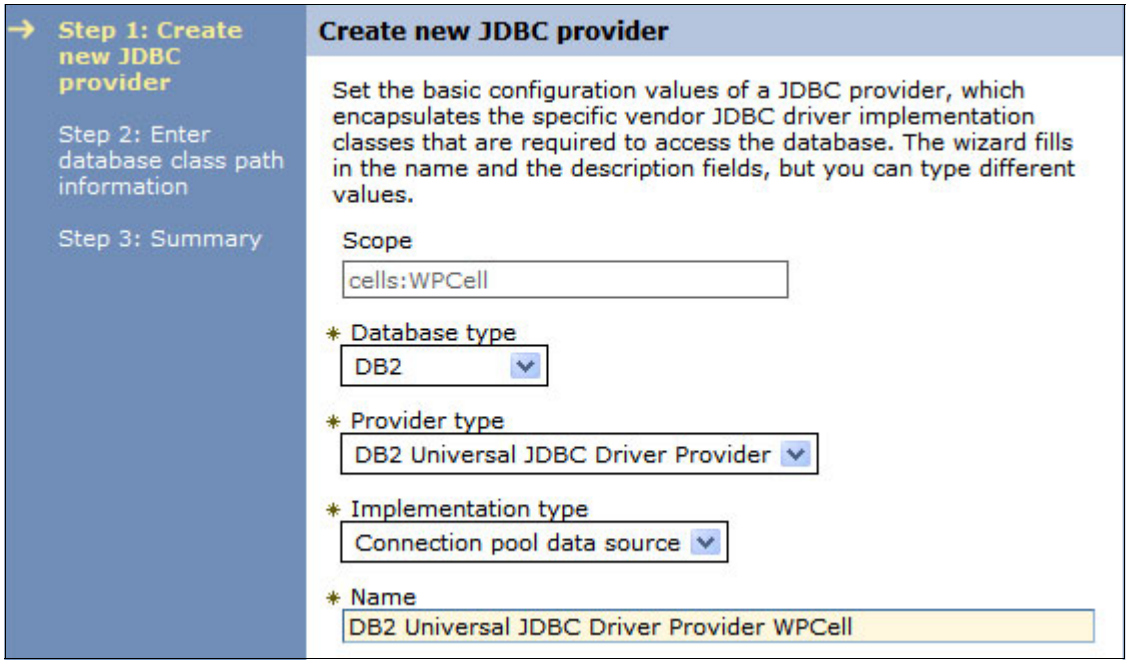

Figure 22 Create a new JDBC provider: Step 1

Click **Next** to continue.

5. In the next panel the JDBC class paths need to be specified as shown in Figure 23 on page 56.

   Here is an example of JDBC class paths:

   Class path:

   `/usr/lpp/db2/d8xg/jcc3/classes`

   Native library path:

   `/usr/lpp/db2/d8xg/jcc3/lib`

The native library path is the path, where the .so files are located.

Both paths `DB2UNIVERSAL_JDBC_DRIVER_PATH` and `DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH` will be saved as WebSphere environment variables with the same scope as the JDBC provider. The JDBC class paths can be adjusted afterwards by changing the value of these variables in **Environment** → **WebSphere variables** in the administrative console.

