Informatica® ActiveVOS
9.2.4.6

Server User Guide

Informatica ActiveVOS Server User Guide
9.2.4.6
March 2020

See patents at https://www.informatica.com/legal/patents.html.

DISCLAIMER: Informatica LLC provides this documentation "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of noninfringement, merchantability, or use for a particular purpose. Informatica LLC does not warrant that this software or documentation is error free. The information provided in this software or documentation may include technical inaccuracies or typographical errors. The information in this software and documentation is subject to change at any time without notice.

NOTICES

This Informatica product (the "Software") includes certain drivers (the "DataDirect Drivers") from DataDirect Technologies, an operating company of Progress Software Corporation ("DataDirect") which are subject to the following terms and conditions:

1. THE DATADIRECT DRIVERS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.
2. IN NO EVENT WILL DATADIRECT OR ITS THIRD PARTY SUPPLIERS BE LIABLE TO THE END-USER CUSTOMER FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR OTHER DAMAGES ARISING OUT OF THE USE OF THE ODBC DRIVERS, WHETHER OR NOT INFORMED OF THE POSSIBILITIES OF DAMAGES IN ADVANCE. THESE LIMITATIONS APPLY TO ALL CAUSES OF ACTION, INCLUDING, WITHOUT LIMITATION, BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS.

Publication Date: 2023-11-27

Table of Contents

Preface 5

Chapter 1: Server User Guide 6

Welcome to Process Server 6

Installing Configuring and Starting Up the Server. 7

Using ActiveVOS Designer 7

Using the ActiveVOS Server Administration Console 8

Using ActiveVOS Central 8

Accessing Service Endpoints 8

List of Process Server URLS 10

Process Server APIs and SDKs 10

Chapter 2: Security Configurations 12

Security Configurations 12

Configuring Authentication for SAML-Secured Services 13

Securing Process Server Components 13

Configuring Your Application Server to Work with Process Server Security Roles 15

Chapter 3: Standard for Java-based Service Endpoints for Process Consumer (My Role) Partner Links 19

Invoking a Process Using Standard SOAP or Policy-Driven Bindings 19

Invoking a Process from Java 21

Chapter 4: Custom Invoke Handlers 24

Referencing the Custom Invoke Handler Interface in a Java-Based Implementation 25

Adding the Custom Invoke Handler Attribute to the Process Deployment Descriptor 26

Packaging the Custom Invoke Handler as an EJB 27

Deploying Custom Invoke Handler Files 28

Preface

This module contains information on using and configuring Process Server.

CHAPTER 1

Server User Guide

This chapter includes the following topics:

- Welcome to Process Server , 6
- Installing Configuring and Starting Up the Server, 7
- Using ActiveVOS Designer , 7
- Using the ActiveVOS Server Administration Console , 8
- Using ActiveVOS Central , 8
- Accessing Service Endpoints , 8
- List of Process Server URLS , 10
- Process Server APIs and SDKs , 10

Welcome to Process Server

Process Server includes the following components to run and manage Business Process Execution Language (BPEL) processes.

- Process Server engine and related database configuration files
- Process Console
- Process Central, which is a request, report, and task management application
- Support for custom task management client applications
- Business event processing for business analysis
- User-defined reports
- Catalog resources for updating business process logic and HTML forms

The Process Server engine manages processes, including process versioning (effective and expiration dates), invocation handlers for partner endpoints, endpoint retry, security, and many other policies, and process exception management.

The Process Console provides pages to manage deployed and active processes, an alerting system, and endpoint locations. It also has engine configuration settings for performance monitoring and management.

Installing Configuring and Starting Up the Server

Be sure to follow the instructions in the Installation wizard that appears when you launch `config_deploy.bat` or `config_deploy.sh`. These executables are located in the `\bin` folder of your product installation folder.

Install steps include:

- Perform pre-installation steps to remove older versions and set up engine clustering, if appropriate.
- Follow all steps in the Install and Configuration Prerequisites page.
- Set up the database, as described in the Install and Configuration Configuration page.
- Install the software using the `config_deploy` script mentioned above.

Configuration steps include:

- Security Configurations
- Securing Process Server Components
- Configuring Your Application Server to Work with Process Server Security Roles

Startup steps include:

- Start your application server to start Process Server.
- Display the Process Console in a Web browser using the following URL:
  `http://localhost:8080/activevos`

  Change the port number if it is already in use on your computer.
- Navigate to the License page to add your license.
- Navigate to the Server Status page to start the server.

Using ActiveVOS Designer

ActiveVOS Designer is a comprehensive environment for creating, testing and deploying BPEL processes.

Based on the Eclipse visual framework, ActiveVOS Designer provides a full complement of WS-BPEL 2.0 constructs and supports BPEL4WS 1.1 processes. You can drag items directly from a WSDL file or from a tools palette onto a Process Editor canvas. As you build your process, the BPEL XML code is automatically generated and validated. Problems are immediately reported with suggested solutions.

Several shortcuts aid you in creating process flows and data assignments.

Deployment wizards package all required files together into a business process archive and move the archive to your ActiveVOS server.

Designer also includes support for BPEL4People, Version 1.0 so that you can include human workflow within BPEL processes. This functionality introduces human workflow into a process for business applications that require the services of a person in addition to Web services. The human tasks are routed to ActiveVOS Central at runtime, where users can complete them and send their response back to the process.

Using the ActiveVOS Server Administration Console

The ActiveVOS Server Administration Console allows you to manage and configure the ActiveVOS engine and the resources that are deployed into it.

The Administration Console provides several ways to select, inspect, and correct process instances and related deployment, execution, and server logs, descriptors, WSDL and other resource files.

You can also configure many ActiveVOS engine properties to monitor and manage performance.

System and user-defined reports provide charts and graphs for analyzing active and complete processes and human tasks.

You can display the Administration Console in a Web browser using the following URL:

```
http://localhost:8080/activevos
```

Change the port number if it is already in use on your computer.

