Integrating WebSphere Service Registry and Repository with WebSphere Process Server and WebSphere ESB

This IBM® Redpapers™ publication describes how you can integrate IBM WebSphere® Service Registry and Repository with IBM WebSphere Enterprise Service Bus and IBM WebSphere Process Server to create a flexible and robust infrastructure for the execution of SOA solutions.

Product descriptions

We demonstrate the scenarios in this paper by using the following product versions:

- IBM WebSphere Service Registry and Repository V6.3
- IBM WebSphere Enterprise Service Bus V6.2
- IBM WebSphere Process Server V6.2
IBM WebSphere Service Registry and Repository

IBM WebSphere Service Registry and Repository (WSRR) is a system for storing, accessing, and managing information (commonly referred to as service meta data) that is used in the selection, invocation, management, governance, and reuse of services in a successful service-oriented architecture (SOA). In other words, it is where you store information about services in your systems, or in other organizations' systems, that you already use, plan to use, or want to be aware of for the future. For example, an application can check the WebSphere Service Registry and Repository just before invoking a service to locate the service instance best satisfying its functionality and performance needs. WebSphere Service Registry and Repository also plays a role in other stages of the SOA life cycle.

IBM Enterprise Service Bus

IBM WebSphere Enterprise Service Bus provides the capabilities of a standards-based enterprise service bus.

WebSphere ESB manages the flow of messages between service requesters and service providers. Mediation modules within the ESB handle mismatches between requesters and providers, including protocol or interaction-style interface and quality of service (QoS) mismatches. In a Service Component Architecture (SCA)-based solution, mediation modules are a type of SCA modules. The mediation modules perform a special role, and therefore have slightly different characteristics from other components that operate at the business level.

Mediation components operate on messages exchanged between service endpoints. In contrast to regular business application components, they are concerned with the flow of the messages through the infrastructure and not just with the business content of the messages. Rather than performing business functions, they perform routing, transformation, and logging operations on the messages.

IBM WebSphere Process Server

IBM WebSphere Process Server provides a standards-based business process engine, using the power of WebSphere Application Server. It also includes the same technology as WebSphere Enterprise Service Bus, providing the same enterprise service bus capabilities.
WebSphere Process Server is a high-performance business engine that enables processes to meet business goals by providing a first class managed runtime environment for business process applications. It allows the deployment of standards-based business integration applications in a service-oriented architecture (SOA), which takes everyday business applications and breaks them down into individual business functions and processes, rendering them as services.

WebSphere Process Server enables the deployment of processes that span people, systems, applications, tasks, rules, and the interactions among them. It supports both long-running and short-running business processes, providing transaction rollback-like functionality for loosely coupled business processes.

Value proposition

Successfully implementing an SOA requires applications and an infrastructure that can support the service-oriented architecture principles. Applications can be enabled by creating service interfaces to existing or new functions that are hosted by the applications. The service interfaces should be accessed using an infrastructure that can route and transport service requests to the correct service provider. As organizations expose more and more functions as services, it is vitally important that this infrastructure supports the management of SOA on an enterprise scale.

The enterprise service bus (ESB) is a middleware infrastructure component that supports the implementation of SOA within an enterprise.

Enterprise service bus

The need for an ESB can be seen by considering how it supports the concepts of SOA implementation by:

- Decoupling the consumer's view of a service from the actual implementation of the service
- Decoupling technical aspects of service interactions
- Integrating and managing services in the enterprise
The ESB is more than just a transport layer. It must provide mediation support to facilitate service interactions (for example, to find services that provide capabilities for which a consumer is asking or to take care of interface mismatches between consumers and providers that are compatible in terms of their capabilities). It must support a variety of ways to get on and off the bus, such as adapter support for existing applications or business connections, which enable external partners in business-to-business interaction scenarios. To support these different ways to get on and off the bus, it must support service interaction with a wide variety of service endpoints. It is likely that each endpoint will have its own integration techniques, protocols, security models, and so on. This level of complexity should be hidden from service consumers. They need to be offered a simpler model. In order to hide the complexity from the consumers, the ESB is required to mediate between the multiple interaction models that are understood by service providers and the simplified view that is provided to consumers.

During runtime execution, WebSphere Service Registry and Repository can be dynamically accessed to select a service provider, or to dynamically enforce an invocation policy. It manages information that enables dynamic binding of service consumers to service providers and allows the infrastructure to enforce registered policies.

Figure 1 show how WebSphere Service Registry and Repository is used for runtime integration.
Key integration features

Mediation flows deployed to WebSphere Enterprise Service Bus and WebSphere Process Server can use WebSphere Service Registry and Repository as a dynamic lookup mechanism providing information about:

- Service endpoints
- Mediation policies

Service endpoints

When you develop an Service Component Architecture module that needs to access service endpoints from WebSphere Service Registry and Repository, you can include the Endpoint Lookup mediation primitive in the mediation flow. At run time, the Endpoint Lookup mediation primitive obtains service endpoints from the registry.

Mediation policies

You can also use WebSphere Service Registry and Repository to store mediation policy information. Mediation policies can help you control service requests by dynamically overriding module properties. If WebSphere Service Registry and Repository contains an SCA module and attached mediation policies, the mediation policies have the potential to override module properties. If you want different mediation policies to apply in different contexts, you can create mediation policy conditions.

**Note:** Mediation policies are concerned with the control of mediation flows in terms of mediation functions, such as routing, transformation, conversion, or logging, rather than non-functional quality characteristics, such as transactions, security, and so on.

When you develop an SCA module that needs to make use of a mediation policy, you can include a Policy Resolution mediation primitive in the mediation flow. At run time, the Policy Resolution mediation primitive obtains mediation policy information from the registry.
JKHL Enterprises overview

In this paper, we demonstrate the integration features of WebSphere Service Registry and Repository by describing integration scenarios at the fictitious company called JKHL Enterprises (JKHLE).

JKHLE has a business challenge of having to access the existing operational services, such as eligibility service, for continuously changing business circumstances. It is one of JKHLE's stated goals: To govern and adapt service interactions between service consumers and service providers rather than modify software application assets.

JKHLE's SOA governance team, consisting of IT architects, has its own goals in architecting a solution. Two of their notable goals are to:

- Access a business service or business data more flexibly.
- Govern service interactions more centrally.

JKHLE's SOA governance team has decided to use the mediation policies and endpoint lookup features of WebSphere Process Server / WebSphere ESB to control the message flows between service consumers and service providers. Additionally, JKHLE's SOA governance team has decided to use WebSphere Service Registry and Repository as its central store of master meta data repository for service interactions.

Endpoint lookup integration scenario

This scenario describes how JKHL Enterprises uses an endpoint lookup mediation primitive in a mediation flow executing on WebSphere ESB to retrieve a service endpoint stored in the organization’s service repository hosted on WebSphere Service Registry and Repository.

Scenario overview

JKHL Enterprises currently has an Account Creation service running in their production environment. This service has recently been upgraded to expose an additional verifyCreation operation, as shown in Figure 2 on page 7. This minor upgrade of the service is backwards compatible with the first version.
JKHL Enterprises uses WebSphere Service Registry and Repository as their service repository. To shield consumer applications from changes to the Account Creation service, JKHL utilizes a mediation flow on the ESB, which looks up the active endpoint of the Account Creation service in WebSphere Service Registry and Repository.

Once version 1_1 of the Account Creation service is deployed and available, the endpoint for version 1_0 will be classified as offline. All service consumer requests for the Account Creation service will be directed to the endpoint for version 1_1, enabling version 1_0 to be deprecated with no impact on service consumers.

**Note:** There is no change to the namespace of the service in this minor upgrade scenario.

**Integration issues**

JKHL Enterprises uses dynamic endpoints for service calls in WebSphere Enterprise Service Bus.

Service definitions are referenced in the mediation flow using dynamic endpoints created with an endpoint lookup mediation primitive. In this endpoint lookup scenario, WebSphere Service Registry and Repository is accessed and retrieves a single matching endpoint for the latest version of the createAccount operation of the AccountCreation service.
Solution overview

The endpoint lookup solution shown in Figure 3 shows how the mediation flow is constructed by using an endpoint lookup primitive to access WebSphere Service Registry and Repository.

Performing the integration

This section describes how to create a mediation module containing an endpoint lookup mediation primitive using WebSphere Integration Developer. It also explains how to configure access from WebSphere ESB or WebSphere Process Server to WebSphere Service Registry and Repository, including the configuration of security certificates.

Finally, it describes how to unit test the mediation module using the Integration Test Client within WebSphere Integration Developer.
Configure access to WebSphere Service Registry and Repository

First, it is necessary to establish communication between WebSphere ESB / WebSphere Process Server and WebSphere Service Registry and Repository. At JKHLE, security is enabled for both WebSphere ESB and WebSphere Service Registry and Repository.

To configure access, perform the following steps:

1. Go to the Servers view in WebSphere Integration Developer.
2. Right-click WebSphere Process Server v6.2 at localhost and select Start from the context menu. Wait for the server to start.
3. When the server has started, right-click WebSphere Process Server v6.2 at localhost and select Administration → Run administrative console.
4. Log in with a user ID and use the password admin.

Note: Because a Web Services Business Process Execution Language (WS-BPEL) process will be deployed in a later scenario, we configure WebSphere Process Server for all examples; however, this first scenario could instead be deployed to WebSphere ESB Server V6.2 instead.

Note: At the time of writing, it was necessary to apply APAR IC62037 to WebSphere ESB or WebSphere Process Server in order to make this scenario work. For information about this APAR, refer to the following address:
http://www.ibm.com/support/docview.wss?uid=swg1IC62037
5. In the left navigation pane, expand **Service integration** and click **WSRR definitions**, as shown in Figure 4.

![Integrated Solutions Console](Image)

*Figure 4  Navigating to WebSphere Service Registry and Repository definitions*
6. Click the **New** button to create a new WebSphere Service Registry and Repository definition, as shown in Figure 5.

![Integrating Solutions Console](image_url)

**Figure 5** Creating a new WebSphere Service Registry and Repository definition
7. Enter the following values into the New WSRR definition section (see Figure 6):
   – WSRR definition name: ITSO_WSRR
   – Description: Service Registry for ITSO
   – Timeout of cache: 300

8. Click **Apply**.

9. Under Additional Properties, enter the Registry URL:

   https://localhost:9446/WSRR6_3/services/WSRRCoreSDOPort
Integrating WebSphere Service Registry and Repository with WebSphere Process Server and WebSphere ESB

Configuring security certificates

If the environment does not have security enabled, the following steps are not required:

1. Set the Authentication alias to CommonEventInfrastructureJMSAuthAlias.
2. Click **SSL Configurations**, as shown in Figure 7.

**Note:** In our environment, WebSphere Process Server and the WebSphere Service Registry and Repository server are running on the same machine where both WebSphere Process Server and WebSphere Service Registry and Repository are secured.

The WebSphere Process Server server admin console is available on port 9043.

The WebSphere Service Registry and Repository HTTP port has been set to 9446.
3. Click the default SSL configuration, **NodeDefaultSSLSettings**, as shown in Figure 8.

![Figure 8 SSL configurations](image)

4. Click **Key stores and certificates**.
5. Click **NodeDefaultTrustStore**.
6. Click **Signer certificates**.
7. Click the **Retrieve from port** button, as shown in Figure 9.

![Figure 9 Retrieve from port](image)
8. Complete the General Properties details as follows:
   - Host: localhost
   - Port: 9446
   - Alias: wsrr_v63

9. Click **Retrieve signer information**, as shown in Figure 10.

![WSRR definitions](Image)

**Figure 10** Retrieve signer information

10. Click **OK** and then **Save**.

11. To test the new connection, click **ITSO_WSRR** and click the **Test connections** button. Figure 11 shows the message that is displayed for a successful connection.

![Messages](Image)

**Figure 11** Successful WebSphere Service Registry and Repository test connection
In our use case configuration, the WebSphere Service Registry and Repository V6.3 instance has its security enabled. For the WebSphere ESB or WebSphere Process Server server instance to communicate with the WebSphere Service Registry and Repository instance using the Secure Socket Layer protocol, perform steps 2 through 10 in this section on the WebSphere ESB or WebSphere Process Server server instance (we performed these steps).

More information about configuring secure connections can be found in the WebSphere Application Server InfoCenter at the following address:


Create the endpoint lookup mediation flow

An endpoint lookup mediation primitive will be used to access WebSphere Service Registry and Repository and retrieve a single matching endpoint from the registry based on the namespace of the Account Creation service and a classification that indicates if an endpoint is active or offline.

Setting up the projects
To set up the projects, perform the following steps:

1. Create the integration solution.
   a. In the Business Integration perspective, click New to create a new integration solution, as shown in Figure 12.

   ![Creating a new integration solution](image)

   Figure 12 Creating a new integration solution

   b. Enter the integration solution name WSRRIIntegration and click Finish.
2. Create a mediation module.
   a. Click **New**, as shown in Figure 13.

*Figure 13  Creating a new integration project*
b. Click **Mediation Module**, as shown in Figure 14.

*Figure 14  Creating a mediation module*
c. Enter the mediation module settings shown in Figure 15.
   - Module name: DynamicEndpointProject
   - Use default location: Selected
   - Target runtime environment: WebSphere Process Server V6.2
   - Create mediation component: Selected
   - Name: DynamicEndpointMediation
   
   Click **Next**.

   ![Figure 15 Entering the mediation module settings](image)

   ![New Mediation Module](image)

   **Create a Mediation Module**

   **Use a mediation module to integrate and connect services. A mediation module can contain mediation flows and can be deployed on WebSphere ESB or WebSphere Process Server.**

   **Module name:** DynamicEndpointProject

   **Target runtime environment:** WebSphere Process Server v6.2

   **Create mediation component:**

   **Name:** DynamicEndpointMediation

   **Open module assembly diagram**

   ![New Mediation Module](image)

   

   d. Click **Select an integrated solution** and click **Finish**.

3. Create and populate a library project:
   a. Create a new library called JKHLE_Library and associate the library with the integration solution.
b. Import the WSDL and schema files for version 1_0 and 1_1 of the Account Creation service into the library by right-clicking the JKHLE_Library project and selecting Import → General → File System. Click Next.

c. Navigate to the directory containing the WSDL and XSD files.

d. Select the required files and click Finish.

e. The library should now contain the entities shown in Figure 16.

Figure 16  Library contents
4. The DynamicEndpointProject uses business object and service definitions from the JKHLE_Library, so a dependency must be added.
   a. In the Business Integration view, expand the **DynamicEndpointProject** and double-click **Dependencies**, as shown in Figure 17, to open the Dependencies editor.

   ![Figure 17 Selecting the Dependencies option](image)

   **Figure 17 Selecting the Dependencies option**

   b. Click the **Add** button in the Libraries section and select **JKHLE_Library**, as shown in Figure 18, and click **OK**.

   ![Figure 18 Selecting the library](image)

   **Figure 18 Selecting the library**
c. Save the changes.

**Implementing the mediation module**

To implement the mediation module, perform the following steps:

1. Construct the DynamicEndpointProject mediation module.
   a. Go to the DynamicEndpointProject Assembly Diagram editor.
   b. Right-click the `DynamicEndpointMediation` component and select `Add → Interface` from the context menu.
   c. Select `AccountCreationV1_1`, as shown in Figure 19, and click **OK**.

![Image of Add Interface dialog box](image)

*Figure 19  Adding an interface*

**Note:** The `AccountCreationV1_1` interface is the new minor version upgrade of JKHLE’s Account Creation service. This upgrade is backwards compatible with the previous version of `AccountCreationV1_0`. 
d. In the Business Integration view, expand JKHLE_Library → Web Service Ports.

e. Drag and drop the AccountCreationV1_1_ProductionPort Web service port onto the assembly diagram canvas.

f. Select SOAP1.1/HTTP as the transport protocol, as shown in Figure 20, and click OK.

Figure 20  Selecting the transport protocol
g. Wire the DynamicEndpointMediation component to the AccountCreationV1_1Import1 component, as shown in Figure 21.

![Figure 21 Wiring the interface](image)

h. Click **OK** in the Add Wire window to add a matching reference on the DynamicEndpointMediation component.

i. Right-click the **DynamicEndpointModule** and select **Generate Export** → **SCA Binding**, as shown in Figure 22.

![Figure 22 Generating an export](image)

j. Save all your changes in the Assembly Diagram editor. It will now appear as shown in Figure 23.

![Figure 23 DynamicEndpointModule with SCA Export](image)

k. In the Assembly Diagram editor, right-click the **DynamicEndpointMediation** component and select **Generate Implementation**.

l. Click **OK** in the Generate Implementation window. The Mediation Flow Editor opens.
2. Create the mediation flow.
   a. In the Operation connections section, wire the createAccount operation of the AccountCreationV1_1 interface to the createAccount operation of the AccountCreationV1_1Partner reference, as shown in Figure 24.

   ![Wiring the operations](image)

   **Figure 24  Wiring the operations**

   b. Right-click the `createAccount:AccountCreationV1_1Partner` callout shown in Figure 25 and select **Show in Properties**.

   ![Selecting the callout](image)

   **Figure 25  Selecting the callout**
c. In the Details tab of the Properties view, ensure that the **Use dynamic endpoint if set in the message header** property is enabled, as shown in Figure 26.