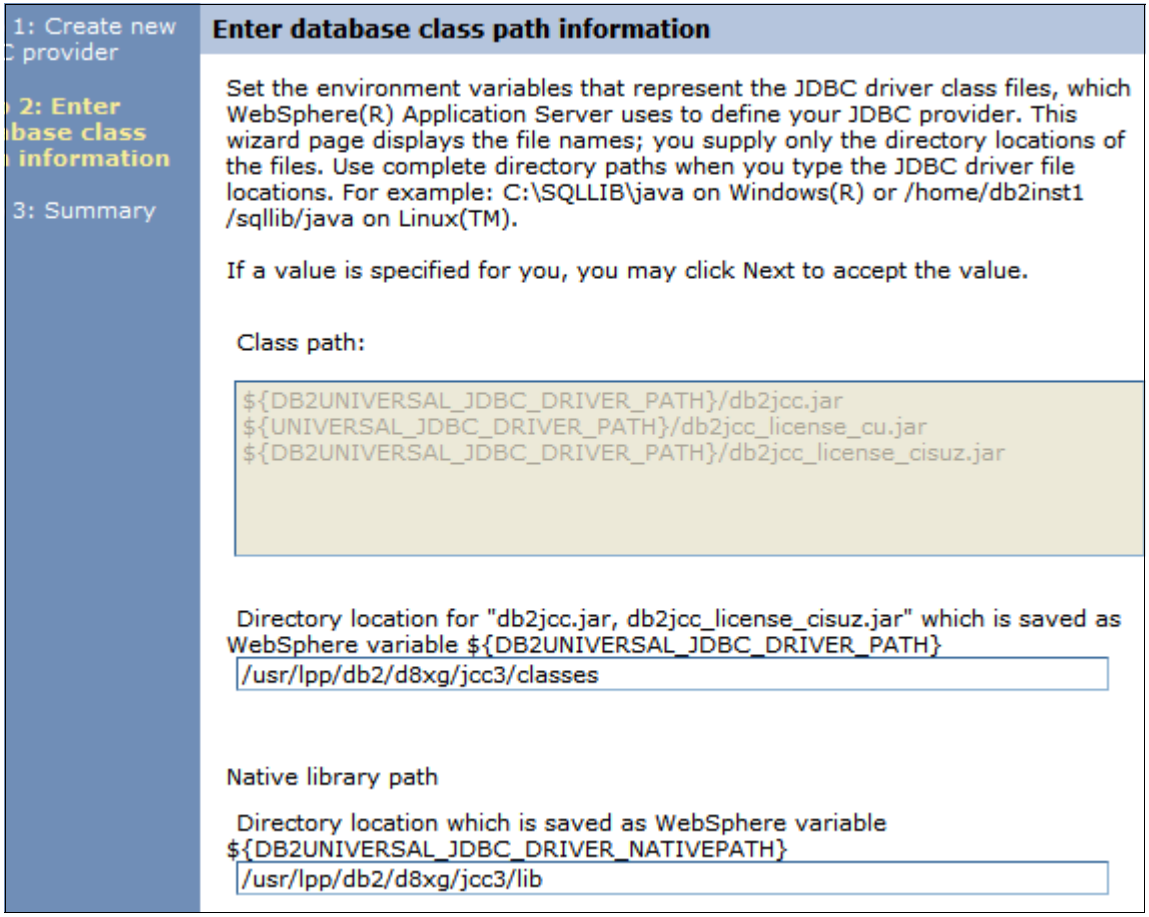

Figure 23 Database class path information

Click “**Next**” to continue.

6. Finally a summary of the JDBC provider configuration will be displayed. Click **Finish** to complete the wizard.

Defining a type 2 data source

The next step is to define the data source.

1. Click the JDBC provider, which has been just created.
2. In the right navigation bar of the JDBC provider configuration, click “**Data sources**” under Additional Properties.
3. Click the **New** button to create a new data source.

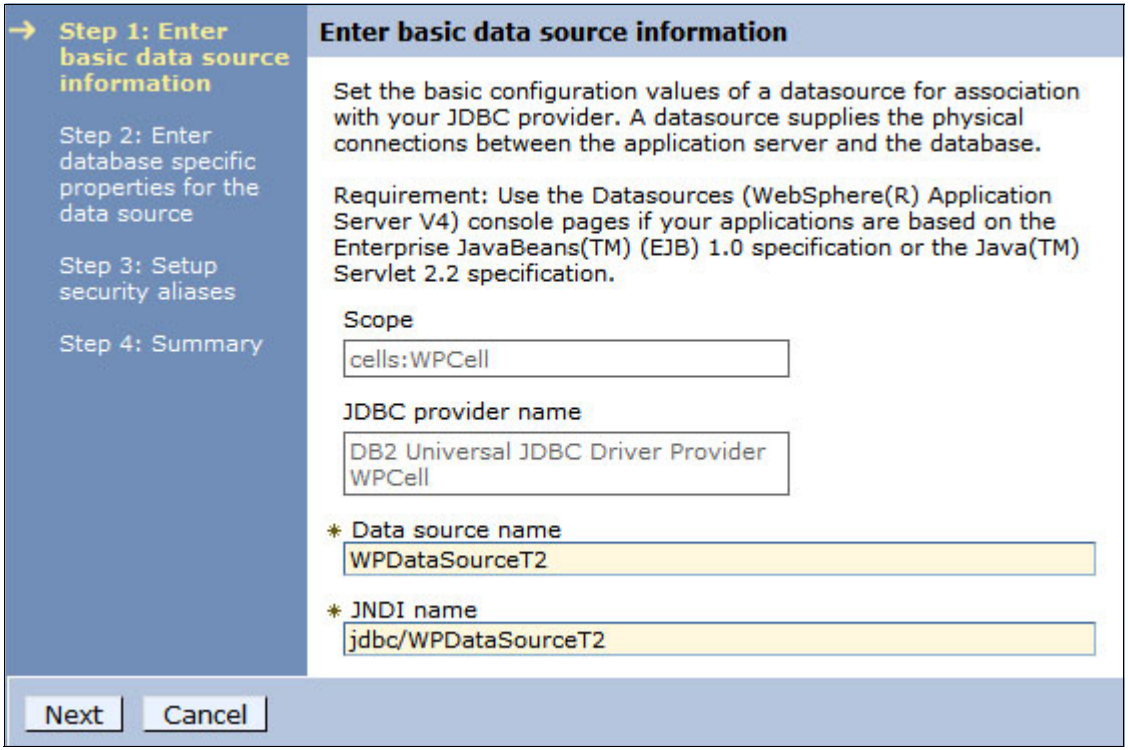

Figure 24 Create a new data source: Step 1

Specify the data source name and the corresponding JNDI name. Usually the prefix “**jdbc/**” is used for data sources.