Using ActiveVOS Central

ActiveVOS Central is a browser-based application where end-users can manage tasks, requests, and reports. The tasks are part of a running business process that requires human interaction. Requests are forms that can be submitted to the server to start a new instance of a deployed business process. Reports display charts and graphs of interest to various users and groups.

ActiveVOS Central is available at the following URL:

```
http://localhost:8080/activevos-central
```

Note that the host and port may be different in your environment.

Users must login to ActiveVOS Central. This requirement means that the ActiveVOS administrator must map the human task security role (abTaskClient) to all users. For details, see *Securing ActiveVOS Server Components*.

When users log in to ActiveVOS Central:

- ActiveVOS Central handles session timeouts, but not authentication. The timeout value is configurable.
- ActiveVOS Central checks to see if any new resources for the user have been deployed to the server
- If new resources have been deployed, then ActiveVOS Central invalidates the cached resources
- The latest resources are available to users when they login

ActiveVOS Central has a debug mode to help developers debug the deployed resources.

Accessing Service Endpoints

When processes are deployed to Process Server, they are made available as service endpoints. Users and automated business processes start up process instances by submitting a request to a service endpoint.

In Process Console, you can see a list of services by selecting Service Definitions.

Default Service Endpoint: SOAP 1.1

By default, all services, except REST services, are listed at:

`http://localhost:8080/active-bpel/services`

This URL displays all services with a SOAP 1.1 binding. A request to start a SOAP 1.1 process can include:

- Binding style of Document Literal, RPC Literal, RPC Encoded, or Policy Driven (handled by policy assertions)
- Multiple-part message
- Processes that may not be WS-I compliant
- SOAP headers
- Binding to HTTP GET and POST methods

Additional Service Endpoints

In addition to SOAP 1.1, you can use the following service endpoints:

- `http..active-bpel/services/SOAP12`
  SOAP 1.2 services. All services can be invoked except for REST services and the `ActiveBpelAdmin` service. Requests may be submitted to services exactly like SOAP 1.1 services, described above.
- `http..active-bpel/services/REST`
  Only REST services can be invoked. A request must be submitted with the following conditions:

  - Requests must conform to the input message defined in `aeRest.xsd`
  - HTTP methods HTTP GET, POST, PUT, and OPTIONS
- `http..active-bpel/services/XML`
  All services can be invoked except for REST and the `ActiveBpelAdmin` service. A request must be submitted with the following conditions:

  - Binding style must be document literal
  - Input and output messages must be a single-part element
  - Process must be WS-I compliant
  - Request is HTTP POST with header content-type of `text/xml`
- `http..active-bpel/services?JSON`
  All services can be invoked except for REST and the `ActiveBpelAdmin` service. A request must be submitted with the following conditions:

  - Binding style must be document literal
  - Input and output messages must be a single-part element
  - Input message must be represented in JSON
  - Process must be WS-I compliant
  - Request is HTTP POST with header content-type of application/json
- `http://host:port/active-bpel/catalog/project:/relative_path_to_resourceFolder`
  Examples:

  `http://localhost:8080/active-bpel/catalog/project:/AutoStep/wsdl/telco.wsdl`

  `http://localhost:8080/active-bpel/catalog/project:/AutoStep/schema/telco.xsd`

List of Process Server URLS

The following table shows a list of available URLs when the Process server is running:

http://<host>:<port>/	Description
activevos	Process Console
activevos-central	Process Central
activevos-central/inbox	Older style task client for B4P processes
active-bpel/services	For details, see the Service Endpoints topic.
active-bpel/catalog/project:/ path/to/catalog/ resource-type/ resource-name	Opens a WSDL, schema, or other resource: http://localhost:8080/active-bpel/catalog/ project:/ HumanTaskDemo/schema/loanRequest.xsd
active-bpel/i18n/ name_of_props.properties	Externalized strings for Process Central multilingual support
active-bpel/avccatalog/project:/ path/to/catalog / name_of_form.html	Opens Process Central request or task form. Required login with abTaskClient security role.
activevos/process/processview.jsp? pid=101&readonly=1&view=2	To make the Active Process Detail view read-only, use the `readonly=1` parameter. To show and hide different components of the Active Process Detail view, use the view parameter with the following arguments: 1: default, show all components (Outline, Canvas, Properties) 2: Canvas only 3: Outline and Canvas 4: Canvas and Properties

Process Server APIs and SDKs

Developers can take advantage of several Process Server APIs. The APIs are documented in the *Process Developer Help*. Here you will also find sample code and applications.

The following table is a partial list of the Process Server interfaces:

Name	Description	Service Endpoint http://<host>:<port>/
Process Central	Process Central primarily relies on the `AvosCentralApi` service for configuration	active-bpel/services/ JSON/AvosCentralApi
Informatica Business Process Manager extensions to WS-HT	Operations for interacting with human tasks, such as *authorize* and *getInstance*	active-bpel/services/ AeB4PTaskClient-aeTaskOperations

Name	Description	Service Endpoint `http://<host>:<port>/`
Admin Service	Operations to communicate with the Process Console	`active-bpel/services/ActiveBpelAdmin`
WS-HT	Web Service-Human Task operations as described by the OASIS standards committee	`active-bpel/services/AeB4PTaskClient-taskOperations`
WS-HT Task Feed	`getMyTasks()` operation represented as a RSS or ATOM syndication feed	`active-bpel/services/REST/AeB4PTaskFeed`
XML and JSON Bindings	Process Server exposes all its processes and services via simple XML and JavaScript Object Notation (JSON) bindings	For details, see Process Server URLS.

CHAPTER 2

Security Configurations

This chapter includes the following topics:

- Security Configurations , 12
- Configuring Authentication for SAML-Secured Services , 13
- Securing Process Server Components , 13
- Configuring Your Application Server to Work with Process Server Security Roles , 15

Security Configurations

The OASIS WS-Security V1.0 (WSSE) standard establishes a framework for assuring the integrity and confidentiality of SOAP messages. Message integrity is assured via digital signature, and confidentiality of message data is achieved via encryption. The standards referenced by WS-Security V1.0 are supported by Process Server.

