AS Java Architecture Manual

SAP NetWeaver Composition Environment 7.1

Applies to

SAP NetWeaver Composition Environment 7.1

Summary

This document provides an overview of the architecture of the Application Server Java (AS Java) in SAP NetWeaver Composition Environment 7.1. The guide describes the cluster concepts in AS Java, the system component architecture, and some of the main features that AS Java provides.

Document Version

1.0 – Initial Version (created on 02 June 2007)

Copyright

Icons in Body Text

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help* → *General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Typographic Conventions

Type Style	Description
Example text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation.
Example text	Emphasized words or phrases in body text, graphic titles, and table titles.
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, `F2` or `ENTER`.

Contents

Architecture Manual

This documentation provides an overview of the architecture of the Application Server Java (AS Java).

AS Java

AS Java is part of the SAP NetWeaver Application Platform. It provides the complete infrastructure for deploying and running Java applications.

AS Java is optimized to run mission-critical business applications and provides a number of new concepts and enhanced features, such as:

- Increased robustness and stability of the server infrastructure and applications
- Simplification of server infrastructure and improved supportability
- Advanced administration capabilities provided by the Zero Administration concept
- Enhanced monitoring and easy access to all monitoring data
- Integration of SAP Java Virtual Machine with additional capabilities
- Central cache management and session management

Generally, the Architecture Manual describes:

AS Java Cluster Architecture [Page 6] – a cluster is a building unit of the application server which includes all the components that actually enable user requests to be processed.

AS Java System Architecture [Page 11] – the logical layers of the AS Java system components and the relations among them.

AS Java Cluster Architecture

The AS Java cluster is a set of processes that work together to build a scalable and reliable system. The cluster architecture is transparent to the client and appears to it as a single server unit.

An AS Java cluster consists of several types of instances, all of which have an instance number and can be started, stopped and monitored separately. They are:

- **Central services instance**

  The central services instance consists of a Message Service and Enqueue Service. They are responsible for lock administration, message exchange and load balancing within the Java cluster.

- **One or more Java instances**

  A Java instance consists of an Internet Communication Manager (ICM) and one or several server processes. The ICM handles requests coming from clients and dispatches them to the available server processes which actually process the requests.

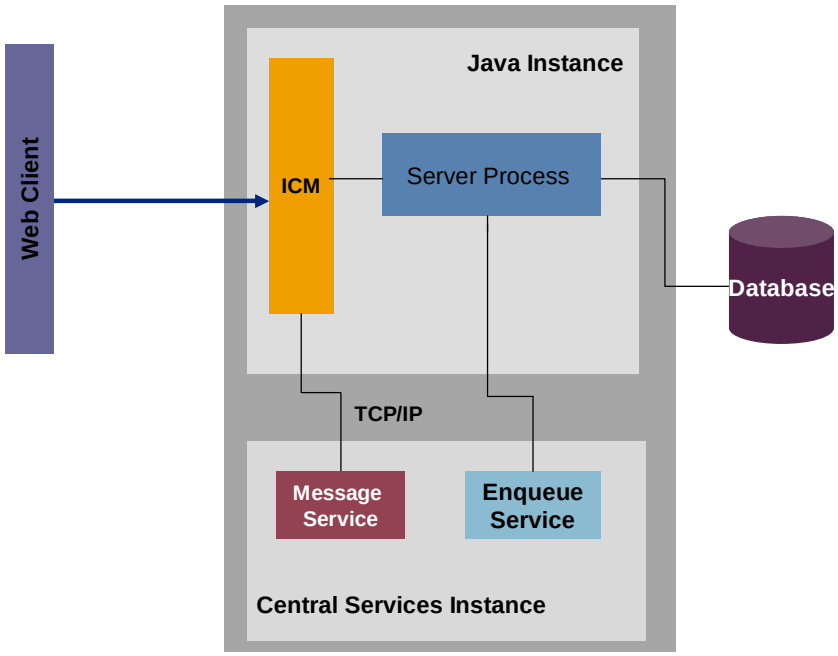
- **One or several databases**

  The database stores system and application data.