Click **Next** to continue.

4. In the next panel the database specific properties of the data source can be specified. Here is an example:
   - **Driver type:** 2
   - **Database name:** DB8X (database location name)
   - **Server name:** wtsc04.itso.ibm.com (host name)
   - **Port number:** 33760

A DISPLAY DDF command can be used to find the database location name, host name, and port number, as shown in Example 10.

Example 10 Display DDF command

```
-D8X1 DIS DDF
RESPONSE=SC04
 DSNL080I  -D8X1 DSNLTDDF DISPLAY DDF REPORT FOLLOWS:
 DSNL081I STATUS=STARTD
 DSNL082I LOCATION           LUNAME            GENERICLU
 DSNL083I DB8X               DEIBMIPA.IPAADB21 -NONE
 DSNL084I IPADDR          TCPPORT RESPORT
 DSNL085I 9.12.4.20       33760   33761
 DSNL086I SQL    DOMAIN=wtsc04.itso.ibm.com
 DSNL086I RESYNC DOMAIN=wtsc04.itso.ibm.com
 DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

For the JDBC type 2 driver, the server name and port number are not required, but they can be specified in order to easily switch between type 2 and type 4 driver (Figure 25).

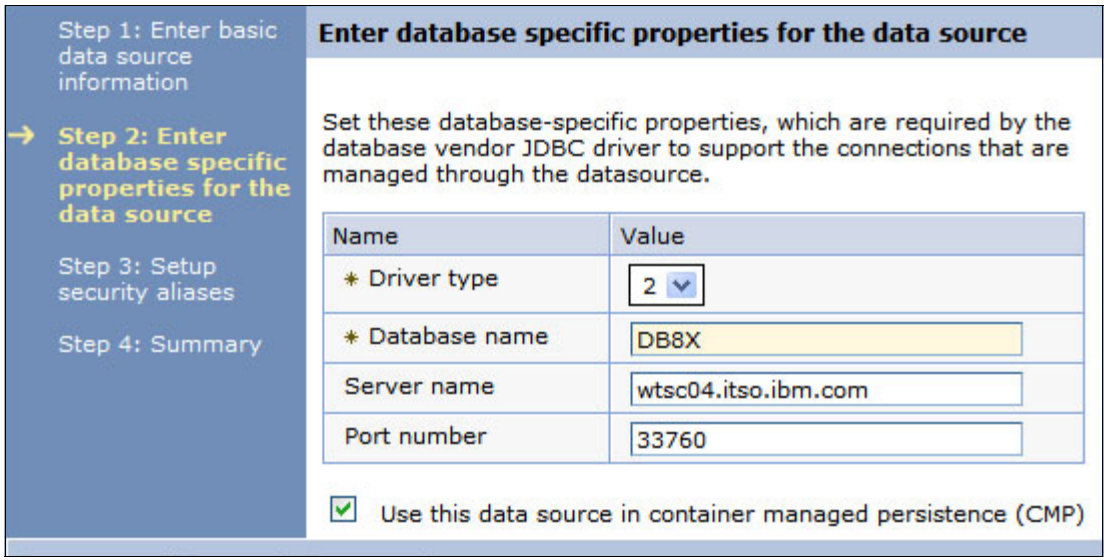

Figure 25 Data source properties

Click "**Next**" to continue.

5. Finally a summary of the data source configuration will be displayed. Click "**Finish**" to complete the wizard.
6. Click **Save and synch changes with nodes**.

A restart of the corresponding application servers is necessary in order to use the JDBC type 2 driver.

Migrating to V7

If you plan to migrate from previous versions of WebSphere Application Server for z/OS to version 7.0, the white paper, *Migrating to WebSphere z/OS V7*, provides all necessary information to successfully migrate to V7. This document is available at:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101329

A lot of effort has been expended to improve the migration process over that found in previous versions.

Notices

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

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

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

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

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

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

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

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

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

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

This document REDP-4575-00 was created or updated on October 13, 2009.

Send us your comments in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbook@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099, 2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks