IBM WebSphere Application Server Enterprise,
Version 5

Enterprise Applications

Contents

Chapter 1. Using extended messaging in applications

These topics provide information about implementing WebSphere® enterprise applications that use extended messaging.

IBM® WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java™ Message Service (JMS) programming interface. *Extended messaging* extends the base JMS support, support for EJB 2.0 message-driven beans, and the Enterprise Java Bean (EJB) component model, to use the existing container-managed persistence and transactional behavior.

Using extended messaging, you can build enterprise beans that can provide messaging services along with methods that implement business logic. The enterprise beans can use the standard JMS styles of messaging (point-to-point and publish/subscribe). However, with extended messaging, the JMS usage is simplified, because JMS support is managed by the extended messaging service. This helps to effectively separate business logic from the messaging infrastructure. The use of data mapping enables messages to drive existing or new enterprise beans as though they were invoked from any EJB client.

You can use IBM WebSphere Studio Application Developer Integration Edition to develop applications that use extended messaging. You can use the IBM WebSphere Application Server runtime tools, like the administrative console, to deploy and administer applications that use extended messaging.

For more information about implementing WebSphere enterprise applications that use extended messaging, see the following topics:
- "Extended messaging - overview"
- "Extended messaging - application usage scenarios" on page 9
- "Extended messaging - components" on page 10
- "Designing an enterprise application to use extended messaging" on page 12
- "Developing an enterprise application to use extended messaging" on page 13
- "Deploying an enterprise application to use extended messaging" on page 14
- "Configuring extended messaging service resources" on page 18
- "Troubleshooting extended messaging" on page 31

Extended messaging - overview

Extended messaging extends the base JMS support, support for EJB 2.0 message-driven beans, and the Enterprise Java Bean (EJB) component model, to use the existing container-managed persistence and transactional behavior.

In addition to providing such *container-managed messaging*, extended messaging provides new types of enterprise beans and administrative objects for messaging, and new functionality like data mapping and late response handling. (The abbreviation, CMM, for the term *container-managed messaging* is sometimes used to represent extended messaging.)

Extended messaging uses the bean-managed messaging implementation to provide the JMS interfaces, which ensures that both bean-managed and extended messaging use consistent JMS support.

An application that uses extended messaging can receive messages by using a *receiver bean*, either by the onMessage() method of a message-driven bean or by a stateless session bean that polls for a message from a named destination. With extended messaging and a message-driven bean, code within the bean can use the message to invoke business logic, as either a method within the same bean or another enterprise bean. Both the incoming message and the invocation of the receiver bean can be included within the scope of a transaction. For outbound messages, an application calls a *sender bean* that turns a method call into a JMS message that is then sent asynchronously. These message beans are implemented as enterprise beans by IBM WebSphere Application Server. Application developers can create these message beans by using IBM WebSphere Studio Application Developer Integration Edition, although they can be created without the help of IBM WebSphere Studio.

With extended messaging, the JMS usage is simplified, because JMS support is managed by the extended messaging service. This helps to effectively separate business logic from the messaging infrastructure. Also, the use of data mapping enables messages to drive existing or new enterprise beans as though they are invoked from any EJB client. IBM WebSphere Studio enables the types of message beans that support extended messaging to be developed easily and hides the messaging infrastructure from developers.

For more conceptual information about extended messaging, see the following topics:

- “Extended messaging - receiving messages”
- “Extended messaging - sending messages” on page 4
- “Extended messaging - data mapping” on page 5
- “Extended messaging - handling late responses” on page 5
- “Extended messaging - transactional support” on page 7
- “Extended messaging - exception handling” on page 8
- “Extended messaging - application usage scenarios” on page 9
- “Extended messaging - components” on page 10

Extended messaging - receiving messages

To receive messages, applications that use extended messaging use a receiver bean, which can be a message-driven bean or a session bean:

- A receiver bean (deployed as a message-driven bean) is invoked when a message arrives at a JMS destination for which a listener is active.
- An application-callable receiver bean (deployed as a session bean) polls a JMS destination until a message arrives, gets the parsed message as an object, and can use getter methods to retrieve the message data.

Receiving messages with extended messaging

This figure shows an application calling a receiver bean (as a session bean) to receive messages from the JMS destination defined on an input port. The application also calls the ReplySender() method of the receiver bean to send a reply to the original message received. For more information about what is shown in this figure, see the text that accompanies this figure.

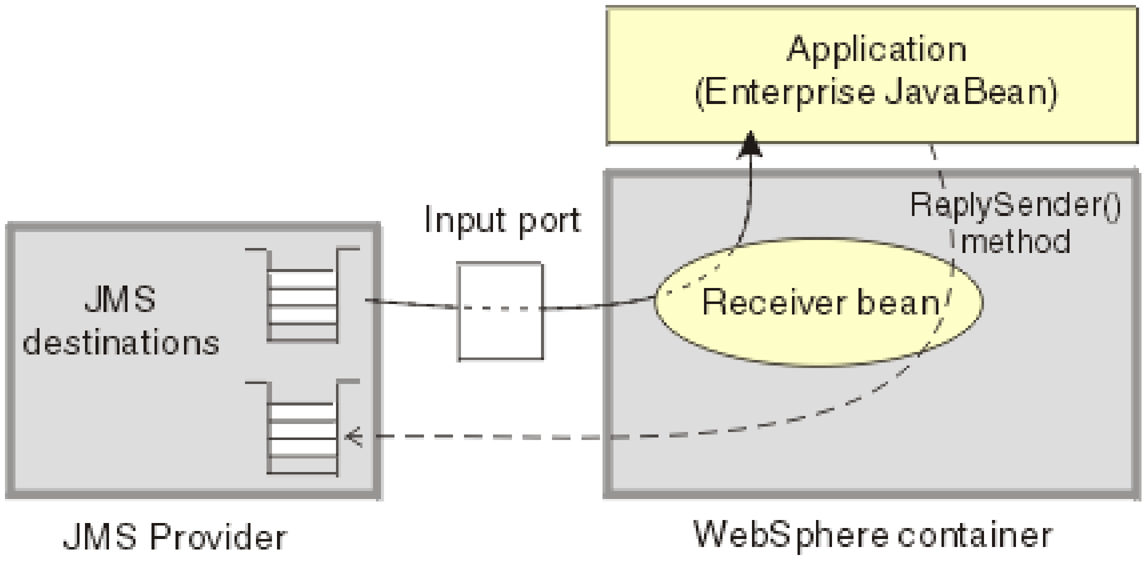


Figure 1. Application calling a receiver bean

When a receiver bean gets a message, it can invoke another method passing either the JMS Message, or a set of parameters extracted from the message content. The invoked method can be contained in the receiver bean or in another enterprise bean (which is the preferred application structure). If data mapping is used, the method invoked by a receiver bean is unaware of the original JMS message.

In addition to receiving messages, extended messaging enables applications to reply to received messages in either of the following ways:

- **Sending a synchronous reply.**

  In this mode, which can be used by only receiver beans deployed as message-driven beans, the reply from the method invoked by the receiver bean is mapped to a JMS message and sent as a reply to the original message, using the replyTo field in the JMS header as the target destination.
- **Sending an asynchronous reply.**

  In this mode, which cannot be used receiver beans deployed as message-driven beans, the application calls the ReplySender() method to send the reply message. If the reply is passed as a set of parameters to the ReplySender() method, the reply is mapped to a JMS message before being sent.

If a receiver bean gets a JMS message, then depending upon the programming model (associated with the receiver bean by WebSphere Studio), one of the following interactions occurs:

- **Receive a request and send no reply.**

  The receiver bean provides an anonymous invocation in the form of a method call. The data passed to the method is either the JMS message (if no data mapping is invoked) or a set of parameters mapped from the JMS message. The receiver bean cannot return a reply to this invocation. This mode of interaction can be used with point-to-point or publish/subscribe messaging.
- **Receive a request and send a synchronous reply.**

If the receiver bean gets a message, it invokes another method either in the same bean or another enterprise bean. When the method returns, the data returned from that call is mapped to a JMS message and sent to the reply destination specified in the original request message. The type of reply destination (queue or topic) must be the same as the type used by the original request received.

- **Receive a request and send an asynchronous reply.**

  For a receiver bean deployed as a message-driven bean, the reply is returned (using the bean's ReplySender method) to the replyTo destination specified in the original request message.

  For a receiver bean deployed as a session bean, the reply is returned to the destination defined in the input port for the receiver bean.

  In addition to the asynchronous model of this interaction, this mode of interaction enables a method to send multiple replies to a single invocation.

Extended messaging - sending messages

To send messages, applications that use extended messaging call a method on a *sender bean*. A sender bean turns its method invocation into a JMS message, then passes that message to JMS. If needed, the sender bean can retrieve a response message, then translate that message into a result value and return it to the caller. If data mapping is used, the method that invokes a sender bean is unaware of the original JMS message. The sender bean methods can use data mapping to build JMS messages from data passed on the method call.

Sending messages with extended messaging

This figure shows an application calling a sender bean to send messages to the JMS destination defined on an output port. The application also calls the receiveResponse() method of the sender bean to receive a reply to the original message sent. For more information about what is shown in this figure, see the text that accompanies this figure.

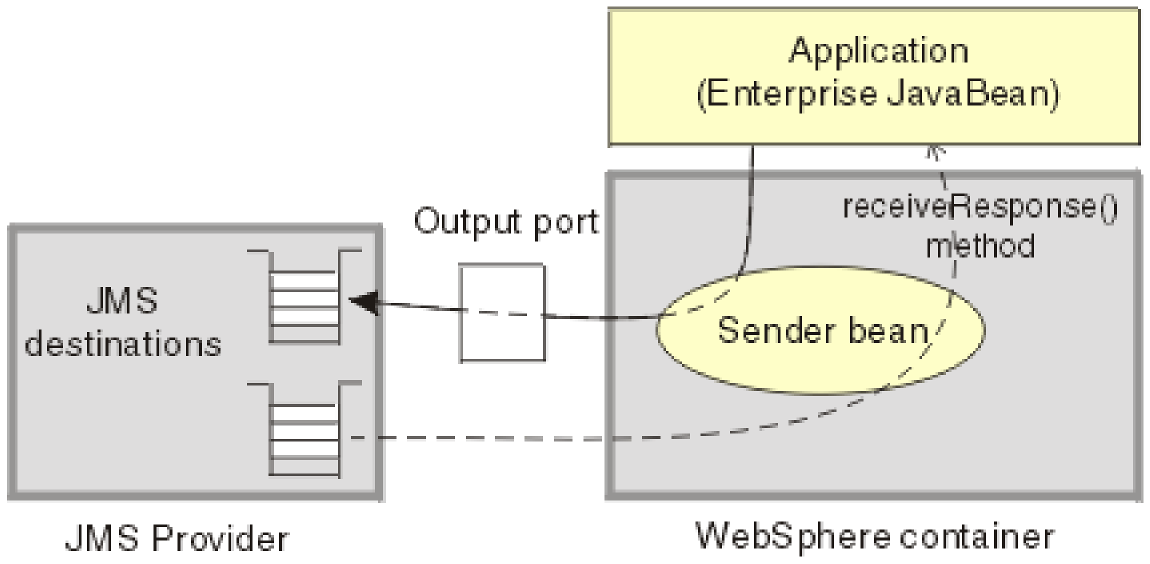


Figure 2. Application calling a sender bean

A sender bean is an enterprise bean (stateless session bean) that can be built by WebSphere Studio Application Developer. A sender bean should not contain any application logic, to help preserve the separation between the messaging and business logic.

Each method defined on a class that implements a sender bean has one of the following modes of interaction (which is defined when the sender bean is built). The interaction extends the sender interface to address the issue of synchronizing anonymous invocations.

- **Send a request and receive no response**

  To send a JMS message, an application invokes the sender bean's method. The caller of the sender bean's method cannot receive a response to the message sent. This mode of interaction can be used with point-to-point or publish/subscribe messaging.
- **Send a request and receive a synchronous response**

  To send a JMS message and wait for a synchronous response, an application invokes the sender bean's method. The sender bean uses the message sender (an interface to JMS provided by extended messaging) to send the message and, when the response is received, to return the response message to the caller of the sender bean. This mode of interaction can be used with point-to-point messaging only.
- **Send a request and receive a deferred response**

  To send a JMS message and wait for a deferred response, an application invokes the sender bean's method. The sender bean uses the message sender to send the message, then returns to the caller without waiting for the response. The response is returned by a generated receiveResponse() method. This mode of interaction enables an application to receive more than one response message, as the application is responsible for retrieving the responses. This mode of interaction can be used with point-to-point messaging only.

Extended messaging - data mapping

A message bean can use data mapping to map between a JMS message and data as arguments:

- With data mapping, the target method of a receiver bean for an anonymous invocation receives the contents of an asynchronous message as arguments. The extended messaging service parses the JMS message and maps from the JMS message to the method arguments. Similarly, to send a message, an application invokes a method on a sender bean with appropriate arguments. The extended messaging service packs appropriate arguments into a JMS message then sends the asynchronous request.
- Without data mapping, the target method of a receiver bean for an anonymous invocation receives a JMS message; no data mapping is performed by extended messaging. Similarly, to send a message, an application invokes a method of a sender bean with a JMS message.

If a developer selects data mapping when creating a sender or receiver bean, extended messaging uses the parameter properties specified on the sender or receiver bean method signatures to perform the data mapping.

Extended messaging - handling late responses

If an application uses a sender bean to send a message, it can optionally retrieve a response to the message. The sender bean can either wait for the response or defer retrieval of the response. Sometimes a response is delayed within the messaging infrastructure, and therefore the application cannot receive the response. Extended messaging can retrieve such a response message (referred to as a *late-response* message) when it does arrive and pass it to a message-driven bean provided by the application to handle late responses. The message-driven bean used to handle the late response is a standard EJB 2.0 message-driven bean or a receiver bean deployed as a message-driven bean. The deployed message-driven bean can then perform its processing on the message.

Late responses should not be considered normal application behaviour.

For extended messaging to handle late responses for an application, the sender bean must be deployed with the **Handle late responses** option enabled.

Definition of a late response

A late response occurs when the application is no longer able to retrieve responses to messages that it has sent, as follows:

- **Send with deferred response**

  The application (enterprise bean) repeatedly tries to retrieve a response until it ends. When the application no longer wants to retry to get a response, it can register a request for extended messaging to handle the late response, by calling a registerLateResponse() method on the sender bean.
- **Send with synchronous response handling**

  When the sender bean sends a message, it waits for the response. The result of this is that either the sender bean retrieves the response message or a timeout error occurs. If the system raises a timeout error, the application can no longer retrieve a response to the message. At this time the extended messaging service registers the the message for a late response.

Handling responses

Extended messaging handles responses in the following stages:

1. Registering an interest in having a late response retrieved when it is available.

   To request the system to handle late responses for a sender bean, you deploy the sender bean with the **Handle late responses** extension to the deployment Descriptor.

   If selected, the **Handle late responses** option defines that extended messaging should pass the response, when it becomes available, to the message-driven bean provided by the application to handle late responses. When the sender bean is deployed a specialized listener port is associated with the bean. This listener port is known as a *handle late response listener port*.

   If the option is not selected, then the system does not handle late responses, and it is the application's responsibility to handle any late responses.
2. Starting a JMS listener to retrieve the message when it is available, which then drives the message bean to handle the JMS message.

   The listener port must be defined with the following properties:

   - The same JMS destination as specified as the JMS response destination on the output port used by the sender bean.
   - A listener port extension with Handle Late Responses enabled.

   You cannot use a temporary destination for late responses.
3. If a request is made to handle a late response, the extended messaging service immediately registers a LateResponse message request with the extended

message consumer for the given listener port. The message request is registered independently of any transaction context that the sender bean has. A request record (containing the MessageID of the late response) is added to the AsyncMessageLog log. When the message is eventually received, it is passed to the message-driven bean deployed against the specified late response ListenerPort.

Extended messaging - transactional support

The global transaction context is not flowed on asynchronous (messaging) requests, so the receipt of an asynchronous message cannot be part of some existing remotely-established transaction. Reliability in an asynchronous environment is built on the message provider's ability to guarantee a once-and-once-only message delivery.

Transactional support with extended messaging builds on, and extends, the transactional support provided with bean-managed messaging, as follows:

- Transactional support for receiving messages (receiver beans)
- Transactional support for sending messages (sender beans)

Transactional support for receiving messages (receiver beans)

The extended messaging transactional behaviour for receiver beans depends upon whether the bean is a receiver bean or an application-callable receiver bean.

- For a receiver bean (deployed as a message-driven bean), incoming message receipts are defined by the Transaction attribute of the onMessage() bean method. Message-driven beans can use bean-managed transactions (BMT) or container-managed transactions (CMT). For message-driven beans using CMT there are only two supported transaction options: `Not supported` and `Required`. If a message is to be received within a transaction, the message-driven bean must be using CMT with the Transaction attribute set to `Required`.
- For a receiver bean as a session bean, the bean only supports container-managed transactions, and the behaviour is defined by the Transaction attribute of the receiver bean method.

Dealing with retries: In the asynchronous environment of transaction processing, rolling back a message receipt means that the message is not removed from the source destination. Although this behaviour is desirable and correct, it causes the message retained on the source queue to be reprocessed until the transaction commits. For receiver beans, you can control this behaviour as follows:

- **Receiver bean**

  To limit the number of times that a transaction is retried, you can either rely on the facilities of the JMS provider or use the retry limit facility of the Message Listener:
  - WebSphere MQ JMS support provides the ability to move the message to a backout queue and uses two queue attributes, the backout threshold and the backout-requeue queue, to perform this.
  - The Message Listener retry count can be used to stop the listener processing the queue if the threshold is reached. The listener behaviour can be disabled by setting the retry count value higher than the JMS provider threshold value.
- **Application-callable receiver bean**

  To limit the number of times that a transaction is retried depends on the facilities of the JMS provider to move the message to a backout queue.

Transactional support for sending messages (sender beans)

The transactional behaviour for sending messages is defined by the Transaction attribute on the send method within the sender bean.

If the send() method is part of a transaction, then the sending of an outgoing message occurs within any currently active transaction. This means that the message is not transmitted until the transaction is committed. If no transaction is active when the request to send the message occurs, then the message is transmitted immediately.

The transactional behaviour where the mode of interaction for a sender bean method specifies a response (that is, either *Send message and receive synchronous response* or *Send message and receive deferred response*) depends upon the type of response, as follows:

Transactional behaviour for a synchronous response
The sending of the request message and the receipt of the response message cannot be performed inside a transaction, because they are both performed within the same method. Therefore, the send is always non-transactional, regardless of the transactional setting for the method. The receive is either transactional or not, depending upon the Transaction setting of the method.

Transactional behaviour for a deferred response
The response message is received by a receiveResponse() method, which can have a different transactional behaviour to the sender method inside the sender bean. The transaction containing the send command must commit before the response can be received. The transactional behaviour is specified on the send and receive methods of the Sender bean.

Extended messaging - exception handling

Extended messaging provides the following exception handling for receiver beans and sender beans:
- Error handling for receiver beans
- Error handling for sender beans

Error handling for receiver beans

The following error conditions can lead to extended messaging exceptions
- Formatting error parsing the message, when performing data mapping
- Exception thrown by the application method
- CMMException when sending the reply

Errors are always logged. If the application does not catch the exception, the default behavior is to roll back any active transaction. If the received message is rolled back, then it can be processed again. This can occur a number of times until the message causing the error is removed from the queue by the JMS provider. (For more information, see *Dealing with retries* in "Extended messaging - transactional support" on page 7.

With data mapping, if a receiver bean is deployed as a message-driven bean and a replyTo destination is configured, then error messages are sent as replies to that destination.

Application enterprise beans that call receiver beans deployed as session beans need to handle the CMMException exception. CMMException is an application exception which is declared in the throws clause of the methods in the generated receiver bean.

Error handling for sender beans

The following error conditions can lead to extended messaging exceptions

- Constructing the JMS message when data mapping from the parameters to the message
- Creating a message sender and sending the message
- Getting the response and parsing the message content

Errors are always logged. If the application does not catch the exception, the default behavior is to roll back any active transaction.

Application enterprise beans that call sender beans need to handle the CMMException exception, which is declared in the throws clause of the methods in the generated sender bean.

Extended messaging - application usage scenarios

Applications can use extended messaging to receive and send messages in a variety of ways:

- To receive messages, applications that use extended messaging use a *receiver bean*(deployed as a message-driven bean) or an *application-callable receiver bean* (deployed as a session bean):
  - A receiver bean is invoked when a message arrives at a destination for which a listener is active.
  - An application-callable receiver bean polls a destination (defined by an *input port*) until a message arrives or a timeout occurs.

  In addition to receiving messages, extended messaging enables applications to send replies in response to the received messages.