![Figure 26 Use a dynamic endpoint](image)

**Note:** It is important that the **Use dynamic endpoint if set in the message header** property is enabled for the Callout because only then will WebSphere ESB inspect the `/headers/SMOHeader/Target/address` element and route the message to this endpoint instead of the statically wired Import, which functions like a default static endpoint in this scenario.

d. From the palette, add an Endpoint Lookup mediation primitive to the mediation flow and rename it to CreateAccountLookup. You might have to expand the Routing section to locate it, as shown in Figure 27 on page 27.
e. From the palette, add an Stop mediation primitive to the mediation flow. You might have to expand the Error Handling section to locate it.

f. Establish the wiring of the request flow (Figure 28).
   i. Wire the output terminal of the createAccount:AccountCreation input to the input terminal of the CreateAccountLookup primitive.
   ii. Wire the output terminal of the CreateAccountLookup primitive to the input terminal of the createAccount:AccountCreationV1_1Partner callout.
   iii. Wire the noMatch terminal of the CreateAccountLookup primitive to the input terminal of the Stop node.
g. Right-click the **CreateAccountLookup** mediation primitive and select **Show in Properties**.

h. Go to the Details tab of the Properties view to configure the CreateAccountLookup primitive, as shown in Figure 29.

i. For the Name field, click **Browse**.

ii. In the Interface Selection panel, select the **AccountCreationV1_1** interface and click **OK**.

iii. Remove the value in the Name field.

**Note:** By default, when the endpoint lookup mediation queries WebSphere Service Registry and Repository, it will match using the port type (Name) and namespace.

In this scenario, two instances of the Account Creation service are deployed to WebSphere Service Registry and Repository with the same namespace and we wish to locate the active service by using the **LifecycleDefinition** classification, not the port type, which is unique.

iv. Enter ITSO_WSRR as the Registry Name.

v. Set the Match Policy to **Return first matching endpoint and set routing target**, as shown in Figure 29.
Note: The Version property can either hold the version of a port type (interface) or the version of an SCA module published to WebSphere Service Registry and Repository.

If you are using a port type, the registry is searched for services implementing that port type and version.

If you are using versioned SCA modules, you have to set the Match Policy to Return endpoint matching latest compatible service version. You then have to set the Version as well as Module and Export properties.

JKHLE is not changing the version of the Account Creation service; instead, they are bringing online a minor upgrade to the existing service. Therefore, this scenario does not include use of the Version property, which remains unspecified.

a. To configure the classification query:
   i. Click Details and then the Advanced tab in the Properties view.
ii. In WebSphere Service Registry and Repository, copy the query string for the active endpoint that is classified as online (Figure 30).

![Figure 30 Copying the classification query](image)

iii. Click **Add** next to the Classifications table to specify a new property.
iv. Paste the query string into the classification properties, as shown in Figure 31, and click Finish.

![Figure 31 Pasting the classification query string](image)

v. The settings in the Advanced tab should match the ones shown in Figure 32.

![Figure 32 Completed classification query](image)

b. In the Mediation Flow Editor, click the **Response: createAccount** tab to implement the response flow.

c. From the palette, add a Stop mediation primitive to the mediation flow.
d. Establish the wiring of the response flow, as shown in Figure 33.
   i. Wire the output terminal of the createAccount:
      AccountCreationV1_1Partner callout response to the input terminal of
      the input response.
   ii. Wire the fail terminal of the callout response to the Stop node.

![Figure 33 Wiring the response flow](image)

iii. Select File → Save All.
Testing the mediation flow

To test the mediation flow, we have two available endpoints for the Account Creation service defined in WebSphere Service Registry and Repository, as shown in Figure 34. These two SOAP Service Endpoints have been defined in the production environment and have been activated, and therefore have an online state.

![SOAP Service Endpoints Table]

*Figure 34  Service endpoints*
WebSphere Service Registry and Repository provides a governance life cycle for endpoints, as shown in Figure 35, which we utilize in this scenario.

![Figure 35   Endpoint life cycle](image)
Click the **Revoke From User** button to make the SOAP Service Endpoint V1_0 of the Account Creation service unavailable, as shown in Figure 36.
The SOAP Service Endpoint V1_0 is now in the Offline state (Figure 37).

**Note:** When revoking the endpoint, the Governance State is changed to Offline. This repeated in the list of Classifications.

![Governance State Table](image)

*Figure 37  Offline endpoint*

The upgraded version of the service has a SOAP Service Endpoint with a state of *Online*, so this is the single endpoint that will be retrieved from WebSphere Service Registry and Repository by the endpoint lookup mediation component.

Using the Integration Test Client, we can now test the mediation flow by performing the following steps:

1. Go to the Servers view and ensure that the WebSphere Process Server V6.2 at localhost is started. If necessary, start the server.

2. Right-click **WebSphere Process Server v6.2 at localhost** and select **Add and Remove Projects**.

3. Add DynamicEndpointProjectApp, AccountCreationV1_0EAR, and AccountCreationV1_1EAR to the server.

4. Click **Finish** and wait for the modules to be deployed.
5. In the Business Integration view, right-click `DynamicEndpointProjectApp` and select **Test → Test Module**, as shown in Figure 38. The Integration Test Client opens.

*Figure 38  Selecting Test Module*
6. Ensure that, in the Detailed Properties section of the Integration Test Client, the Component is set to DynamicEndpointModuleApp, the Interface is set to AccountCreationV1_1, and the Operation is set to createAccount, as shown in Figure 39.

![Figure 39 Test Detailed Properties](image)

7. Click **Continue** (Figure 40).

![Figure 40 Continue button](image)
8. In the Deployment Location window, select **WebSphere Process Server v6.2 at localhost** and click **Finish**, as shown in Figure 41.

![Figure 41 Selecting the deployment location](image)

9. Accept all the defaults in the User Login window and click **OK** to start the test (Figure 42).

![Figure 42 User Login window](image)
10. The DynamicEndpointLookup module uses WebSphere Service Registry and Repository to locate the relevant endpoint and invokes the Account Creation service at this endpoint, as shown in Figure 43.

![Test event trace](image)
11. In the Events window of the Integration Test Client, click \texttt{CreateAccountLookup} under the Fine-Grained Trace section, as shown in Figure 44.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure44.png}
\caption{Selecting the lookup}
\end{figure}
12. Examine the headers/SMOHeader/Target/address value in the Mediation Message window and note that the endpoint for the AccountCreationV1_1 service (https://localhost:9443/AccountCreationV1_1/services/AccountCreationServiceV1_1_ProductionPort) has been retrieved from WebSphere Service Registry and Repository (Figure 45).

![Figure 45 Verifying the endpoint lookup](image)

The extract from the system logs (Example 1) shows the effect of executing the endpoint lookup mediation before and after the deployment of the upgraded service.

**Example 1  System log extract**

```
[7/07/09 04:50:36:793 EST] 000000d9 ExportRuntime I   processMessage
[7/07/09 05:06:12:168 EST] 000000d9 ExportRuntime I   processMessage
[7/07/09 05:06:19:996 EST] 000000d9 ExportRuntime I   processMessage
```
Mediation policy integration scenario

This scenario describes how JKHL Enterprises uses a policy resolution primitive in a mediation flow executing on WebSphere ESB to look up, retrieve, and enforce policies stored in WebSphere Service Registry and Repository.

Scenario overview

JKHLE’s IT developers have decided to use WebSphere Integration Developer tooling to develop Service Component Architecture modules consisting of mediation message flows to mediate service interactions. They intend to deploy these mediation module logic components on to WebSphere Process Server / WebSphere ESB run times.

Additionally, JKHLE’s IT developers have decided to identify the points of variability in JKHLE’s requirements and promote them as mediation module properties so that these points of variabilities become dynamic and visible to the run time. Since dynamic policies can be overridden at run time using mediation policies stored in WebSphere Service Registry and Repository. This approach allows IT developers to build a solution that enables central governance of service interactions using service meta data rather than application logic embedded into application programs.

Integration issues

JKHL Enterprises will be dynamically controlling a mediation flow on a message by message basis. This scenario shows how to develop a mediation module that incorporates a policy resolution primitive to look up, retrieve, and enforce policies stored in WebSphere Service Registry and Repository.
When SCA modules consisting of mediation, routing, transformation, and event logic components are deployed to a WebSphere ESB / WebSphere Process Server run time, the dynamic mediation module properties encapsulating properties and their values become visible to IT administrator/operational professionals on the administrative console. The IT administrator/operational staff now has two choices to control the service interactions occurring between consumers and providers:

- Changing the promoted properties that represent the points of variability using the administrative console
- Updating the promoted properties representing the points of variability visible in the administrative console by dynamically overriding their values with the values retrieved from mediation policies stored in WebSphere Service Registry and Repository

The first approach leads to changing the configuration of that mediation flow logic that affects all the flows that reference these promoted properties. The change is permanent within the deployed application, and the change takes place without having to restart the server.

The second approach leads to updating the promoted property through a mediation policy retrieved from WebSphere Service Registry and Repository that has an effect on the lifetime of current invocation of the flow only. In this implementation approach, a policy resolution primitive needs to be at the start of mediation flow in order for all the rest of primitives that follow to be dynamically configurable.

The policy resolution primitive can conditionally load WS-Policy formatted mediation policies to override promoted properties, depending on the message context. This results in a flexible mechanism to construct a single mediation flow that can exhibit multiple behaviors, based on the number of mediation policies available to it in WebSphere Service Registry and Repository on a per message context basis.