An appropriate cluster setup is one of the main prerequisites for the efficient performance of your system. You can scale the Java cluster by installing additional Java instances to your system, or by adding server processes to an already existing Java instance. For high availability reasons, the different instances can be split up among different physical machines.

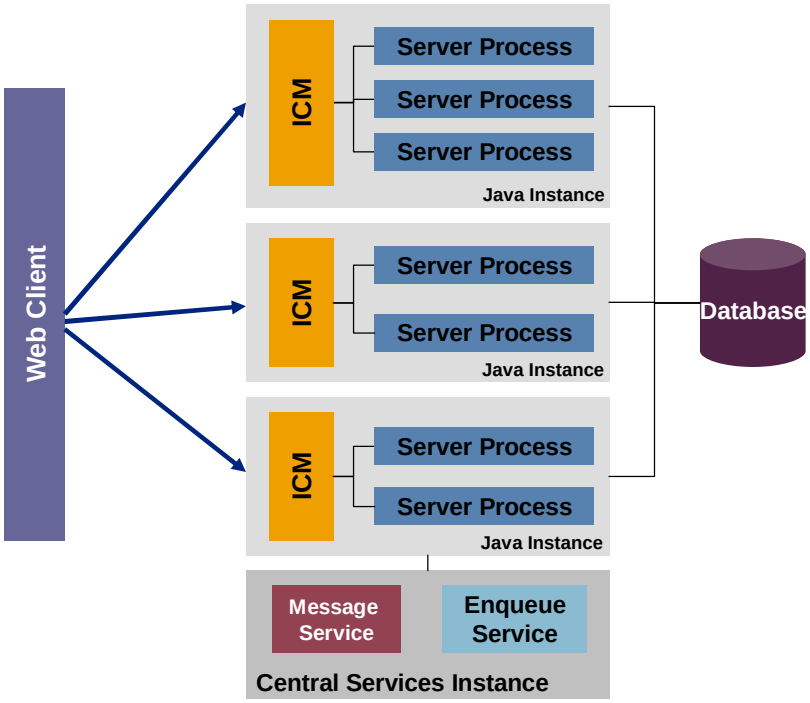
Minimum AS Java Cluster Installation

A minimum AS Java cluster installation consists of a central services instance, one Java instance with one server process, and a database.



Large AS Java Cluster Installation

A larger AS Java installation can have several Java instances with more than one server process each, a central services instance, and one or several databases.



More Information:

Central Services [Page 10]

Java Instance [Page 8]

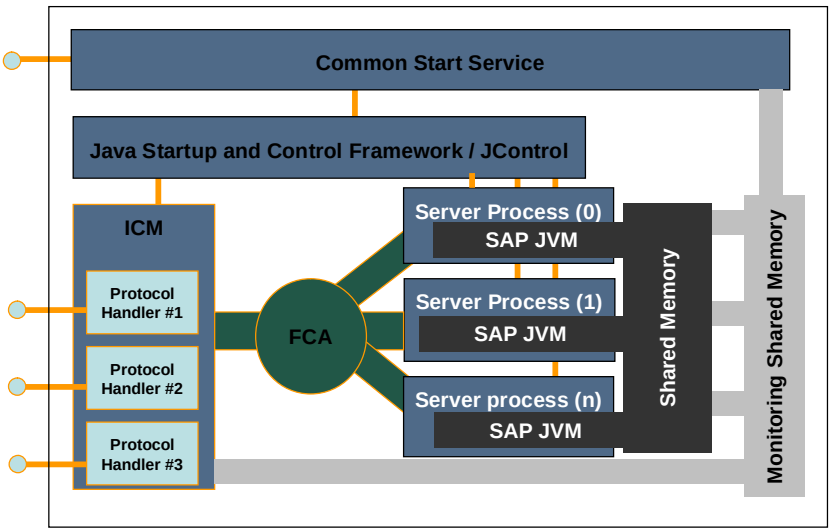
Java Instance

A Java instance is a unit in the AS Java cluster which is identified by its instance number. The elements that form an instance run on one physical machine. Also, it is possible to run several instances on one physical machine, but it is recommended that you split the different instances among different physical machines.

A Java Instance consists of:

- **Internet Communication Manager (ICM)**
- **One or several server processes**

Java Instance Architecture



Internet Communication Manager

The ICM is an element of the Java instance which handles requests coming from clients and dispatches them to the available server processes. Data is transferred from the ICM to the server processes and vice versa using the Fast Channel Architecture (FCA), which allows fast and reliable communication between them.

The ICM reads the request from the TCP/IP stack into the FCA, decides which server process should handle the request (load balancing), and then sends the requests directly to the individual process. The required information for load balancing is retrieved by the ICM from the Message Service.

When a server process has sufficient resources to consume a request, it takes it out of the FCA queue, processes it, and writes it back into the queue so it is returned to the originator of the request.

Server Process

The server processes of the AS Java actually execute the Java application. They are responsible for processing incoming requests which are assigned to them by the ICM.

Each server process is multi-threaded, and can therefore process a large number of requests simultaneously.

When more than one server processes run inside a Java instance, all of them have the same capabilities.

During installation, the installation procedure configures the optimal number of server processes in an instance based on the available hardware resources. You can add more server processes to an existing Java instance.

Server processes in an instance have a shared memory which enables much faster interaction. In the shared memory, server processes and the ICM store all their monitoring information, which can be used for detailed analysis of the current internal status of each Java instance.

All VMs in the instance have access to a shared memory area used as a session store, which is also a safeguard against VM failures. This is enabled by the use of SAP's own implementation of a Java Virtual Machine.

SAP Java Virtual Machine (SAP JVM)

AS Java uses SAP JVM as its runtime platform. The SAP JVM is based on the Hotspot Java VM provided by Sun Microsystems but it also possesses some additional features, such as:

- Memory analysis - easier detection of out-of-memory situations and analysis of memory footprint due to memory debugging features embedded into the VM.
- Robustness - due to fast session failover based on shared memory.

  The robustness concept is based on two main ideas: fewer active user requests per VM and a VM independent safe storage of inactive user sessions. More than one VM can run on each machine to reduce the amount of parallel processed user requests per VM. Inactive user sessions are separated from the VMs and stored in a shared memory region.

Central Services Instance

Central services form the basis of communication and synchronization for the AS Java cluster. They are responsible for lock administration, message exchange, and load balancing within the cluster.

Central services run on one physical machine and constitute a separate instance. They comprise:

- **Message Service**

  The Message Service keeps a list of all server processes in the AS Java cluster and provides information about their availability to the ICM. It also represents the infrastructure for data exchange between the participating server processes.

  The Message Service is responsible for the following tasks in the AS Java cluster:

  - Notification of events that arise in the cluster, for example, when a service is started or stopped
  - Communication between different services
  - Forwarding of messages and requests to all participants
  - Guaranteed message transmission
  - Exchange of cache information in the cluster

- **Enqueue Service**

  The Enqueue Service manages logical locks. It has the following tasks:

  - Internal synchronization within the AS Java cluster.
  - The applications can lock objects and release locks again. The Enqueue Service processes these requests and manages the lock table with the existing locks.

Integration

The central services are always required when an AS Java cluster is installed. The central services instance has its own instance number.

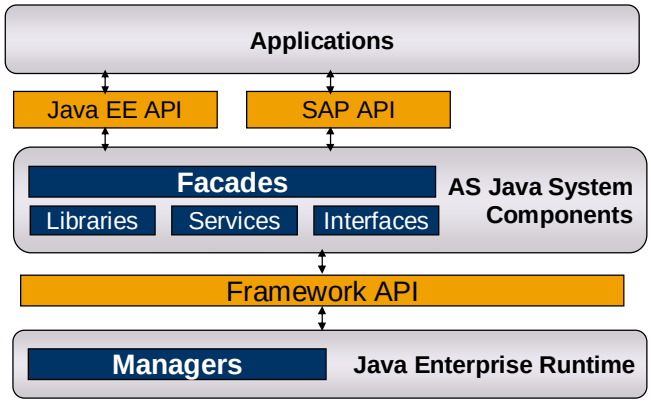
When central services are running, the other Java instances are started with the program JControl.

AS Java System Architecture

The AS Java system consists of three logical layers:

- **Java Enterprise Runtime** – comprises low-level subsystems that provide functions such as classloading, cluster communication, persistent configuration data management, and so on.
- **AS Java System Components** – consists of facades, interfaces, libraries and services components that provide various runtime functions and programming APIs.
- **Applications** – refers to the applications that are deployed and run on the AS Java.

AS Java System Architecture



Features

The AS Java system architecture is based on the following general rule: components from a higher level can use components from a lower level only through a set of defined APIs – facades; whereas components from a lower level are not aware of the APIs of the components from a higher level and therefore cannot use them.

This rule is reflected in the order of starting the system modules: the runtime is started first, then the services (the libraries are loaded, the interfaces resolved at this phase), and the applications are started last. The system is considered as started when all runtime managers and core services components are started properly.

The AS Java system components use the Framework APIs to connect to the Java enterprise runtime. Applications use the AS Java system components using the APIs that are defined by Java EE 5 specification (and supporting specifications) and SAP-proprietary APIs.

Java Enterprise Runtime

The Java enterprise runtime provides the core functions of the system. It is made up of several low-level subsystems containing manager components which provide the infrastructure and runtime to support the upper layer of the AS Java system components.

The Java enterprise runtime provides a number of key concepts which significantly enhance the robustness, stability and supportability of the AS Java, such as central cache management and session management.

Cluster Communication

AS Java cluster elements communicate via cluster messages. The system is able to send messages to one cluster element, to a group of elements, or to all elements in the cluster. Cluster elements use messages to broadcast notification events. The Cluster Manager is responsible for managing this communication.

There are several types of cluster communication depending on the abilities they provide for transferring messages. The type of cluster communication that is used when processing one message is determined by the Cluster Manager and is transparent for the cluster elements.

The successful message delivery is guaranteed, except in cases when the corresponding element stops or is timed out. In this case, the corresponding exception occurs and can be handled by the sending party. The ability of a server element to process a notification depends on the load level of that particular VM – if the load is too high, the server element will not be able to process the notification immediately.

Cache Management

The AS Java uses a central cache management to better control the memory consumption of the components' caches. It provides a complete cache framework - the Cache Management Library (CML) - which drastically reduces the overall memory footprint of the system. This lower memory footprint results in the capability of running more VMs.

Cache management has the following features:

- One cache implementation is used by all layers to reduce redundancies
- Central cache configuration for all system caches
- Advanced monitoring and administration of the whole cache landscape
- Optimizations and tuning of the cache can be done on a higher level and therefore more efficiently.
- Cluster-wide invalidation of cache entries is possible

Session Management

AS Java has a central session management which provides the following features:

- Enhanced control of the sessions in the server
- Enables a session failover safety mechanism

In general, sessions are used to keep the state of a user accessing an application between several requests. The AS Java architecture provides a reduced number of sessions per VM, which is achieved through separating the inactive user sessions from the active user sessions and reducing the number of sessions stored in a VM.

An active user session is a session which is currently bound with a request that is processed by the server, while an inactive user session is currently not bound with a request.

Active user sessions are stored inside the memory space of the VM, while inactive user sessions are stored in a shared memory region which is not damaged if the VM crashes due to certain problems such as Out of Memory errors, JVM bugs, and so on. In case of failure, the inactive user sessions stored in the shared memory region remain unaffected, and they can be mapped to a different VM process when a request for that session needs to be processed.

Thread Management

The AS Java thread management system provides common handling, maintenance, configuration, error processing, and monitoring for the thread resources used for execution of parallel system and application requests. The thread management system comprises two thread managers which handle the system and application operations separately. The Thread

Manager supplies threads for the AS Java system operations such as framework tasks, system events handling, and services management, while the Application Thread Manager supplies threads for application requests processing.

Thread management controls the thread usage in the system and ensures that the overload on one AS Java component cannot deplete the thread resources and leave the other components blocked due to lack of threads. This is achieved by effective load control for application requests, optimal resource management, low synchronization level, and high degree of isolation.

Reference

The managers that constitute the subsystems of Java Enterprise Runtime are:

- **Application Thread Manager** – handles threads in which client applications' source code is executed.
- **Cache Manager** – provides a centralized cache management infrastructure that consolidates caches used by various components within the system.
- **Class Loader Manager** – responsible for registering and removing loaders and references between them.
- **Cluster Manager** – manages communication between elements in the AS Java cluster. It updates the information about the status of each cluster element and the services running on it.
- **Configuration Manager** - manages the process of storing and reading persistent configuration data to and from a relational database.
- **Database Manager** – provides a pooling functionality for database connections.
- **Licensing Manager** – handles SAP licenses.
- **Locking Manager** – handles locking as interface to the Enqueue Service.
- **Log Manager** – manages the process of logging system events.
- **Pool Manager** – manages the process of pooling Java objects.
- **Service Manager** – manages the lifecycle of AS Java system components and acts as a container in which all services in the cluster work.
- **Session Manager** – manages the lifecycle of user sessions and provides failover capabilities.
- **Thread Manager** – handles threads in which the AS Java system operations are executed.

 AS Java System Components

The AS Java system components build the second level of the system which provides various runtime functions and programming APIs. Built on top of the runtime and being able to communicate and use each other, these components form the complete system infrastructure to run both Java EE and SAP proprietary applications.

Components

The following types of components exist:

- **Facades** – they simplify the relationships between SAP NetWeaver layers and client applications. Facades are the only official way for clients to access the AS Java API.

They help to define what is an ‘external (publicly available) API’ and what is an ‘internal API’. Everything that is part of a facade is public and the client code must be built against it. Everything that is not part of a facade is not official and the client code should not rely on it. Clients in this context can be components from other layers of the product and customer applications. If a client needs a reference to the public API of a certain component (service, interface, or a library), it must reference the facade which contains the API of the component.

- **Interfaces** – they define how different components of the system work together. At runtime, they provide the system with their name and classes (no objects). They are used by services components that provide their implementation.
- **Libraries** – they provide name, classes and objects to the system. These objects are created by the system when it loads the library, or when an object is first requested. Libraries are not active components – they have no definite life cycle, do not allocate resources themselves and do not keep any kind of configuration information in the system. Other library components or services components usually access them using static methods.
- **Services** – they provide the system with their name, classes, and runtime objects. The runtime objects are registered in the system once the components classes have been loaded. Service components can access and utilize functions of the runtime through the Framework API. Services are active components with a definite life cycle. They can allocate resources at their startup time and are responsible for releasing them at shutdown time.

  There are core services which provide the core functionality and should always be running, otherwise the system will stop. The rest of the services provide additional functionalities and are not required for the operation of the system.

Component References

Two types of references describe the dependencies between different system components – weak and strong.

Weak Reference

Component A sets a weak reference to component B if it needs to use classes of B. Throughout the life cycle of a component, a weak reference means:

- If component B exists, a reference from the class loader of A is set to the class loader of B.
- Component B must be resolved for component A to be resolved.
- All events that refer to a change of state of component B are sent to component A.
- If the system has to unload component B, it must first unload component A.