  For more conceptual information about receiving messages, see “Extended messaging - receiving messages” on page 2.
- To send messages, applications that use extended messaging call *sender bean* methods. The sender bean sends messages to the target destination defined by an *output port*. The sender bean methods can be passed either a JMS message or a number of parameters that are mapped by extended messaging into a JMS message. Whether or not data mapping is used is specified when the application is developed.

  In addition to sending messages, applications can choose to receive a response to the message, and can handle any responses either synchronously or asynchronously. If a response is not received in time, then the system can handle the late response by directing the message to a message-driven bean.

  For more conceptual information about sending messages, see “Extended messaging - sending messages” on page 4.
- Applications can combine receiving and sending messages in a variety of different ways. For example, a receiver bean deployed as a message-driven bean can forward the message by calling a sender bean. The receiver bean can give message data to sender bean in either of the following ways:

- The receiver bean can pass the JMS message to the sender bean, which forwards that message.
- The receiver bean can extract data from the initial message and send that data to the sender bean. The sender bean can then map the data values to a new JMS message, which it forwards.

The application can receive a response to the message that it sent, and then can send the message received or a new message built from data in the message received, as a response to the original message.

Also, data mapping can be used to hide the JMS message structure from the application logic. For more information about data mapping, see “Extended messaging - data mapping” on page 5

Extended messaging - components

Extended messaging builds on the base support for JMS messaging and message-driven beans provided by IBM WebSphere Application Server. The new messaging components for extended messaging are referred to as the *Message Bean package*.

Components for receiving messages

The following components, shown in the figure Figure 3 on page 11, are used to receive messages:

Receiver bean
An application that uses extended messaging can receive messages by using a *receiver bean* (using the onMessage() method of a message-driven bean) or an *application-callable receiver bean* (a stateless session bean that polls for a message from a named destination). Both receiver beans and application-callable receiver beans can receive and process asynchronous messages, and optionally return selected data as a response message.

Input port
An input port specifies the properties needed by receiver beans as session beans, by defining the following information:

- Information about the source destination for the message to be received
- Information about how to select and handle the message received
- Optional information about a reply destination, which is used if a reply is expected and replyTo information is not present in the JMSHeader of the message received.

A receiver bean as a deployed message-driven bean uses the associated listener port, so does not need an input port. For more information about message-driven beans and listener ports, see ″Message-driven beans - components″ (not in this document).

For more conceptual information about receiving messages, see “Extended messaging - receiving messages” on page 2.

Components for receiving messages

This figure shows an application calling a receiver bean (as a session bean) to receive messages from the JMS destination defined on an input port. The application also calls the ReplySender() method of the receiver bean to send a reply to the original message received. For more information about what is shown in this figure, see the text that accompanies this figure.

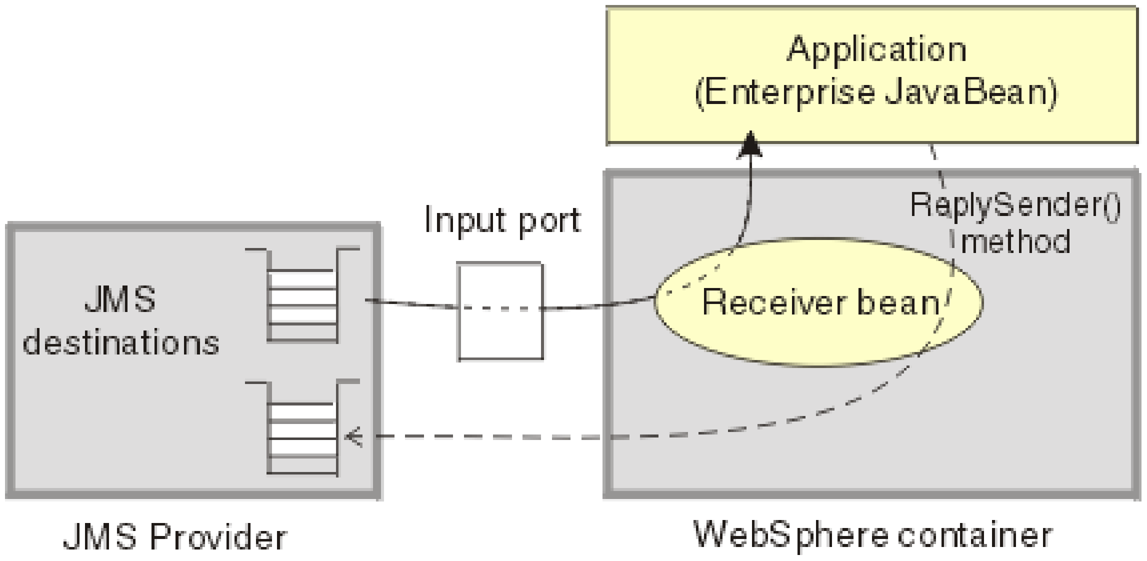


Figure 3. Components for receiving messages

Components for sending messages

The following components, shown in the figure Figure 4 on page 12, are used to send messages:

Sender bean
: Bean instances of a sender bean (also known as a *message sender bean*) can send asynchronous messages. The sender bean methods can be passed either a JMS message or a number of parameters that are mapped by extended messaging into a JMS message, which is then passed to JMS.