The mediation module accepts account creation requests containing customer address information and account creation information. The request and response message may be logged and the customer name is transformed depending on the message criteria, before being passed to the eligibility Web service. A transformation primitive generates a customer name as a concatenation of customer nationality, first name, and last name.
The logging mediation primitive’s “Enabled” property will be promoted, which will allow logging to be controlled either administratively or by using a policy. The transformation primitive’s “Mapping file” property will also be promoted, allowing different transformations to be specified. The Policy Resolution mediation primitive will query the WebSphere Service Registry and Repository to determine which promoted property values to override for each incoming message.

Solution overview

The mediation policy solution shown in Figure 3 on page 8 shows how the mediation flow will be constructed using a policy resolution primitive to retrieve and enforce policy information sourced from WebSphere Service Registry and Repository.
Performing the integration

The section describes how to create a mediation module containing a policy resolution mediation primitive using WebSphere Integration Developer.

It explains how to:
- Create a mediation module to communicate with a Web service.
- Implement a mediation flow component using a Policy Resolution mediation primitive and promote mediation primitive properties.
- Override mediation primitive properties using the Integrated Solutions Console.
- Import the mediation module into WebSphere Service Registry and Repository.
- Create policies in WebSphere Service Registry and Repository and attach them to Mediation Modules.
- Create gate conditions in WebSphere Service Registry and Repository.

Finally, it describes how to deploy and test the mediation module using the Integration Test Client within WebSphere Integration Developer.

Create the policy mediation flow

JKHLE’s business requirements dictate:
- Dynamic request and response flow logging of service interactions for a special and high risk/value customer with customerNumber = “1000”.
- Service requests originating from customers residing in Germany or UK or USA require country specific infrastructure transformations for services being provided in the Eligibility service provider.
- Service requests originating from customers residing in countries other than Germany or UK or USA require USA country specific infrastructure transformations for services being provided in the Eligibility service provider.
- In the future, additional default country specific infrastructure transformations for services being provided in the Eligibility service provider may be added in addition to USA.
JKHLE’s IT implementation requirements dictate control of points of variability to occur at a central governance point and should default to a disabled state in the SCA component of the mediation module being developed.

- The logging of request and response flows are disabled for the promoted mediation module properties in the SCA mediation module.
- No transformations exist for services being provided in the Eligibility service request flows.
- The default behavior of the Eligibility service is to respond with a Boolean success return value for all service requests in the initial implementation.

### Setting up the projects

To set up the projects, perform the following steps:

1. Create a new mediation component.
   - Go to the Business Integration view. Expand **AccountCreationModule** and double-click the **Assembly Diagram** in the Business Integration view.
   - From the palette of the Assembly Diagram editor, select **Mediation Flow** and click the canvas to create a new mediation component.
   - Rename the Component1 mediation component to **AccountCreationModule**.
   - Right-click **AccountCreationModule** and select **Add → Interface**.
   - Select the **AccountCreationV1_0** interface and click **OK**.
   - Right-click **AccountCreationModule** and select **Add → Reference**.
   - Select the **EligibilityV1_0** interface and click **OK**.
2. Create an import for the EligibilityService.
   a. In the Business Integration view, expand **AccountCreationLibrary** → **Web Service Ports** and drag and drop **EligibilityServiceV1_0_ProductionPort** onto the assembly diagram.
   b. Select **Import with Web Service Binding** in the Component Creation panel and click **OK**. Leave the radio button selected for **SOAP1.1/HTTP using JAX-RPC** in the Transport Selection window and click **OK**.
   c. Wire the **AccountCreationModule** component to the **EligibilityV1_0Import1** component (Figure 47).

![Figure 47 AccountCreationModule assembly diagram](image)

3. Save your changes.

**Implementing the mediation flow component**

To implement the mediation flow component, perform the following steps:

1. In the Assembly Diagram editor, right-click the **AccountCreationModule** mediation component and select **Generate Implementation**.
2. In the Generate Implementation window, click **OK**. This will open the Mediation Flow Editor.
3. In the Mediation Flow Editor, wire the createAccount operation of the **AccountCreationV1_0** interface to the validate operation of the **EligibilityV1_0Partner** reference (Figure 48 on page 49).
Configuring the request flow

To configure the request flow, perform the following steps:

1. Add a Policy Resolution mediation primitive to the request flow and rename it to AccountCreationPolicy. You may have to expand the Routing section to see it.

2. Wire the out terminal of the input to the in terminal of the AccountCreationPolicy primitive.

3. Add a Message Logger mediation primitive to the request flow and rename it to LogRequest. You may have to expand the Tracing section to see it.

4. Wire the out terminal of the AccountCreationPolicy primitive to the in terminal of the LogRequest primitive.

5. Add an XSL Transformation mediation primitive to the request flow and rename it to RequestTransform. You may have to expand the Transformation section to see it.

6. Wire the out terminal of the LogRequest primitive to the in terminal of the RequestTransform primitive.

7. Wire the out terminal of the RequestTransform primitive to the in terminal of the Callout.

The request flow should now look like Figure 49.
8. Configure the AccountCreationPolicy mediation primitive.
   a. Right-click the **AccountCreationPolicy** mediation primitive and select **Show in Properties**.
   b. On the Details tab of the Properties view, leave the default setting for the Registry Name.

   **Note:** The <use default registry> property value will result in the Policy Resolution mediation primitive using the default registry defined in the WebSphere ESB run time. In this scenario, the registry has already been defined for you in the run time. However, you can view this registry definition in the Integrated Solutions Console by selecting **Service integration → WSRR definitions**.

   c. Click **Add** to add a new Condition (Figure 50), which will open the Add/Edit window.

   ![Figure 50 Adding a new Condition in the Policy Resolution primitive](image)

   a. Enter customerNumber for the Policy condition name.
   b. Click **Edit** for the XPath to launch the XPath Expression Builder.
   c. Expand **ServiceMessageObject → body → createAccount → customer → customer**, double-click **customerNumber**, and then click **Finish**.
d. You can leave the comment field empty in the Add/Empty window. Click **Finish** to complete the Condition.

d. Add another condition by clicking **Add**.

a. Enter country for the Policy condition name.

b. Click **Edit** for the XPath to launch the XPath Expression Builder.

c. Expand `ServiceMessageObject → body → createAccount → customer → customer → Address`, double-click `country`, and then click **Finish**.

d. You can leave the comment field empty in the Add/Edit window. Click **Finish** to complete the Condition.

e. Select the **Propagate mediation policy to response flow** check box.

**Note:** You would need to add another Policy Resolution mediation primitive in the response flow to resolve policies should you require information for the WebSphere Service Registry and Repository query that is only obtainable in the response message. However, in this scenario, we use only the information in the request message to resolve the policies for both the request and response flow. Selecting the **Propagate mediation policy to response flow** ensures that the policy information is preserved and propagated to the response flow.

f. The Policy Resolution properties window should now look like Figure 51.

![Figure 51 Policy Resolution properties](image)

9. Configure the LogRequest mediation primitive.

a. Right-click the **LogRequest** mediation primitive and select **Show in Properties**.
b. In the Details tab of the Properties view, select **Custom** from the Logging type drop-down menu, then clear the **Enabled** property check box, as shown in Figure 52.

![Figure 52 The LogRequest primitive properties](image)

**Note:** You can enable and disable Message Logger mediation primitives within a request and response flow. If the primitive is disable, the message will not be logged and the next mediation primitive along the flow will be invoked.

c. Click the **Promotable Properties** tab in the Properties view.
d. Select the **Promoted** check box for the Enabled property.
e. Leave all the other properties at their defaults, as shown in Figure 53.

![Figure 53 Promoting the LogRequest Enabled property](image)

**Figure 53** Promoting the LogRequest Enabled property

10. Configure the TransformAddress mediation primitive.

a. Right-click the **RequestTransform** mediation primitive and select **Show in Properties**.

b. On the Details tab of the Properties view, click **Browse** for the Mapping file to launch the Mapping File Selection window.
c. Select **NoFormat** from the list, as shown in Figure 54, and click **OK**.

![Figure 54 Selecting a mapping file](image)

**Note:** The listed mapping files have already been created for you. The TransformAddress_xx files transforms the GeneralAddress into the appropriate format for the country. The NoFormat mapping file simply maps the Address elements one by one to the target message. This file will be overridden by the promoted property override values.

These are predefined XSL files, so you will not be able to open them in the Mapping editor. To view these XSL files, open them in the XSL editor.
d. The TransformAddress properties window should now look like the one shown in Figure 55.

![Figure 55 TransformAddress properties](image)

Figure 55 TransformAddress properties

e. Click the **Promotable Properties** tab in the Properties view.

f. Select the **Promoted** check box for the Mapping file property.

g. Leave all the other properties at their defaults, as shown in Figure 56.

![Figure 56 Promoting the TransformAddress Mapping file property](image)

Figure 56 Promoting the TransformAddress Mapping file property

11. Save all your changes.

**Configuring the response flow**

To configure the response flow, perform the following steps:

1. Select the **Response** tab in the mediation flow editor.

2. Add a Message Logger mediation primitive to the response flow and rename it to LogResponse. You may have to expand the **Tracing** section to see it.