Weak Reference Appliance Matrix

Reference to Reference from	Interface	Library	Service
Interface	Yes	Yes	No
Library	Yes	Yes	Yes
Service	Yes	Yes	Yes

Even though it is possible to set a weak reference from a library to an interface or a service, it is not common practice. Therefore, if you encounter such a situation, double-check the design of your library before you set the link.

Strong Reference

Component A sets a strong reference to component B if it needs to use classes and runtime objects that B provides. Throughout the life cycle of a component, a strong reference means:

- If component B exists, a reference from the class loader of A is set to the class loader of B.
- Component B must be resolved for component A to be resolved.
- All events that refer to a change of state of component B are sent to component A.
- Component B must be started for component A to be started.
- If the system has to stop component B, it must first stop component A.
- If the system has to unload component B, it must first unload component A.
- If component A has a strong reference to component B, then B cannot have a strong reference to A.

Strong Reference Appliance Matrix

Reference to Reference from	Interface	Library	Service
Interface	No	No	No
Library	No	No	No
Service	Yes	No	Yes

In the case of a service component with a strong reference to an interface, the referencing service will be started only if there is another service that implements the referenced interface and this service is started.

Applications

Applications, which can range from simple Web applications to complete Enterprise JavaBeans (EJB) based solutions, form the third level of the AS Java system architecture. The boundary between the applications level and the AS Java system components level is defined by the Java EE APIs along with a number of SAP proprietary APIs. Applications use them to utilize the functions of the different types of components on the lower level in the system.

An enterprise application consists of several types of application components. These components reside in different containers on the AS Java system, such as Web container, EJB container, Web services container and so on. Each of these containers provides runtime services and lifecycle management for the application components.

Applications inform other applications or AS Java system components that they need to use their classes or runtime objects by defining references to them.

Application References

The AS Java provides a way of fully integrating the applications running on it with the existing AS Java system components, by defining a reliable and comprehensible mechanism for defining dependencies between them. Using this mechanism, an application can reuse the resources of existing components.

The purpose of this mechanism is to hide the complexity from application developers having to explicitly define references between the application class loader and other system class loaders, and register them in the AS Java Class Loading System. The Deploy Service has a built-in mechanism that performs all these tasks using the references that the application developer has defined.

Application References Appliance

The mechanism is based on defining references from an application to another component. An application can set references to:

Other applications that are deployed on the AS Java

- AS Java system components (services, libraries, and interfaces)
- A standalone Web module (a WAR file)
- A standalone EJB module (a JAR file)
- A standalone resource adapter module (an RAR file)

There are two types of application references – weak and strong.

Weak Application References

An application sets a weak reference to another component or application if it needs to use its classes. If the referenced component or application is not started (or not loaded for libraries), the application will be started but will not be able to use classes of those components.

Strong Application References

An application sets a strong reference to another component or application if it needs its classes and runtime objects. If the referenced component or application is not started (or not loaded for libraries), the application will not be started at all.

Default Application References

For the purpose of simplifying the work of the application deployer, the AS Java provides a set of default references. If you want to see a list of the default references, see the default value of the StandardApplicationReferences property of the Deploy Service.

If you need to set additional references, you must set them at application deployment time.

Do not remove any of these references or change the order in which they are loaded. Even if you remove some of the references at runtime, they will be loaded again at startup, that is, you cannot permanently remove these references.

Zero Administration

Use

The main goal of Zero Administration is to simplify the technical configuration within the AS Java by removing all system-dependent settings from the configuration database. This is done via dynamic configuration which adapts itself to the system environment, and template-based configuration which ensures configuration consistency.

Zero Administration supports:

- Built-in configuration templates.

- System copies and load based installations without the need to adapt configuration manually.
- Changing the system environment (system name, instance name, hosts and so on) without changing the configuration in the database.
- Adding instances with a minimum configuration overhead.