Output port
: An output port specifies the properties needed by sender beans, to define the destination for the message being sent, and other optional properties if a response is expected. It is associated with the Sender Bean at deployment time and contains the following information:
  - Information about the target destination for the message to be sent
  - Information about how to select and handle the message to be sent
  - Information about the destination used for the response.

For more conceptual information about sending messages, see "Extended messaging - sending messages" on page 4.

Components for sending messages

This figure shows an application calling a sender bean to send messages to the JMS destination defined on an output port. The application also calls the receiveResponse() method of the sender bean to receive a reply to the original message sent. For more information about what is shown in this figure, see the text that accompanies this figure.

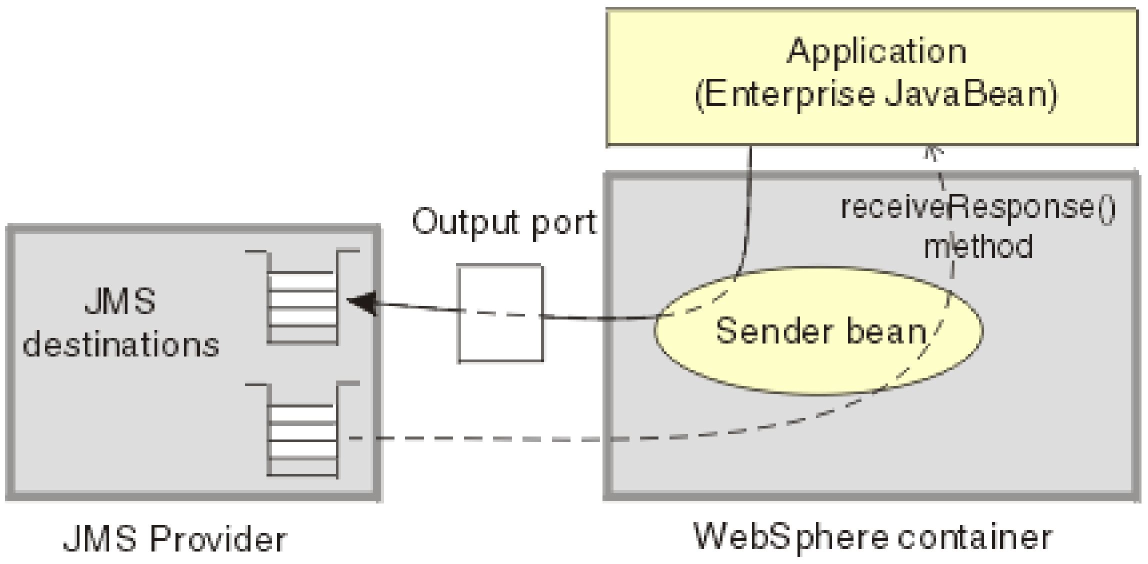


Figure 4. Components for sending messages

Designing an enterprise application to use extended messaging

This topic describes things to consider when designing an enterprise application to use extended messaging.

The design of JMS-usage for applications that use extended messaging is the same as the design for JMS and message-driven beans, except that the JMS-usage is simplified because JMS support is managed by the extended messaging service. For design considerations related to JMS and message-driven beans, see the following topics:

- Designing an enterprise application to use JMS
- Designing an enterprise application to use message-driven beans

The extra design consideration for applications that use extended messaging are as follows. For more detail, see the related topics.

Steps for this task

1. For a receiver bean, decide whether to use a message-driven bean or stateless session bean.

   Message-driven bean

   You can use a deployed message-driven bean as a receiver bean, to automatically handle messages received at the associated listener port. As with any message-driven bean, when a message is received on the JMS destination monitored by the listener port, the message is passed to the onMessage() method of the message-driven bean.

   You need to develop and deploy the message-driven bean, and configure its associated listener port, separately from the extended messaging tasks.

Stateless session bean

You can use a stateless session bean as a receiver bean, to poll for messages on a named destination associated with an input port.

You need to develop and deploy the session bean separately from the extended messaging tasks, but configure the associated input port as part of the extended messaging tasks.

2. Decide whether or not you want to use data mapping.

   If you call the methods of sender and receiver beans with data arguments, you need to use data mapping to construct the JMS messages needed. For data mapping, you need to decide what data arguments need to be specified as properties on the sender or receiver bean method signatures.

   For a receiver bean deployed as a message-driven bean, you can define the mapping behavior if a data exception is caught by extended messaging. That is, you define whether a message should be flowed back if a ReplyTo destination is defined in the JMS message header.

3. Decide whether or not you want to handle late responses.

   A sender bean can optionally retrieve a response to messages sent. If a response is delayed within the messaging infrastructure, the bean cannot receive the response. Extended messaging can retrieve such a response message (referred to as a late-response message) when it does arrive and pass it to a message-driven bean provided by the application to handle late responses. To handle late responses, you need to develop and deploy a standard EJB 2.0 message-driven bean that contains a registerLateResponse() method, and associate it with a listener port to be used to receive late responses.

Developing an enterprise application to use extended messaging

This topic describes how to develop an enterprise application to use extended messaging.

This task description assumes that developers are using the WebSphere Studio Application Developer to develop the application code (receiver and sender beans).

To develop an enterprise application to use extended messaging, complete the following steps:

Steps for this task

1. Creating the Enterprise Application project.

   Because the sender and receiver beans used for extended messaging are EJB 2.0 enterprise beans, you must first have created a J2EE 1.3 Enterprise Application project for which extended messaging beans will be created.

   a. Ensure that you have selected 1.3 as the highest J2EE version that is to be used in WebSphere Studio.