To use WS-Security features, you must set up the container in which Process Server is deployed with the necessary certificates manager or keystores that are often used for standard SSL processing. Also, you must provide a properties file that contains settings required for Process Server to interact with your platform's encryption and certificate management. The properties file is named `crypto.properties`.

The following properties must be set in crypto.properties to match the your platform:

- **org.apache.ws.security.crypto.provider=<provider>**
  where `<provider>` must be the default unless a custom provider is supplied that implements the apache crypto interface. The default is: `org.apache.ws.security.components.crypto.Merlin`
- **org.apache.ws.security.crypto.merlin.keystore.type=<type>**
  The `<type>` is based on the format of the keystore, usually `jks` or `pks12`.
- **org.apache.ws.security.crypto.merlin.keystore.alias=<alias>**
  The `<alias>` is the name that the private key and certificate are known by.
- **org.apache.ws.security.crypto.merlin.keystore.password=<password>**
  where `<password>` is an optional property. Include a password if one is required for a keystore.
- **org.apache.ws.security.crypto.merlin.file=<keystore filename>**

The keystore must be accessible by the server from the file system using the path specified by `org.apache.ws.security.crypto.merlin.file` in `crypto.properties`. Also, the crypto.properties file must be available on the server's classpath. The target location of these files varies depending on the target platform.

The following is an example of crypto.properties:

```
org.apache.ws.security.crypto.merlin.keystore.password=pw
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.file=ae.keystore
org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.keystore.alias=myadmin
```

Configuring Authentication for SAML-Secured Services

Process Server supports the Security Assertions Markup Language (SAML) standard for exchanging authentication and authorization data between an identity provider (a producer of assertions) and a service provider.

To configure Process Server to use SAML policy assertions, you must do the following:

- In the Process Deployment Descriptor (PDD) of your BPEL processes, add a SAML policy assertion for my roles and/or partner roles that send/receive SAML-authenticated messages. This policy describes parameters that a service uses to make access control decisions. For details, see *SAML* in the *Process Developer Help*.
- Add a SAML properties section to the Process Server **crypto.properties** file.

Here is an example of the SAML properties to add:

```
org.apache.ws.security.saml.issuer.key.name=aeadmin
org.apache.ws.security.saml.issuer.key.password=password
org.apache.ws.security.saml.issuer=http://www.abe-saml-demo.com/saml
org.apache.ws.security.saml.subjectNameId.qualifier=http://www.abe-saml-demo.com/saml
```

Note that the key name and password must match what you have in the Process Server keystore.

Securing Process Server Components

You can provide permission to groups of users to access Process Serve by configuring the security roles provided.

Note: Configuration of one role is required. It is `abTaskClient`, which is required for access to Process Central. The remaining roles are optional and can be configured during the config-deploy process. However, if you have a license for the Multi-Tenant feature, you must configure security.

To secure Process Server components:

1. Run the `config-deploy` utility (the installation utility), and navigate to the Security page if you have already installed the application.
2. On the Security page, select the checkboxes for the components you want to secure:
   - Administrative functions. This setting enables you to configure three levels of authenticated users for access to the Process Console and deployed processes.
   - Process services. This setting enables you to configure authenticated users to initiate process instances of deployed processes.

- Process Server Identity Service Consumer. This setting enables you to configure authenticated users of Process Developer to open and use the Identity Service (member directory) configured in Process Server Console. The Identity Service is a resource used in many types of processes.

3. Complete the `config-deploy` installation. If you are only setting up security, note that all your other settings from a previous installation are still in tact.
4. Review the security roles definitions in the table below.
5. Assign the roles to users and groups as desired, to tell your application server to use them.

Process Server Security Roles

The following sections describe the roles that you use to secure Process Central, Process Console, and deployed processes.

Administrative Functions

These functions add security parameters to the ActiveVOS Consoles and services by setting the following roles:

abAdmin

Users associated with this role have full administrative rights to ActiveVOS Server.

abBusinessManager

Users associated with this role have access to process instance details (but cannot operate on them). They can monitor active processes and tasks, and work queues. They have a read-only view of process instance details.

abDeployer

Users associated with this role have rights restricted to deploying business process archive files to ActiveVOS Server.

abDeveloper

Users associated with this role have rights restricted to service artifacts, endpoint information, and sample messages for the services they consume and expose (that is, processes) after they are deployed. Developers need the ability to deploy process deployment archives, initiate process execution and analyze them. Developers also need to configure global function contexts for custom functions, URN mappings, and the ability to schedule process execution. Specifically, this user has access to the Active Process list, the Process Instance View, the Active Task and Work Queues lists, the Server Log, the Dashboard and all reports, and the catalog's content.

abOperator

Users associated with this role have rights restricted to operating the system. These include observing the functionality of processes, managing process instances using the process instance detail view, running reports, logging, viewing exceptions, acquiring information on service operations, adding and removing tenants, and managing the scheduled database delete schedule.

abTaskClient

Required. You must configure permission to access ActiveVOS Central for all users. In addition, users who interact with the Human Task (WS-HT) API must have this role.

ActiveVOS Central presents a login page to users.

Process Services

The process services adds security parameters to the Web Services handler for all deployed BPEL services with a role. The services listed at `http://[host]:[port]/active-bpel/services` are secured. The process services (roles) are:

abRestrictedServiceConsumer

Users associated with this role cannot access a service unless it is deployed with allowed roles specified in the `pdd` and the user belongs to at least of these roles. If no roles are specified in the `pdd`, access to services with no roles specified in the `pdd` are also denied. Users in this role can view the `wsdl` files for other services like **abServiceConsumer**; however, they are blocked at runtime.

abServiceConsumer

Users associated with this role have rights restricted to start process instances of deployed processes, including from ActiveVOS Central, the Eclipse Web Tools Project, or other client application,such as SOAPUI.

abTenantAdmin

(For a Multi-Tenant licensed server only.) Users associated with this role have rights to deploy and manage contributions into a configured tenant on the server.