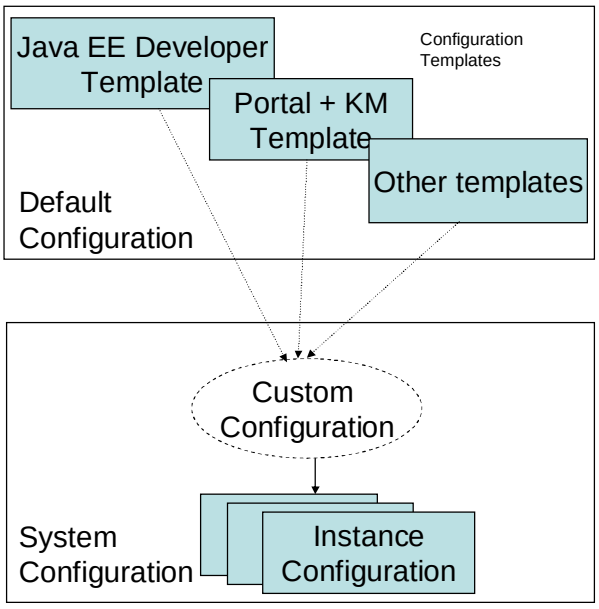
Integration

The goals and benefits of Zero Administration are achieved by a set of new features within the Configuration Manager and the infrastructure itself.

Features

Zero Administration concept is implemented in:

- Configuration Templates [Page 18]
- Dynamic System Configuration [Page 18]



AS Java Cluster Configuration Structure

The general AS Java cluster configuration structure consists of two levels:

- Default system-independent configuration including configuration templates, which are provided during installation. A configuration template is derived from a default configuration and overwrites the setting according to the specific usage it is assigned to.
- System configuration, including the actual configuration of the specific instances, which are part of the system. The instance configuration is derived from the configuration template belonging to the corresponding use case. The system-dependent configuration dynamically adapts itself to the actual system environment without overwriting any settings inherited from the template.

If you need to customize any additional AS Java components (for example landscape connectivity), you can do custom configuration within system configuration using the Config

Tool. Custom configuration is stored separately and does not affect default configuration. Custom configuration changes are visible across all instances that are configured via the same configuration template.

If you want to customize a setting for one particular instance only, apply the change on the instance level.

For more information about using the Config Tool to configure the AS Java parameters, see the online documentation on help.sap.com.

Configuration Templates

Definition

Configuration templates contain the predefined instance configuration for specific scenarios.

A configuration template is system independent and can be moved between different systems as all system dependencies are configured using dynamic configuration, which includes parameterized and computed settings and value links. A configuration template is derived from a default configuration and overwrites the setting according to the specific usage the template is assigned to.

Structure

A configuration template may contain the following configuration:

- Instance layout – the number of server nodes running on the instance defined as an arithmetic expression dependent on the hardware available for this instance (number of CPUs and memory available).
- Java Virtual Machine (JVM) configuration – JVM memory settings and JVM parameters. If no settings are applied, the default settings will take effect.
- Kernel configuration – system independent properties of the manager components of the AS Java.
- Service setting – system-independent service properties of each service component which is part of the installation.
- Application configuration – system independent application configuration of each application which is part of the installation.
- Runtime filter configuration – for activation and deactivation of components according to the use case scenario the template belongs to (the components which are not needed for the particular usage are disabled, which means they are not started whereas those that are needed are enabled).

Dynamic System Configuration

Use

The main idea of dynamic system configuration is to use parameterized settings instead of static values for system-dependent configuration. Thus, if the system environment is changed or new hardware is added, the overall configuration is changed dynamically without the need to adapt the configuration in the database manually.

Features

Parameterized Settings

Parameterized settings contain parameters specified in the instance profile provided within the Configuration Manager. These parameters are transparently substituted during runtime.

The structure of a parameter is ${<Instance profile parameter>}.