   Wire the out terminal of the CalloutResponse to the in terminal of the LogResponse primitive.

3. Add an XSL Transformation mediation primitive to the response flow and rename it to ResponseTransform. You may have to expand the **Transformation** section to see it.

   a. Wire the out terminal of the LogResponse primitive to the in terminal of the TransformResponse primitive.
b. Wire the out terminal of the TransformResponse primitive to the in terminal of the InputResponse.

The response flow should now look like the one shown in Figure 57.

![Figure 57 The response flow](image)

4. Configure the LogResponse mediation primitive.
   a. Right-click the LogResponse mediation primitive and select Show in Properties.
   b. In the Details tab of the Properties view, select Custom from the Logging type drop-down menu and clear the Enabled property check box (Figure 58).

![Figure 58 The LogResponse properties](image)

c. Click the Promotable Properties tab in the Properties view.

d. Select the Promoted check box for the Enabled property.
e. Leave all the other properties at their defaults, as shown in Figure 59.

![Figure 59 Promoting LogResponse Enabled property](image)

5. Configure the TransformResponse mediation primitive.
   a. Right-click the TransformResponse mediation primitive and select **Show in Properties**.
   b. Click the **Details** tab in the Properties view. Select **New** for the mapping file to launch the New XML Mapping window.
c. Click **Finish** in the window to use the default values, as shown in Figure 60.

![New XML Mapping](image)

*Figure 60  Creating a new mapping file*
d. Expand the target message in the mapping editor. Right-click the complete element and select Create Assign, as shown in Figure 62.

![Figure 61 Create Assign menu item]

e. In the General tab for the Assign mapping, enter “true”, as shown in Figure 62.

![Figure 62 Assigning a value to the mapping]

**Note:** Typically, you would check that the response message returned from the service provider was valid before assigning “true” to the client response message. However, in this scenario, we have hardcoded the value “true” to reduce the complexity of the flow.
f. Wire the fail terminal of the validate ; EligibilityV1_0Partner primitive to the in terminal of the validate : EligibilityV1_0Partner Fail primitive.

g. Enter the details for the fail terminal, as shown in Figure 63.

![Figure 63 Fail node details](image1)

h. Specify the error message on the fail node, as shown in Figure 64.

![Figure 64 Fail node error message](image2)

6. Save all your changes.

**Exporting the mediation module**

To export the mediation module, perform the following steps:

1. Right-click **AccountCreationModule** in the Business Integration view and select **Export**.

2. Expand **Business Integration** in the Export window and click **Integration Module**.

3. Click **Next**.

4. Keep the defaults to export the AccountCreationModule with the EAR files for server deployment option and click **Next**.

5. Enter D:\ for the Target Directory and click **Finish**.
Importing the mediation module into WebSphere Service Registry and Repository

Importing this policy mediation module to WebSphere Service Registry and Repository involves the following activities:

- Importing the mediation module into WebSphere Service Registry and Repository
- Creating Policy documents
- Attaching the Policy documents to the mediation module
- Creating Gate Conditions for the Policy Attachments

**Import the mediation module**

To import the mediation module, perform the following steps:

1. Log in to the WebSphere Service Registry and Repository console.

2. Go to the **SOA Governance** perspective. Select **Actions → Load Documents**. Click **Browse**, as shown in Figure 65.

![Figure 65  Loading SCA integration modules](image)
3. Browse to the module location and click **OK** to load the module.

WebSphere Service Registry and Repository loads the module and all the system-default policies that were generated by WebSphere Integration Developer.

4. View the system-default policy that was generated by WebSphere Integration Developer and imported automatically into WebSphere Service Registry and Repository.
   a. Click **View**.
   b. Select **Service Documents → Policy Documents**, as shown in Figure 66.
   
   ![Figure 66 Viewing the system-default policies](image)

   c. Click **AccountCreationModule-AccountCreationModule.xml** and select the **Policy** tab, as shown in Figure 67 on page 63, to review this policy.
Creating mediation policies

We can now create a number of policies in WebSphere Service Registry and Repository.

Create a policy for transforming addresses into the German format

To do this task, perform the following steps:

1. Select Service Documents → Policy Documents and then click New, as shown in Figure 68, to create a new Policy Document.
2. Select the **WSPolicy Framework 1.5** check box, as shown in Figure 69.

![Figure 69   Selecting the Policy Framework Domain](image.png)
3. Click **Next**. Enter AddressTransformation_DE as the Policy Document Name and click the **Policy** link, as shown in Figure 70.

![New Policy Document](image)

*Figure 70  Creating a new policy*
4. Click **Add Property**, as shown in Figure 71.
5. Select **Policy Identifier** from the Optional Properties drop-down menu, as shown in Figure 72.

![Image of New Policy Document with Policy Identifier selected](image)

**Figure 72** Adding the Property Identifier for a new policy

6. Click the **Add** button.
7. Enter urn:addressTransformation_DE for the Policy Identifier value, as shown in Figure 73.

![Figure 73 Assigning a unique Property Identifier value](image)

**Note:** The Property Identifier value must be unique for this WebSphere Service Registry and Repository instance.
8. Click the **Select Policy Domain** link and select **Mediation Domain [AccountCreationModule]** from the Policy Domains drop-down menu, as shown in Figure 74. Click **Apply**.

![Figure 74 Selecting the Policy Domain](image)

**Note:** The Policy Domain name is based on the Property Group for that mediation module and was specified as **AccountCreationModule** in WebSphere Integration Developer.
9. Click the **Select Policy Type** link, as shown in Figure 75. This selects the Property Group for that particular Policy Domain.

![Select Policy Type](image)

*Figure 75  Selecting the Policy Type for a new policy*

10. Click **Select** to accept that Policy Type.
11. Click the **Add Assertion** link, as shown in Figure 76. Select **Assertion CountrySpecificTransform** from the Assertion Options drop-down menu.

*Figure 76  Adding an assertion to a new policy*
12. Click **Add**. Enter `xslt/RequestTransform_DE.xsl` in the value field, as shown in Figure 77.

![Figure 77 Adding an assertion value](image)

**Note:** This value refers to the location of the pre-supplied transformation file that transforms a GenericAddress business object into the required format for a German address.

13. Click **Publish** to complete the creation of the policy.

*Create a new policy for the UK address transformation*

Repeat the steps in “Create a policy for transforming addresses into the German format” on page 63 with the following changes:

- Enter `AddressTransformation_UK` as the Policy name.
- Enter `urn:addressTransformation_UK` for the Policy Identifier value.
- Enter `xslt/RequestTransform_UK.xsl` in the Value field.

You then need to create a new Policy to enable logging in the flow. You will be adding two assertions to this Policy:

1. Enter `AccountCreationLoggingQoSPolicy` as the Policy name.
2. Enter urn:accountCreationLoggingQoSPolicy for the Policy Identifier value.

3. Click the Add Assertion link. Select Assertion LogRequest.enabled from the Assertion Options drop-down menu and click Add.

4. Enter true in the Value for Assertion field.

5. Select Add Assertion.

6. Select Assertion LogResponse.enabled from the Assertion Options drop-down menu and click Add.

7. Enter true in the Value for Assertion field. You will now see two assertions for this policy, as shown in Figure 78.

![New Policy Document](image)

**Figure 78 Adding the logging assertions**

**Note:** You have added the LogRequest.enabled and LogResponse.enabled assertions to this policy to enable logging in both the request and response flow. If you wanted to just configure the request flow or just the response flow, you could create additional policies containing just one assertion.
8. Click **Publish** to complete the policy creation. You should see the list of policies shown in Figure 79.

![Figure 79 Policy list](image)

**Attaching policy documents**

To use system-default or user-created policies, you must attach them to the mediation module. In this scenario, we only attach the user-created policies to demonstrate how the overrides are used from the Integrated Solutions Console when none of the policy conditions are met.

Perform the following steps:

1. Select **Service Documents** → **SCA Integration Modules** → **SCA Module Documents**, as shown in Figure 80.

![Figure 80 Viewing the SCA Modules](image)

2. Click **AccountCreationModule** in the list of SCA Module Documents.
3. Click **AccountCreationModule**, which is located under **Relationships → Modules**, as shown in Figure 81.

![Figure 81 Selecting the module](image)

4. Select the **Policy** tab and click **Edit Policy Attachments**, as shown in Figure 82.

![Figure 82 Editing policy attachments](image)
5. Select **Attach Policy**, as shown in Figure 83.

![Figure 83 Attaching a policy](image)

6. Click the **Search** button to show all the policies that are available for attachment.

7. Select the check box for **urn:addressTransformation_DE**, as shown in Figure 84.

![Figure 84 Attaching the germanAddress policy](image)

8. Click the **AttachSelectedPolicy** button.

9. Click the **Attach Policy** (Figure 85 on page 77) link and click the **Search** button to once again list all available policies to attach.