      For example: **Window-> Preferences... J2EE preferences-> Select the highest J2EE version that is to be used-> 1.3**

   b. Create a J2EE 1.3 Enterprise Application project, as described in the WebSphere Studio article entitled, ″Creating an Enterprise Application project″ (which can be accessed only when this topic is installed in a WebSphere Studio doc plug-in)

2. Creating the application code.

To create the application code, use WebSphere Studio to generate the sender and receiver beans needed by the application, by completing one or more of the following subtasks as described in the WebSphere Studio Extended Messaging documentation:

- ″Creating a sender bean″ (which can be accessed only when this topic is installed in a WebSphere Studio documentation plug-in)
- ″Creating a receiver bean″ (which can be accessed only when this topic is installed in a WebSphere Studio documentation plug-in)
- ″Creating an application-callable receiver bean″ (which can be accessed only when this topic is installed in a WebSphere Studio documentation plug-in)
- ″Creating a sender bean and receiver bean″ (which can be accessed only when this topic is installed in a WebSphere Studio documentation plug-in)
- ″Creating a sender bean and application-callable receiver bean″ (which can be accessed only when this topic is installed in a WebSphere Studio documentation plug-in)

The result of this stage is an enterprise bean, containing code automatically generated for extended messaging, that can be assembled into an .EAR file for deployment.

3. Assembling and packaging the application for deployment.

   You can use WebSphere Studio to assemble and package the application for deployment.

   The following aspects are specific to extended messaging:

   a. **(Optional)** Configure a message selector for a receiver bean.
   b. Associate the JNDI names for sender and receiver beans with output and input ports.
   c. **(Optional)** Specify the timeout for a sender bean response.
   d. **(Optional)** Configure that a sender bean is to handle late responses and identify the listener port to be used for late responses.

Results

The result of this task is an .EAR file, containing an application enterprise bean with code for extended messaging, that can be deployed in IBM WebSphere Application Server.

What to do next

For information about deploying an application to use extended messaging, see “Deploying an enterprise application to use extended messaging”.

Deploying an enterprise application to use extended messaging

This topic describes how to deploy an enterprise application to use extended messaging.

This task description assumes that you have an .EAR file, which contains an application enterprise bean with code for extended messaging, that can be deployed in IBM WebSphere Application Server.

The Application Install task is also a standard IBM WebSphere Application Server task. As part of the install procedure you need to associate the Input and Output ports defined in System Management with the installed .EAR.

To deploy an enterprise application to use extended messaging, complete the following steps:

Steps for this task

1. Use the administrative console to define and configure the extended messaging resources to be used by the application, as described in "Configuring extended messaging service resources" on page 18.

   You should define the input ports for receiver beans, the output ports for sender beans, and listener port extensions for any sender beans that are to handle late responses.
2. Ensure that the deployment descriptor attributes for the sender and receiver beans match those of the extended messaging resources that you configured using the administrative console.

   The deployment descriptor values can be set when you generate the deployment code for the application. You can change the deployment descriptor values by using the application assembly tool, as described in the following topics:
   - "Configuring deployment attributes for a receiver bean"
   - "Configuring deployment attributes for a sender bean" on page 17
3. If a sender bean is to handle late responses, deploy the message-driven bean to be used for late responses.

   For more information about deploying message-driven beans, see ″Deploying an enterprise application to use message-driven beans″ (not in this document-see the InfoCenter).
4. Install the application into IBM WebSphere Application Server.

   This stage is a standard IBM WebSphere Application Server task, as described in ″Installing a new application″ (not in this document-see the InfoCenter).

   When you install the application, you are prompted to specify the name of the listener port that the application is to use for late responses. Select the listener port, then click **OK**.

Configuring deployment attributes for a receiver bean

Use this task to configure the deployment attributes for a receiver bean for use with the extended messaging service.

You can specify these deployment attributes on each EJB method, as part of the deployment of the receiver bean. Changes to the deployment attributes override the values defined when the receiver bean was developed and deployment code was generated for the application.

Note: After deployment code has been generated for an application, the deployable archive is renamed with the prefix Deployed_ . Any subsequent changes to the archive from within the Application Assembly Tool are applied to the version of the archive that existed prior to code generation. To see changes reflected in your application, you must regenerate deployment code and re-install the deployable archive.

To configure the deployment attributes for a receiver bean, you can use the Application Assembly Tool to complete the following steps:

Steps for this task

1. Launch the Application Assembly Tool.

2. Create or edit the application EAR file.

   For example, to change attributes of an existing application, click **File-> Open** then select the EAR file.
3. In the navigation pane, select the receiver bean instance; for example, expand ***ejb_module_instance*-> Extended messaging** then select the bean instance.

   A property dialog notebook for the receiver bean is displayed in the property pane.
4. In the property pane, specify appropriate deployment attributes:

   Input port
      For an application-callable receiver bean, this is the name of the input port to be used to receive messages.

   Message selector
      For an application-callable receiver bean, this is a string used to select messages to be received.

   For more information about the deployment attributes for receiver beans, see "Extended messaging assembly properties for EJB modules".
5. To apply the changes and close the Application Assembly Tool, click **OK**. Otherwise, to apply the values but keep the property dialog open for additional edits, click **Apply**.
6. **(Optional)** To see changes reflected in your application, regenerate deployment code and reinstall the deployable archive.