Based on a Tenant Definition configured by the ActiveVOS Server administrator (with the abAdmin role), a tenant administrator user can log into the tenant context on the server. A service consumer user can create process instances for processes deployed to the tenant context.

Identity Service Consumer

The identity service consumer adds security parameters to the Web Services handler for Process Identity service used by the ActiveVOS Central application using the following roles:

abIdentityListConsumer

Only users associated with this role or **abAdmin** have rights to submit Web Service requests to the identity service from Process Developer.

Configuring Your Application Server to Work with Process Server Security Roles

Setting up secure access to Process Server includes the following steps:

- Run the Process Server `config-deploy` utility and select security options. When Process Server is deployed, it is configured to use its security roles.
- Configure your application server to use Process Server security roles by mapping roles to users and groups (discussed below).
- Configure the security details of your application server for authentication and authorization methods (discussed below).

Mapping Roles to Users and Groups

Each application server has a different set of steps for configuring security for deployed applications. You must familiarize yourself with your application server environment to understand how to configure the security methods you need. However, here are some general guidelines and links to documentation.

Application Server	Setup Guidelines
Tomcat	Add a *realm* to `server.xm`l, connecting to an existing "database" of usernames, passwords, and user roles. See examples below. For an LDAP-based database, add the Process Server security roles to your database and map them to groups. Refer to *Realm Configuration HOW-TO* at the following URL (Tomcat 6.0): `http://tomcat.apache.org/tomcat-7.0-doc/realm-howto.html`
JBoss	Refer to `https://docs.jboss.org/author/display/AS71/Admin+Guide` Add a security domain and login module to your chosen configuration file. The security domain must be named the same as what was selected in the Process Server `config-deploy` utility. If the name was not changed in the utility, it defaults to "ActiveVOS". Add roles to one of the following: a set of user/role files, LDAP-based database, or JDBC-based database.
WebLogic	Login to the WebLogic console and set up the LDAP provider, if you have not already done so: (Navigate to Security Realms. In the default realm, select **Providers > Authentication > Default Authenticator** and change the Control Flag from REQUIRED to OPTIONAL. Add and configure your LDAP provider.) For details, see *Configuring Authentication Providers* at `http://download.oracle.com/docs/cd/E13222_01/wls/docs103/secmanage/atn.html` In your security realm, navigate to Roles and Policies. Expand Global Roles, select Roles, and enter the Process Server security roles. For details, see *Users, Groups, and Security Roles* at `http://download.oracle.com/docs/cd/E12840_01/wls/docs103/secwlres/secroles.html`
WebSphere	Login to the WebSphere console and follow the links similar to this example: **Applications > Application types > WebSphere enterprise applications > Process Server** Look for the *Security role to user/group mapping* group. Map the existing Process Server security roles to groups. **Note:** You must ensure that WebSphere application security is set up correctly. On the WebSphere Console navigation area, select **Security > Secure administration, applications, and infrastructure**. Then select the checkbox next to Enable Application Security and select **Apply**.

Tomcat Examples

`tomcat\conf\server.xml` (*file-based configuration*)

```
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
              resourceName="UserDatabase"/>
         tomcat\conf\tomcat-users.xml file:
   <role rolename="abTaskClient"/>
   <role rolename="abServiceConsumer"/>
   <role rolename="abAdmin"/>
   <user username="admin" password="admin"
                 roles="abAdmin, abTaskClient, abServiceConsumer"/>
```

`tomcat\conf\server.xml` (*LDAP-based configuration*)

```
<Realm  className="org.apache.catalina.realm.JNDIRealm" debug="99"
   connectionName="uid=ldapadmin,ou=system"
   connectionPassword="password"
   connectionURL="ldap://<LDAP_Server_name>:<Port>"
   userSubtree="true"
   userBase="ou=users,o=xyzuser"
   userSearch="(uid={0})"
   roleSubtree="true"
   roleBase="ou=groups,o=xyzrole"
   roleName="cn"
   roleSearch="(uniqueMember={0})"
/>
```

JBoss Examples

Note: The following two examples are specific to JBoss 7.1.1; other versions require different configurations. Also, what you see here is just to indicate how you might configure JBoss. What you will actually declare will almost certainly be different. For all JBoss versions, you must review the JBoss documentation.

[Path] `\configuration\` *[file.xml]* (*file-based configuration*)

The file name you will enter will be unique to your installation. The only part of the name that will be there is “configuration”. Here is an example: `C:\servers\jboss-as-7.1.1.Final\standalone\configuration\standalone-full.xml`

```
<security-domain name="ActiveVOS" >
  <authentication>
    <login-module code="RealmUsersRoles" flag="required">
      <module-option name="usersProperties"
             value="${jboss.server.config.dir}/application-users.properties"/>
      <module-option name="rolesProperties"
             value="${jboss.server.config.dir}/application-roles.properties"/>
      <module-option name="realm" value="ApplicationRealm"/>
      <module-option
            name="unauthenticatedIdentity"> anonymous
      </module-option>
    </login-module>
  </authentication>
</security-domain>

${jboss.server.config.dir}/application-roles.properties
   admin=abTaskClient,abAdmin,abServiceConsumer

${jboss.server.config.dir}/application-users.properties
    admin=admin
```

[Path] `\configuration\` *`[Path]`*(*LDAP-based configuration*)

The file name you will enter will be unique to your installation. The only part of the name that will be there is “configuration”. Here is an example: `C:\servers\jboss-as-7.1.1.Final\standalone\configuration\standalone-full.xml`.

```
<security-domain name="ActiveVOS">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapExtLoginModule"
            flag="optional">
      <module-option name="java.naming.provider.url"
            value="ldap://myserver:3268"/>
      <module-option name="bindDN"
            value="CN=admin,CN=Users,DC=myDomain,DC=myCompany,DC=local"/>
      <module-option name="bindCredential" value="admin"/>
      <module-option name="baseCtxDN"
            value="DC=myDomain,DC=myCompany,DC=local"/>
      <module-option name="baseFilter"
            value="(sAMAccountName={0})"/>
      <module-option name="rolesCtxDN"
            value="DC=myDomain,DC=myCompany,DC=local"/>
      <module-option name="roleFilter" value="(member={1})"/>
      <module-option name="roleAttributeID" value="memberOf"/>
      <module-option name="roleAttributeIsDN" value="true"/>
      <module-option name="roleNameAttributeID" value="cn"/>
      <module-option name="roleRecursion" value="2"/>
      <module-option name="searchScope" value="SUBTREE_SCOPE"/>
      <module-option name="allowEmptyPasswords" value="true"/>
      <module-option name="java.naming.referral"  value="follow"/>
      <module-option name="unauthenticatedIdentity" value="aeadmin"/>
    </login-module>
  </authentication>
</security-domain>
```