10. Select the check box for **urn:accountCreationLoggingQoSPolicy** and click **Attach Selected Policy**.

11. Repeat these steps, this time selecting **urn:addressTransformation_UK**, and click **Attach Selected Policy**.
12. You should now have the list of Policy Attachments shown in Figure 85. Click Finish.

![Figure 85](image1.png)  
**Figure 85** List of policy documents to be attached

**Note:** Remember, do not attach the system-default policy.

**Add gate conditions**

Gate conditions can be added to the attachments for user defined policies by performing the following steps:

1. Select **Service Documents → Policy Documents → Documents with Policy Attachments**, as shown in Figure 86.

![Figure 86](image2.png)  
**Figure 86** List of policy attachments
2. Click 
AccountCreationModule_Module_AccountCreationModule_urn:account
CreationLoggingQoSPolicy.xml, as shown in Figure 87.

![Figure 87 Selecting policy attachment](image)

3. Select the 
AccountCreationModule_Module_AccountCreationModule_urn:account
CreationLoggingQoSPolicy.xml entry under Attachments, as shown in
Figure 88.

![Figure 88 Selecting the attachment](image)
4. Click **Edit Properties**, as shown in Figure 89.

5. Click **Add Property**, as shown in Figure 90, to add a property to the Policy Attachment.
6. Enter the value `medGate_customerNumber`. Click **Add**, as shown in Figure 91.

![Policy Attachment](image)

**Figure 91** Adding the gate condition name

**Note:** All gate condition names must start with the prefix `medGate_` and are case sensitive.
7. Enter the value `customerNumber = 1000`, as shown in Figure 92.

![Figure 92 Adding the gate condition value](image)

**Note:** The Gate Condition value must be in the format `policy_condition_name operation gate_value`.

In this example, the Policy Condition name is `customerNumber`, which is defined in the Policy Resolution primitive. The operation is “=”, as we want an equality operation. The value is “1000”, as we want this policy to only apply when this condition is met for a particular message.

8. Click **Apply**.

9. Repeat for the `AccountCreationModule_Module_AccountCreationModule_urn:addressTransformation_DE.xml` attachment with the following changes:
   a. Enter `medGate_CountryDE` for the Condition name.
   b. Enter `country = "DE"` for the Condition value.

10. Repeat for the `AccountCreationModule_Module_AccountCreationModule_urn:addressTransformation_UK.xml` attachment with the following changes:
    a. Enter `medGate_CountryUK` for the Condition name.
    b. Enter `country = "UK"` for the Condition value.
This completes the setup for WebSphere Service Registry and Repository.

**Note:** If you make any subsequent changes to the WebSphere Service Registry and Repository configuration after you have deployed and tested the mediation module, you need to clear the WebSphere ESB runtime cache.

You can clear the cache in the Integrated Solutions Console by selecting **Service integration → WSRR definitions** and selecting the WSRR definition check box, followed by selecting **Clear all active caches**.

**Deploy the mediation module**

We can now deploy the mediation module and use the Integrated Solutions Console to override the Mapping file promoted property by performing the following steps:

1. **Deploy the applications.**
   a. In WebSphere Integration Developer, go to the **Servers** view and start the server.
   b. Once the server starts successfully, right-click **WebSphere Process Server v6.2 at localhost** and select **Add and Remove Projects**.
   c. Click **Add All** to deploy the AccountCreationModuleApp project and the EligibilityV1_0EAR Web service to the server, and then click **Finish**. Wait for the applications to be deployed.

2. **Modify the promoted properties.**
   a. Right-click the server in the Server view and select **Administration → Run administrative console** to launch the Integrated Solutions Console.
   b. If prompted, log into the console using admin for both the User ID and Password.
   c. Expand **Applications** and click **SCA Modules**. Click **AccountCreationModule**, as shown in Figure 93 on page 83.
d. Click **Module properties**, as shown in Figure 94.
e. Expand the `AccountCreationModule` and change the value of the property to `xslt/RequestTransform_USA.xsl`, as shown in Figure 95.

![Figure 95  Modifying the promoted property value](image)

**Note:** The `NoFormat.xsl` file was specified in the tooling to be used for the address transformation. This property can be changed at run time either by applying dynamic property overrides from a Policy Resolution primitive or through the Integrated Solutions Console. Dynamic Policy Resolution property overrides takes precedence over any values contained in the Integrated Solutions Console.

In this scenario, if no dynamic Policy Resolution overrides occur for `TransformAddress.XSLTransform`, then the `xslt/TransformAddress_USA.xsl` value will be used at run time.

f. Click **OK**.

g. Click **Save** to save the changes in the Integrated Solutions Console.

**Testing the mediation module**

To test the mediation module, perform the following steps:

1. In the Business Integration view, right-click `AccountCreationModule` and select **Test** → **Test Module**.

2. In the Integration Test Client, ensure that the Component is set to `AccountCreationModule` and the Operation is set to `createAccount`. 
Message transformation will be illustrated using the following test cases:

- Country = UK
- Country = DE
- Country = DE and customerNumber = 1000
- Country other than DE or UK

**Test case: country = UK**

In this test, specifying a country of UK should result in the policy information being obtained from WebSphere Service Registry and Repository and the dynamic property value override for RequestTransform_UK.xsl being used by the XSLT mediation primitive.

To perform this test, perform the following steps:

1. Enter the following test data:
   - creditLimit: 5000
   - active: false
   - customerNumber: 100
   - dateProfileCreated: 06-24-2009
   - firstName: John
   - lastName: Smith
   - homePhoneNumber: 111-222-3333
   - workPhoneNumber: 444-555-6666
   - email: john_smith@yahoo.com
   - dataValid: false
   - street: High Lane
   - city: Winchester
   - state:
   - country: UK
   - zip: S022 123

**Optional:** You could add a value to the zip code to demonstrate that the property override is working correctly, as this value is not mapped in the RequestTransform_UK.xsl transformation file.
2. Click **Continue** to start the test. Select **WebSphere Process Server v6.2 at localhost** as the deployment location and click **Finish** to start the test. Ensure that the User ID is admin and the password is admin if presented with the login window and click **OK**.
3. Check that the policy information was obtained correctly by expanding the Policy Resolution results, as shown in Figure 96. You may need to expand the sections under the **propertySets** to see the full results.

![Figure 96 Viewin](image-url)g the policy information obtained by the policy resolution primitive
4. You can also observe the address format printed out by the EligibilityV1_0 Web service in the server logs, as shown in Figure 97.

**Figure 97** Viewing the server logs

5. Open Windows® Explorer and navigate to the System temp directory C:\Documents and Settings\Administrator\Local Settings\Temp to ensure that the message was not logged to the file system. Although the LogRequest and LogResponse message log mediation primitives have been configured to log to the file system, their enabled property is set to false.

**Test case: country= DE**

To perform this test, perform the following steps:

1. Enter following test data, ensuring that the country value is “DE” and rerun the test:
   - creditLimit: 5000
   - active: false
   - customerNumber: 100
   - dateProfileCreated: 06-24-2009
– firstName: John
– lastName: Smith
– homePhoneNumber: 111-222-3333
– workPhoneNumber: 444-555-6666
– email: john_smith@yahoo.com
– dataValid: false
– street: High Street
– city: Munich
– state:
– country: DE
– zip: 80995

2. Examine the server logs shown in Figure 98. Notice the different address format used for this particular message.

![Figure 98 Viewing the server logs](image)

3. Examine the policy information returned by the Policy Resolution primitive. Notice how the value xslt/RequestTransform_DE.xsl is returned for this message.

**Test case: country= DE and customerNumber =1000**
Changing the customerNumber value to “1000” will result in additional policy overrides being returned for LogRequest.enabled and LogResponse.enabled, as the gate condition for the LoggingEnabled policy will be met.

To perform this test, perform the following steps:

1. Enter the following test data and rerun the test:
   – creditLimit: 5000
   – active: false
   – customerNumber: 1000
   – dateProfileCreated: 06-24-2009
   – firstName: John
   – lastName: Smith
   – homePhoneNumber: 111-222-3333
   – workPhoneNumber: 444-555-6666
   – email: john_smith@yahoo.com
   – dataValid: false
   – street: High Street
2. Examine the Policy Resolution primitive results by fully expanding the dynamicProperty context to view the information returned for the three dynamic properties.

Note: The information returned for the LogRequest.enabled and LogResponse.enabled properties contain the value “true” and will be propagated to the appropriate mediation primitives, resulting in the message loggers being enabled for this particular message.

3. Using Windows Explorer, navigate to C:\Documents and Settings\Administrator\Local Settings\Temp and note how the message has been logged to the file system in the file MessageLog.log for both the request and response.

Test with a country other than DE or UK

Using a country other than DE or UK in a message results in the property values from the Integrated Solutions Console being used.

Using a value other than DE or UK for the country will result in none of the gate conditions being met for the policies defined for the CountrySpecificTransform property attached to this module.

Because you did not use the system-default policy in WebSphere Service Registry and Repository, the value from the Integrated Solutions Console is used as the dynamic property override. The value specified in the Integrated Solutions Console is xslt/RequestTransform_USA.xsl.