For example: *Message Server host: ${MSGSRV_HOST}*

Computed Settings

Computed settings are used in case a simple parameter substitution is not sufficient and the value needs to be calculated out of specific system parameters (such as heap size, cache size).

Computed settings are arithmetic expressions containing system parameters from the instance profile. The parameters are transparently substituted and the arithmetic expression is evaluated during runtime.

An arithmetic expression consists of parameters, constants, simple operators and brackets. Operators supported are “+”, ”-“, ”*”,” /”; “min”, “max” , “round” and “truncate” function;

Format: expr = expr1 min expr2 [corresponds to expr = min(expr1, expr2)].

round(${AMOUNT_MEMORY}/3)

For example: *Number of Server Nodes per Instance: 2*${CPU_COUNT}*

Value Links

Value links contain links to other settings in case a setting is dependent on another setting stored somewhere else in the Configuration Manager. A value link is transparently resolved and substituted during runtime.

The structure of a value link is *$link{<config_path>#<property_key>}*

Value links are intended for internal use only.

Instance Profile

The Zero Administration instance profile is a defined set of system-dependent configuration parameters provided by the Configuration Manager. They include:

- SAP system parameters
- Java instance parameters
- Hardware and OS parameters
- Node-specific parameters

The instance profile parameters can be referenced by using parameterized settings.

Instance Profile

The following table lists the current instance profile parameter set provided by the Configuration Manager.

Parameter	Description
SAP system parameters	
${SYSTEM_NAME}	The three-character name of the system (for example, D70).
${SYS_GLOBAL_DIR}	The path to the system global share (for example, \\centralhost\sapmnt\D70\SYS\global).

${SYS_DATASOURCE_NAME}	The name of the system data source (for example, SAPN69DB).
${MSGSRV_HOST}	Host where the Message Server is located.
${MSGSRV_PORT}	Port of the Message Server.
${ENQSRV_HOST}	Host where the Enqueue Server is located.
${ENQSRV_PORT}	Port of the Enqeue Server.
${START_TIME}	Start time of the instance as a long value.
Java instance parameters	
${INSTANCE_ID}	The ID of the current instance.
${INSTANCE_NAME}	The name of the current instance (for example, 'JC20' or 'J20' for a dialog instance).
${INSTANCE_NUMBER}	The number of the current instance (for example, for instance 'JC20' the number is '20').
${INSTANCE_HOST}	The host name of the host on which the instance is running.
${INSTANCE_DIR}	Path to the instance folder (for example, c:\usr\sap\D70\JC20).
${FULL_INSTANCE_NAME}	The full instance name of the current instance (for example, D70_JC20_pcj2ee01, where D70 is system name; JC20 is instance name and pcj2ee01 is instance host).
${SYSTEM_INFO}	The configuration path to the System Info configuration, which exposes parameters for the usage in the components configuration.
${JAVA_HOME}	The path to the bin folder of the Java installation.
${DB_DRIVER_LOCATION}	The driver location string specifying the driver location needed for the DB connection.
${VM_VENDOR}	The name of the VM vendor (for example, SAP).
${EXE_DIR}	Directory where native executables are located.
Hardware and OS parameters	
${CPU_COUNT}	The number of CPUs available on the host which should be used by the instance.
${AMOUNT_MEMORY}	The amount of physical memory on the host which should be used by the instance.
${OS_NAME}	The SAP name of the OS.
${OS_BITLENGTH}	The bit length (for example, 32, 64) of the platform.
${OS_UNICODE}	Specifies if platform is Unicode enabled or not (uc, nuc).
Node-specific parameters	
${NODE_INDEX}	Instance unique ID identifying a cluster node on an instance.

Related Content

For more information about configuring and administering AS Java and the components running on top, see:

- ***SAP NetWeaver Composition Environment 7.1 Master Guide*** on sdn.sap.com
- ***SAP NetWeaver Composition Environment 7.1 Administrator's Guide*** on help.sap.com