CHAPTER 3

Standard for Java-based Service Endpoints for Process Consumer (My Role) Partner Links

A service endpoint for a process consumer lets you send a message to the Process Server that either creates a new process instance or gets routed to an existing process instance. In BPEL terminology, the endpoint is known as a *My Role service endpoint*. In Process Developer, the My Role endpoint is referred to as the *Process Consumer*.

The deployment options for service endpoints are configured as `myRole` elements within the process deployment descriptor (PDD) and are automatically published by Process Developer when you deploy a business process archive (`.bpr`).

Services can be deployed as SOAP web services, REST-ful HTTP endpoints, or as unpublished services only accessible to internal clients. Process Server uses the binding attribute on the `myRole` service to determine how to deploy the endpoint and make it accessible.

When you create a Process Deployment Descriptor, you can select one of the standard bindings for a `myRole` service. You can also create a Java implementation, if desired.

Invoking a Process Using Standard SOAP or Policy-Driven Bindings

When you create a Process Deployment Descriptor (PDD), you can select one of the standard bindings for a My Role service. These selections are:

- SOAP Bindings
- Policy-Driven Bindings
- Unpublished

SOAP Bindings

The simplest way to deploy a service endpoint as a Web Service is to select one of the standard SOAP bindings:

- Document Literal
- RPC Literal

- RPC Encoded

You only need to select the binding style and a service name. The SOAP binding elements for the selected style are generated based on the Port Type and Policies associated with the `myRole` element in the PDD.

After deployment, Process Server handles the receipt and reply of messages designated for the service endpoint. The messages are formatted and processed by Process Server based on the binding.

Services are deployed to the engine's internal web service provider. Web services are supported by the Metro SOAP stack. (Metro was developed by Sun as a core component of the Glassfish project.)

Both SOAP 1.1 and SOAP 1.2 endpoints are deployed for JAX-WS compliant services (Document Literal and RPC Literal). Legacy bindings such as RPC Encoded are deployed as SOAP 1.1 only.

When a request is received, the URL determines the target service and SOAP version. The last segment of the path corresponds to the service name specified in the PDD:

- For SOAP 1.1 the URL is:
  `http://host:port/active-bpel/services/<MyRoleServiceNameInPDD>`
- For SOAP 1.2 the URL is:
  `http://host:port/active-bpel/services/soap12/<MyRoleServiceNameInPDD>`

The WSDL for a deployed web service endpoint is accessible at:

```
http://host:port/active-bpel/services/<MyRoleServiceNameInPDD>?wsdl
```

A listing of all deployed web services hosted within the Process Server engine is found at the following URL:

```
http://host:port/active-bpel/services
```

In the Process Console, select the Service Definitions link.

The WSDL that is published for the service includes standard WS-Policy attachments for policy assertions including WS-Addressing and WS-Security.

Policy-Driven Bindings

A policy-driven binding defines how a service is exposed using a policy assertion defined on the `myRole` element.

There are many policy assertions, but only the following three can be used as a binding:

- **Referenced WSDL Binding**: As an alternative to letting the engine generate a SOAP binding based on a given style, you can reference a specific SOAP binding in WSDL by using a Referenced WSDL Binding policy assertion in the PDD. Use this feature to deploy services with message parts mapped as SOAP headers or specify non-empty soapAction attributes.
- **WS-Reliable Messaging**: The presence of WSRM policy dictates that the service is deployed as a Web service with support for the WS-Reliable Messaging protocol.
- **REST**: If REST policy is used, a REST-ful service endpoint is exposed using a URL to HTTP clients.

For details about policy assertions, see Endpoint References and WS-Policy.

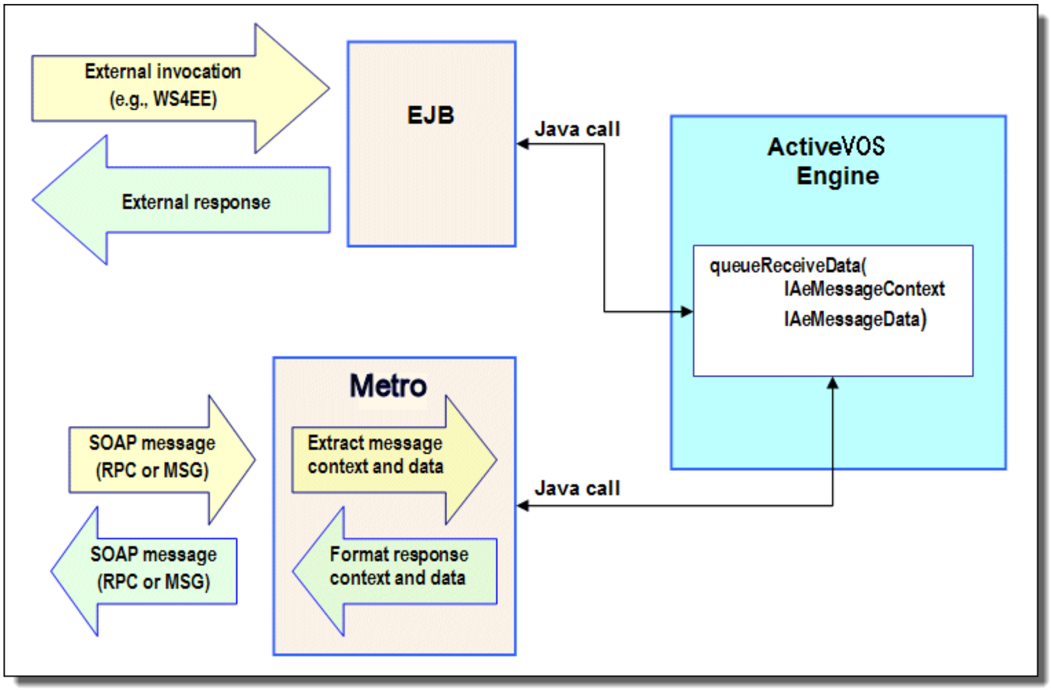
Unpublished Binding Style

When Unpublished is selected as the binding, the endpoint is not published to the SOAP stack and is only accessible to internal clients using process invokes or through custom java clients. For details, see *Invoking a Process from Java*.