To perform this test, perform the following steps:

1. Enter the following test data and rerun the test:
   - creditLimit: 5000
   - active: false
   - customerNumber: 100
   - dateProfileCreated: 06-24-2009
   - firstName: John
   - lastName: Smith
   - homePhoneNumber: 111-222-3333
   - workPhoneNumber: 444-555-6666
   - email: john_smith@yahoo.com
2. Examine the server logs that show the different address transformation.

3. Examine the Policy Resolution primitive results. Notice that no information was returned, as none of the conditions were met for this particular message.

Note: If you left the customerNumber field value as “1000”, then you would still see policy information returned for the LoggingEnabled policy.

Troubleshooting

In order to verify that the policy resolution primitive logic is working as expected, you should perform the troubleshooting techniques outlined in this section.

Verify the contents of the default module mediation policy

Before importing the mediation module EAR file into WebSphere Service Registry and Repository, verify that the WS-Policy XML file contents are accurate.

Figure 99 shows the default mediation WS-Policy file location in the enterprise archive’s EAR file.

Figure 99  Default mediation WS-Policy file location in EAR file
Example 2 shows the contents of default mediation WS-Policy file to be imported into WebSphere Service Registry and Repository.

**Example 2  Default mediation WS-Policy file**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy
xmlns:wsrr="http://www.ibm.com/xmlns/prod/serviceregistry/6/2/wspolicy"
xmlns:sibx="http://www.ibm.com/wbi/mediation/200812/AccountCreationModule"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsrr:policyClass="AccountCreationModule_CLASS"
wsu:Id="AccountCreationModule_DefaultPolicy"
wsrr:policyClassDomain="http://www.ibm.com/wbi/mediation/200812/AccountCreationModule">
  <sibx:CountrySpecificTransform
    value="xslt/NoFormat.xsl">
  </sibx:CountrySpecificTransform>
  <sibx:LogResponse.enabled
    value="false">
  </sibx:LogResponse.enabled>
  <sibx:LogRequest.enabled
    value="false">
  </sibx:LogRequest.enabled>
</wsp:Policy>
```

**Business process integration scenario**

This scenario describes how JKHL Enterprises uses a WS-BPEL business process to automatically establish an account for an existing customer.

The WS-BPEL process invokes the endpoint lookup mediation component created in “Endpoint lookup integration scenario” on page 6 to retrieve the Account Creation service endpoint stored in the organization’s service repository hosted in WebSphere Service Registry and Repository.
Scenario overview

JKHL Enterprises uses the front end Customer Care Web application to create customer and account records. The Customer Care Web application acts as a service consumer for the Establish Customer business process described in this section.

This business process shown in Figure 100 first checks whether a customer record exists for the customer before automatically creating their account. If the customer is new to JKHLE, the business process will not create the account, but will return an indicator to the service consumer application so that the required records can be manually created. (The service consumer logic is not demonstrated in this paper.)

![Figure 100 Establish customer business process]

The Account Creation service invoked by the business process validates the eligibility of the customer by utilizing the Eligibility mediation module created in “Mediation policy integration scenario” on page 43. This paper does not illustrate the Account Creation service invoking the Eligibility SCA component.

Integration issues

JKHL Enterprises uses WebSphere Process Server to host both the Establish Customer business process and the endpoint lookup mediation component in this scenario.

The Establish Customer business process invokes the endpoint lookup mediation component using SCA binding. A simple Java™ class will be used as the Identify Customer component.
WebSphere Service Registry and Repository will be accessed to retrieve a single matching endpoint for the latest version of the createAccount operation of the AccountCreation service.

Solution overview

The solution overview diagram shown in Figure 101 illustrates how the business process will be implemented using a BPEL process that invokes the endpoint lookup mediation flow to access WebSphere Service Registry and Repository.

Performing the integration

The section describes how to create a WS-BPEL business process that invokes an endpoint lookup mediation component using WebSphere Integration Developer. It describes how to test the mediation module using the Integration Test Client within WebSphere Integration Developer.
Configure access to WebSphere Service Registry and Repository

Refer to “Configure access to WebSphere Service Registry and Repository” on page 95 for information about this topic.

Create the business process

The WS-BPEL business process includes two invoke activities. The first invokes a Java class to check if the customer is known to JKHLE.

A choice activity checks the result of this step and, in the case of existing customers, invokes the endpoint lookup mediation component to create the customer account.

Setting up the project

To set up the project, perform the following steps:

1. Create a module by clicking New in the Business Integration view, as shown in Figure 102.

   ![Figure 102 Creating a module](image)

2. Select Module as the Business Integration Project Type.
3. Complete the module settings and click **Next**.
   - Module name: CustomerCareProject
   - Use default location: selected
   - Open module assembly diagram: selected

4. Select **JKHLE_Library** and click **Next**.

5. Select **Select an integrated solution**, accept the default of **WSRRIntegration**, and click **Finish**.

### Implementing the CustomerCareProject

To implement the CustomerCareProject project, perform the following steps:

2. Go to the CustomerCareProject Assembly Diagram editor.
3. Add a process component from the palette to the assembly diagram.
4. Right-click the process component and select **Show in Properties**.
5. Change the name to EstablishCustomer.
6. Right-click the **EstablishCustomer** process and click **Add Interface**.
7. Click the **CustomerCare** interface and click **OK**.
8. Open the **JKHLE_Library** and drag and drop the CustomerIdentification interface on to the assembly diagram canvas. Specify “Component with no Implementation Type” component in the Component Creation window. Click **OK**.
9. Right-click this component and select **Show in Properties**.
10. Change the name to IdentifyCustomer.
11. Wire the EstablishCustomer process to the IdentifyCustomer component and click this component to create a matching reference.
12. Right-click the **IdentifyCustomer** component and select **Generate Implementation → Java**. Click **OK** to accept the default package location.
13. Modify the code so that the identifyCustomer() method returns a Boolean value of true if the customerNumber input parameter is populated, as shown in Example 3.

Example 3  Updating the identifyCustomer implementation