Invoking a Process from Java

Process Server provides an API for dispatching a message directly to the engine. This API is available as an Enterprise Java Bean (EJB) on platforms that support EJB or as a plain Java API for deployment platforms without an EJB container.

The following illustration shows an overview of the external and standard implementations of My Role service endpoints.



The Process Server engine's Enterprise Java Bean (EJB) binding implements the following interface:

```
public interface org.activebpel.wsio.receive.IAeMessageQueue
{
   /**)
   * Delivers the message to the BPEL engine's message queue.
   *
   * @param aData
   * @param aContext
   * @return IAeWebServiceResponse
   * @throws RemoteException
   * @throws AeRequestException
   */
     queueReceiveData
     (org.activebpel.wsio.IAeWebServiceMessageData aData,
          org.activebpel.wsio.receive.IAeMessageContext aContext)
     throws RemoteException,
     org.activebpel.wsio.receive.AeRequestException;
```

where:

- `IAeMessageQueue` is name of the interface. The package is `org.activebpel.wsio.receive`.
- `IAeWebServiceResponse` is the interface for the data or fault that can be returned from the calling service.
- `queueReceiveData` is a method that a service endpoint can invoke passing in context and data.
- `IAeWebServiceMessageData` contains the data for the inbound message.
- `IAeMessageContext` contains the contextual information for the request.

Steps to Invoke a Process using plain Java or EJB API

Step 1: Include required .jar files on your java project classpath

The API is contained in the following `.jar` files. If using the plain Java API, `only ae_wsio.jar` is required, if using the EJB implementation, both are required in the classpath.

- `ae_wsio.jar` contains the core interfaces used to send messages to the engine.
- `ae_awfwsio.jar` contains the required interfaces for interacting with the Process Server engine EJB.

Step 2: Lookup the EJBHome

From Java code, lookup the `EJBHome` for the `AeMessageQueueBean` from JNDI context and create an instance of the bean. For example:

```
InitialContext ctx = new InitialContext();
object home = ctx.lookup("ejb/AeMessageQueueBean");
Home = (AeMessageQueueHome)
portableRemoteObject.narrow(home,AeMessageQueueHome.class);
eMessageQueueRemote msgQueueBean = mHome.create();
```

Step 3: Create an instance of AeMessageContext.

The `AeMessageContext` object contains the relevant details about the My Role partner link from the BPEL process. These details make up the context for the inbound messages sent to the engine. and are:

AeMessageContext Parameters	Description
setProcessName	Process `Qname`. The `Qname` includes the namespace plus the local part of the process name. In a BPEL source file, this is the target namespace. **Note**: You can specify a service name instead of a process/partnerlink name.
`setPartnerLinkName`	Partner link name from the BPEL process
`setServiceName`	Optional. As an alternative to setting the process and partner link name, you may specify the My Role (Process Consumer) service name. This name is in the Process Deployment Descriptor Editor on the Partner Links page and is also on the Service Definitions page in the Process Console. This property is mutually exclusive with the process name and only one or the other may be set on the context.
`setOperation`	Operation from the `portType` of the partner link type defined for My Role
`setProcessVersion`	*Optional*. Messages are sent to the current process if a process version is not specified.
`setPrincipal`	*Optional*. This value is used by the engine for performing lookups on deployed partner definitions when using the "Principal" endpoint type. Identifies the principal name value for inbound messages. Processes that include human tasks also use this value to identify the process initiator.

Example code for creating a new message context:

```
// Create the message context object for the request
/ containing the routing information for the process
eMessageContext context = new AeMessageContext();
context.setProcessName(new    QName(aProcessNamespace,aProcessName));
context.setPartnerLink(aPartnerLink);
context.setOperation(aOperation);
```

Step 4: Create an instance of AeWebServiceMessageData

The context details are as follows:

AeWebServiceMessageData Parameters	Description
Constructor QName	The QName specified in the constructor corresponds to the QName of the WSDL message associated with the input message
setData	Each message part is defined in this parameter. Message parts must be primitive types like strings, integers, or XML passed as DOM Document object. Complex types get passed as documents. The XML document must conform to the schema definition of the complex type. If a message has correlated properties in it, the engine extracts these properties using the property aliases and routes the message to the correct process instance.

WSDL Message for the message data object:

```
<wsdl:message name="creditInformationMessage">
    <wsdl:part name="part1" element="tns:LoanProcessRequest" />
    <wsdl:part name="part2" element="tns:LoanProcessAddress" />
</wsdl:message>
```

Example code for creating message data object:

```
AeWebServiceMessageData data = new AeWebServiceMessageData  (new
      QName("http://tempuri.org/services/loandefinitions","creditInformationMessage"));
   data.setData("part1", domOfLoanProcessRequest);
   data.setData("part2", domOfLoanProcessAddress);
```

Invoke the `queueReceiveData` method on the bean.

```
IAeWebServiceResponse response = msgQueueBean.queueReceiveData(data,context);
```

The context details are as follows:

IAeWebServiceResponse Parameter	Description
`getMessageData()`	Message data returned from the invoke. See the description of `AeWebServiceMessageData` for details.
`isFaultResponse()`	Return true if the response wraps a fault
`getErrorCode()`	Accessor for the error code `QName`
`getErrorString()`	Returns an error message associated with the fault or null if there is none
`getErrorDetail()`	Returns a stack trace or other detailed information associated with the fault or null if there was none
`getRootCause()`	Returns a Throwable that is the root cause of the error code
`getBusinessProcessProperties()`	Return a Map of (string) name/value pairs from the business process

CHAPTER 4

Custom Invoke Handlers

You can write Java custom handlers to invoke the service endpoints in BPEL processes.