```java
public Boolean identifyCustomer(DataObject customerDetails) {
   Boolean customerIdentified = new Boolean(true);
   String customerNumber = customerDetails.getString("customerNumber");
   System.out.println("<!-- Identifying Customer " + customerNumber
+ " -->");
   if (customerNumber.length() == 0)
   {
      System.out.println("<!-- Identifying Customer--> Not our customer -->");
      customerIdentified = false;
   }
   return customerIdentified;
}
```

14. Go to the assembly diagram for the CustomerCareProject. From the Business Integration view, drag the DynamicEndpointModuleExport from the DynamicEndpointProject onto the assembly diagram canvas.

15. From the Business Integration view, drag the DynamicEndpointModuleExport from the DynamicEndpointProject onto the assembly diagram canvas.

16. Select Import with SCA Binding and click OK.

17. Right-click the Import1 component and select Show in Properties.

18. Select the Description tab first, then change the name to DynamicEndpointImport.

19. Wire the EstablishCustomer process to the DynamicEndpointImport component and click OK to create a matching reference.

20. From the palette, drag an Export component onto the assembly diagram.

21. Right-click the Export1 component and select Show in Properties.

22. Select the Description tab first, then change the name to EstablishCustomerExport. This newly created export needs to be wired to EstablishCustomer module. Click OK in the Add Wire panel.
23. Save the project. The assembly diagram now looks like the one shown in Figure 103.

![CustomerCareProject assembly diagram](image)

**Creating the business process implementation**

To create the business process implementation, perform the following steps:

1. Double-click the **EstablishCustomer** component and click **Yes** to generate an implementation. Click **OK** to select the default folder location. The business process editor will open.

2. From the palette, drag two Invoke activities, one Assign activity, and a Choice activity with two Case activities and arrange them as shown in Figure 104.

![Creating the business process](image)
3. Update each activity by right-clicking and selecting **Show in Properties** and adding the Display Names as follows:
   - For the Invoke activity, enter Identify Customer.
   - For the Assign activity, enter Assign Result.
   - For the Choice activity, enter Customer Identified?.
   - For the first Case activity, enter Case Identified.
   - For the second Case activity, enter Case Not Identified.
   - For the Invoke1 activity, enter Create Account.

4. Set the details for the Identify Customer activity, as shown in Figure 105).
   a. Select the **Identify Customer** activity in the business process editor.
   b. In the Properties view, click the **Details** tab.
   c. Click the **Browse** button to select a Partner.
   d. Select **CustomerIdentificationPartner** and click **OK**.
   e. For the Inputs, click the link under **Read from Variable** and select **customerDetails**.
   f. For the Outputs, click the link under **Read from Variable**, click **New**, and click **OK** to accept the variable **customerIdentified**.

![Figure 105 Identify Customer settings](image-url)
5. Set the details for the Case Identified activity, as shown in Figure 106.
   a. Select the **Case Identified** activity in the business process editor.
   b. In the Properties view, click the **Details** tab.
   c. Click the **Create a New Condition** link.
   d. Build the expression `customerIdentified==true`.

*Figure 106  Case Identified settings*
6. Set the details for the Create Account activity, as shown in Figure 107. 
   a. Select the **Create Account** activity in the business process editor.
   b. In the Properties view, click the **Details** tab.
   c. Click the **Browse** button to select a Partner.
   d. Select **AccountCreationV1_1** and click **OK**.
   e. Select the **createAccount** operation.
   f. For the Inputs, click the link under **Read from Variable**, click **New**, and click **OK** to accept the variable **customer**.
   g. For the Outputs, click the link under **Read from Variable** and select **result**.

![Figure 107  Create Account settings](image)

7. Set the details for the Case Not Identified activity, as shown in Figure 108.
   a. Select the **Case Not Identified** activity in the business process editor.
   b. In the Properties view, click the **Details** tab.
   c. Click the **Create a New Condition** link.
   d. Build the expression **customerIdentified==false**.

![Figure 108  Case Not Identified settings](image)
8. Set the details for the Assign Result activity, as shown in Figure 109.
   a. Select the **Assign Result** activity in the business process editor.
   b. In the Properties view, click the **Details** tab.
   c. Select **customerIdentified** in the Assign From pane.
   d. Select the Boolean value of the result variable in the Assign To pane.

![Figure 109 Assign Result settings](image)

9. The business process should now look like the diagram shown in Figure 110.

![Figure 110 Establish Customer business process](image)
Testing the business process

Using the Integration Test Client, we can now test the mediation flow. Perform the following steps:

1. Go to the Servers view and ensure that WebSphere Process Server V6.2 at localhost is started. If necessary, start the server.

2. Right-click **WebSphere Process Server v6.2 at localhost** and select **Add and Remove Projects**.

3. Add the CustomerCareProjectApp. The DynamicEndpointProjectApp, the AccountCreationV1_0EAR, and the AccountCreationV1_1EAR have previously been deployed to the server.

4. Click **Finish** and wait for the module to be deployed.

5. In the Business Integration view, right-click **CustomerCareProjectApp** and select **Test → Test Module**. The Integration Test Client opens.

6. Ensure that in the Detailed Properties section of the Integration Test Client that the Component is set to CustomerCareProjectApp, the Interface is set to CustomerCare, and that the Operation is set to establishCustomerAccount, as shown in Figure 111.

![Detailed Properties](image)

*Figure 111  Test Detailed Properties*
7. Enter a value for the customerNumber input parameter, as shown in Figure 112. If a value is not entered for the customerNumber, the Identify Customer activity will return false, and the process will complete without creating the customer account.

![Figure 112 Entering test data](image)

8. Click **Continue**, as shown in Figure 113.

![Figure 113 Continue button](image)

9. In the Deployment Location window, select **WebSphere Process Server v6.2 at localhost** and click **Finish**.

10. Accept all the defaults in the User Login window. Specify a non-zero length value for the customerNumber entry field in the request parameter, such as 100. Click **OK** to start the test.
11. The CustomerCareProject module uses the DynamicEndpointProject access WebSphere Service Registry and Repository to locate the relevant endpoint, which invokes the Account Creation service at this endpoint, as shown in Figure 43 on page 40.

![Events](image)

**Figure 114** Test event trace

More detailed information about using the Integration Test Client can be found in “Testing the mediation module” on page 84.

**Troubleshooting**

The following section describes techniques to assist you with troubleshooting problems by enabling a higher level of tracing in runtime environments.

**Enable tracing of WebSphere Process Server or WebSphere ESB runtime environments**

To enter a trace string to set the trace specification to the desired state, perform the following steps:

1. Select **Troubleshooting → Logs and Trace** in the console navigation tree.
2. Select a server name and click **Change Log Level Details**.
3. Click **Configuration**.

4. If **All Components** has been enabled, you might want to turn it off, and then enable specific components.

5. Click a component or group name and select the desired log level setting. If the selected server is not running, you will not be able to see an individual component in graphic mode.

6. Alternatively, you can enter a trace string into the trace string box directly.

7. To enable tracing of all events for all WebSphere Process Server or WebSphere ESB components, use the following string:

   \[
   \text{com.ibm.ws.sibx.*}=all
   \]

8. Select **Apply** and then **OK**.

Example 4 shows the expected output for a correctly working use case featuring policy resolution primitive logic.

**Example 4  Example trace log**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Se:smo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:Se="http://www.ibm.com/websphere/sibx/smo/v6.0.1"
  xmlns:v1_1="http://jkhle.itso.ibm.com/EligibilityV1">
  <context>
    <dynamicProperty isPropagated="true">
      <propertySets>
        <group>AccountCreationModule</group>
        <properties>
          <name>CountrySpecificTransform</name>
          <value>xslt/RequestTransform_UK.xsl</value>
        </properties>
        <properties>
          <name>LogRequest.enabled</name>
          <value>true</value>
        </properties>
        <properties>
          <name>LogResponse.enabled</name>
          <value>true</value>
        </properties>
      </propertySets>
    </dynamicProperty>
  </context>
  <headers>
    <SMOHeader>
```
Enable tracing of WebSphere Service Registry and Repository runtime environments

To enable tracing of WebSphere Service Registry and Repository runtime environments, follow the sequence of steps in “Enable tracing of WebSphere Process Server or WebSphere ESB runtime environments” on page 105, but use the following trace string instead:

```
*=info:com.ibm.serviceregistry.*=all:com.ibm.sr.*=all
```

Review the trace log files for specific and detailed error messages. Consult the Messages section of any respective Information Center resources.
WebSphere Service Registry and Repository meta data caching in WebSphere Process Server

WebSphere Application Server's dynamic cache service intercepts calls to cacheable objects and either stores the output of the object or serves the content of the object from the dynamic cache.

The Java API for XML-Based RPC (JAX-RPC) Web services cache is a part of the dynamic cache service that is used to increase the performance of Web services clients by caching responses from remote Web services. The DistributedMap and DistributedObjectCache interfaces also form part of the dynamic cache. WebSphere Service Registry and Repository provides EJB, Web Services, and REST interfaces to access this functionality. Using DistributedMap and DistributedObjectCache interfaces, a mediation module application featuring a custom Java primitive to perform an endpoint or policy lookup into WebSphere Service Registry and Repository can cache and share Java objects by storing a reference to the object in the cache.

Mediation module applications can also retrieve data objects from the dynamic cache when required. This approach can potentially lead to gains in response time, throughput, and scalability considerations.

Refer to the developerWorks® article found at the following address for a specific example:

References

- WS-Policy, found at:
  http://www.w3.org/Submission/WS-Policy/
- WS-Policy 1.5 Framework, found at:
  http://www.w3.org/TR/ws-policy
- WS-Policy Attachment, found at:
  http://www.w3.org/Submission/WS-PolicyAttachment/
- Using a cache to improve performance for custom policy retrieval from WebSphere Service Registry and Repository in WebSphere ESB, found at:
The team who wrote this IBM Redpapers publication

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

Wendy Neave works as an IT Architect in IBM Global Services in Australia. She has 20 years of experience in application design and development and is an IBM Certified IT Specialist. She holds a Bachelor’s degree in Education (Environmental Science) and an Associate’s degree in Computing. Over the last 9 years, Wendy has worked primarily on Web and SOA based applications. Wendy has previously contributed to the IBM Redbooks® publications Patterns: Direct Connections for Intra- and Inter-enterprise, SG24-6933, BPEL4WS Business Processes with WebSphere Business Integration, SG24-6381, and Patterns: Extended Enterprise SOA and Web Services, SG24-7135.

Bhargav Perepa is a WebSphere IT Specialist in IBM Federal Software Group in Washington D.C., USA. He has approximately 15 years of experience in application and system software development at IBM. Bhargav holds a Master’s degree in Computer Sciences from IIT, Chicago and an MBA degree from the University of Texas. His areas of expertise include solution building, development, implementation and skill transfer of WebSphere Application Server, Business Activity Monitoring (BAM), SOA, Web Services, and Service Governance technologies. Prior to his current IBM job role, Bhargav helped develop the C++ middleware product Component Broker (CB), the Java middleware product WebSphere Application Server, Fault Tolerant CORBA (FT CORBA), and Object Request Broker (ORB), and helped with the ORB interoperability efforts at IBM Austin Research Laboratory. He worked on IBM VisualAge®, Enfin Synchronicity, Digitalk Smalltalk, and IBM Visualage C++ at IBM in Chicago. Bhargav is an IBM developerWorks Contributing Author and has written several IBM developerWorks articles in his areas of expertise.

Thanks to the following people for their contributions to the paper:

- Martin Keen
- Nicole Hargrove
- Ian Heritage
- Prasad Imandi
- Laura Olson
- Andrew White
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.

© Copyright International Business Machines Corporation 2009. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
This document REDP-4557-00 was created or updated on November 9, 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

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

  developerWorks®
  IBM®
  Redbooks®
  Redpaper™
  Redpapers™
  VisualAge®
  WebSphere®

The following terms are trademarks of other companies:

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.