Process Server's standard invocation framework works with the standard deployment of a BPEL process. In this scenario, a service endpoint is identified in the process deployment descriptor (PDD) with standard Partner Role attributes. To invoke the endpoint, Process Server uses its internal Web service framework based on Metro, the Apache SOAP engine. Metro marshals the data from a data structure into a SOAP message and sends the SOAP message to the service endpoint.

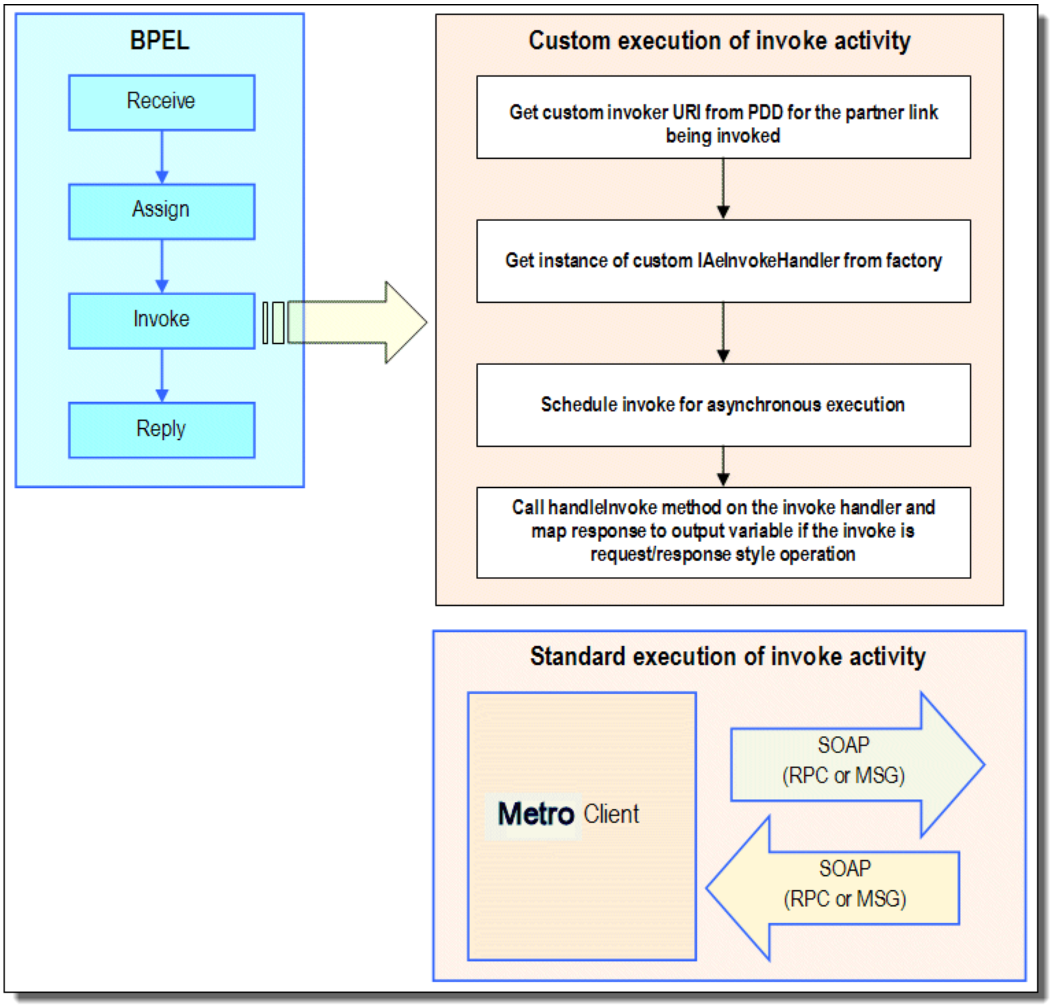
Instead of using the standard framework, you can implement a custom invoke handler to bypass the creation and transport of SOAP messages over Metro. This means that when an invoke activity executes, you can route the outbound message directly to a service.

A custom invoke handler is a Java class or EJB container loaded by Process Server that calls out to a Partner Role service endpoint when the service is invoked within a BPEL process. The handler implements the Process Server engine's custom invocation interface, `IAeInvokeHandler`. The BPEL process' deployment descriptor contains a custom invoke handler attribute for the partner role service.

For example, you may want to:

- Provide a more efficient technique for sending and receiving messages between processes and services on the same machine. You can create Java objects rather than SOAP messages.
- Provide messaging to an application that is not exposed as a Web service
- Take advantage of your own SOAP engine

The following illustration shows an overview of the custom and standard invocations of Partner Role service endpoints.


BPEL
Receive
Assign
Invoke
Reply
Custom execution of invoke activity
Get custom invoker URI from PDD for the partner link being invoked
Get instance of custom IAeInvokeHandler from factory
Schedule invoke for asynchronous execution
Call handleInvoke method on the invoke handler and map response to output variable if the invoke is request/response style operation
Standard execution of invoke activity
Metro Client
SOAP (RPC or MSG)
SOAP (RPC or MSG)

Referencing the Custom Invoke Handler Interface in a Java-Based Implementation

The Process Server's custom invocation handler implements the following interface:

```
package org.ActiveVOS.wsio.invoke;
import org.ActiveVOS.wsio.IAeWebServiceResponse;
public interface org.ActiveVOS.wsio.invoke.IAeInvokeHandler
{
   /**
    * Handles the invoke call. Query data will be null if
    * none was specified on the customInvokerUri.
    * @param aInvoke
    * @param aQueryData
    */
   public IAeWebServiceResponse
          handleInvoke
          (IAeInvoke aInvoke, String aQueryData);
}
```

where:

- IAeInvokeHandler is the name of the interface. The package is org.ActiveVOS.wsio.invoke. This interface imports org.ActiveVOS.wsio.IAeWebServiceResponse.
- IAeWebServiceResponse is the interface for the data or fault that is returned from the calling service.
- handleInvoke is the entry point into the service.
- aInvoke contains the service's message data and endpoint reference address data from the input message.
- aQueryData contains additional parameters that can be passed in from the invokeHandler attribute for the partner role in the process deployment descriptor

Code Sample

The following Java code snippet shows an implementation of the interface's handleInvoke method. In this example, a loan approval service receives an amount for a loan approval and returns an "accepted" or "denied" message based on the maximum loan amount allowed.

```
public IAeWebServiceResponse handleInvoke(IAeInvoke aInvoke,
       String aQueryData)
{
  // Parse the query data section of the uri to create the
  // message type qname,
  // then create the appropriate response and return.
  Map data = parse( aQueryData );
  Integer maxLoan = new Integer(
         (String)data.get(URI_PARM_MAXLOAN));
  String ns = (String)data.get(URI_PARM_MSG_NAMESPACE);
  String lp = (String)data.get(URI_PARM_MSG_LOCALPART);
  QName messageType = new QName(ns, lp);

  // Get input message data.
  IAeWebServiceMessageData input = aInvoke.getInputMessageData();
  Map input_data = input.getMessageData();
  Integer amount = (Integer)input_data.get(MSG_AMOUNT);

  // Determine response.
  String approval = APPROVER_DENY_RESP;
  if ( validateLoan(amount, maxLoan) )
       approval = APPROVER_ACCEPT_RESP;

  // Build and return response message. Note that in this example,
  // the implementation is for a request/response invocation.
  // For a request-only invocation, you would not need to
  // setMessageData.
  Map output_data = new HashMap();
  output_data.put(MSG_ACCEPT, approval);
  AeWebServiceMessageData msgData = new
    AeWebServiceMessageData(messageType, output_data);
  AeInvokeResponse response = new AeInvokeResponse();
  response.setMessageData( msgData );
  return response;
}
```

Adding the Custom Invoke Handler Attribute to the Process Deployment Descriptor

To bypass the Process Server's standard invocation framework, you must add the custom invoke handler attribute to the partner role in the Process Deployment Descriptor (PDD) file.

The custom invoke handler attribute tells the engine to use a custom invocation for the partner service. Add the attribute, along with optional message parameters to pass into the service, and then deploy the business process archive (BPR) file.

Depending on your invocation implementation, you must specify the URI information for a Java class or EJB container.

Java Class Implementation

Use the Process Developer's PDD wizard to specify a Java invoke handler. The following syntax show how to include the custom invoke attribute in the PDD file for a Java class file.

```
<partnerLink name="qname">
  <partnerRole endpointReference="type"
   invokeHandler="java:org.ActiveVOS.rt.axis.bpel
    .MyInvokeHandler?parameters+
    ...
  </partnerRole>
</partnerLink>
```

Example:

```
invokeHandler="java:org.ActiveVOS.rt.axis.bpel.
    AeApproverInvokeHandler?type-namespace=
       http://tempuri.org/services/loanapprover&
           type-localpart=approvalMessage&amp;maxLoan=1000000">
```

EJB Implementation

The following syntax show how to include the custom invoke attribute in the PDD file for an EJB.

```
<partnerLink name="qname">
  <partnerRole endpointReference="type"
   invokeHandler="ejb:jndiLocation/
    MyInvokeHandler?parameters+
    ...
  </partnerRole>
</partnerLink>
```

The `jndiLocation` defines the context that specifies where to look for the custom invoke handler.

Example:

```
invokeHandler="ejb:ejb/AeApproverInvokeHandler?type-namespace=
    http://tempuri.org/services/loanapprover&
           type-localpart=approvalMessage&maxLoan=1000000">
```

Packaging the Custom Invoke Handler as an EJB

If your custom invoke handler is an EJB, you must package the EJB and other files into an EAR archive.

Packaging includes the following:

- JAR files for the custom invoke handler
- Deployment descriptors

Service Implementation Bean class and their dependent classes. These files are `ae_wsio.jar`, `ae_awfwsio.jar`, and `qname.jar`.

Sample EAR package

```
META-INF
      Manifest.mf
          jboss-app.xml
          application.xml
```

```
name.jar
customInvokeSample.jar
ae_awfwsio.jar
```

The following table describes the contents of the sample EAR file.

Web Application File	Description
META-INF `Manifest.mf`	Contains meta-information about the EAR file and can also contain information about the other files that are packaged in the archive. This file name and pathname are required for an EAR file.
`jboss-app.xml`	A representative configuration file from a J2EE server, in this case JBoss
`application.xml`	Descriptor required for the EAR file
`customInvokeSample.jar`	Custom Invoke handler referencing the `IAeInvokeHandler` implementation
`ae_wsio.jar`	Contains the core interfaces used to send messages to the engine. The file is located in the Process Server EAR file.
`ae_awfwsio.jar`	Contains the required interfaces for interacting with the Process Server EJB. The file is located in the Process Server EAR file.
`qname.jar`	Required for handling WSDL. The file is located in the Process Server EAR file.

Deploying Custom Invoke Handler Files

Follow these instructions for deploying a custom invoke handler to Process Server.

Process Server for Apache Tomcat:

1. Copy your JAR file to `[Tomcat installation folder]/lib`.
2. Deploy your BPR file using the Process Console or Process Developer.
3. Restart Tomcat.

Process Server Application Servers that Support EARs:

1. Copy your EAR file into your J2EE application server's deploy server. For example in JBoss, deploy to`[JBoss installation folder]/server/ActiveVOS server/deploy`.
2. Deploy your BPR file using the Process Console or Process Developer.

Note: You can deploy the Java class to the Process Server, but you must ensure that the class is on the classpath of the server EAR file.