Extended messaging assembly properties for EJB modules

Use this page to configure extended messaging settings for methods of an enterprise bean.

Name: Specifies a name for the mapping between extended messaging settings and one or more methods.

Datatype
   String

Description: Contains text that describes the mapping

Datatype
   String

Methods: The methods to which these settings apply.

To add a new method, click **Add**. Expand the tree to select the method or methods from the EJB module

Reply timeout: The time in milliseconds after which replies, delayed within the messaging infrastructure, are considered as late.

Type the global reply timeout to be used if a reply timeout is not specified on a sender method call. If you leave this field blank, no global timeout is defined.

This is the time in milliseconds after which replies, delayed within the messaging infrastructure, are considered as late and therefore the application cannot receive the response. The extended messaging service can retrieve such a response message (referred to as a *late-reply* message) when it arrives and pass it to a message bean provided by the application.

Data type
Integer

Units Milliseconds

Default
0

Range An integer number of milliseconds, greater than or equal to 0 (0 indicates that reply messages never timeout).

Message selector: The JMS message selector to be used to determine which messages the method handles.

For example:

```
JMSType='car' AND color='blue' AND weight>2500
```

The selector string can refer to fields in the JMS message header and fields in the message properties. Message selectors cannot reference message body values.

Data type
String, whose syntax is based on a subset of the SQL92 conditional expression syntax.

Handle late responses: Select this check box if you want to generate a method to handle late responses.

Late response handler listener port name: The name of the input port used to handle late responses.

This string must match the name of an input port defined in the Administrative Console.

Data type
String

Configuring deployment attributes for a sender bean

Use this task to configure the deployment attributes for a sender bean.

You can specify deployment attributes on each EJB method.

Changes to the deployment attributes override the values defined when the sender bean was developed and deployment code was generated for the application.

Note: After deployment code has been generated for an application, the deployable archive is renamed with the prefix Deployed_ . Any subsequent changes to the archive from within the Application Assembly Tool are applied to the version of the archive that existed prior to code generation. To see changes reflected in your application, you must regenerate deployment code and re-install the deployable archive.

To change the deployment attributes for a sender bean, you can use the Application Assembly Tool to complete the following steps:

Steps for this task

1. Launch the Application Assembly Tool.
2. Create or edit the application EAR file.

For example, to change attributes of an existing application, click **File-> Open** then select the EAR file.

3. In the navigation pane, select the sender bean instance; for example, expand ***ejb_module_instance*-> Extended messaging** then select the bean instance.

   A property dialog notebook for the sender bean is displayed in the property pane.

4. In the property pane, specify appropriate deployment attributes:

   Output port
   This is the name of the output port to be used to send messages.

   Handle late responses
   Select this checkbox if the sender bean is to handle late responses. If you select this checkbox, also specify the following properties: **ReplyTimeout** and **Late response handler listener port name**.

   ReplyTimeout
   For a sender bean that has been developed to handle late responses, this is the time after which responses are considered late. This property is used if a response timeout is not specified on a sender method call.

   Late response handler listener port name
   For a sender bean that has been developed to handle late responses, this is the name of the listener port to be used for late responses.

   For more information about the deployment attributes for sender beans, see "Extended messaging assembly properties for EJB modules" on page 16.

5. To apply the changes and close the Application Assembly Tool, click **OK**. Otherwise, to apply the values but keep the property dialog open for additional edits, click **Apply**.

6. **(Optional)** To see changes reflected in your application, regenerate deployment code and reinstall the deployable archive.

Configuring extended messaging service resources

Use these tasks with the WebSphere Administrative console to configure resources needed by the extended messaging service and applications that use extended messaging.

You can use IBM WebSphere Application Server system management to configure resources needed by the extended messaging service and applications that use extended messaging.

For more information about the tasks involved, see the following topics:

- "Adding a new input port" on page 19
- "Adding a new output port" on page 19
- "Configuring an input port" on page 20
- "Configuring an output port" on page 20
- Configuring a listener port to handle late responses (not in this document-see the InfoCenter)

```
Find a link for the previous document
```

Adding a new input port

Use this task to add a new input port to IBM WebSphere Application Server.

An input port is for use by an application that uses extended messaging.

During this task you configure the initial properties of the input port. You can later change the properties of the port, as described in "Configuring an input port" on page 20.

To add a new input port, complete the following steps:

Steps for this task

1. Start the WebSphere Administrative console.
2. In the navigation pane, select **Resources-> Extended messaging provider**
   This displays resources for extended messaging in the content pane.
3. In the Additional Properties table of the content pane, select **Input ports**
   This displays a list of the input ports in the content pane.
4. Click **New**.
5. Specify appropriate (properties of the input port).
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

Adding a new output port

Use this task to add a new output port to IBM WebSphere Application Server, and configure its properties, for use by an application that uses extended messaging.

During this task you configure the initial properties of the output port. You can later change the properties of the port, as described in "Configuring an output port" on page 20.

To add a new output port, complete the following steps:

Steps for this task

1. Start the WebSphere Administrative console.
2. In the navigation pane, select **Resources-> Extended messaging provider**
   This displays resources for extended messaging in the content pane.
3. In the Additional Properties table of the content pane, select **Output ports**
   This displays a list of the output ports in the content pane.
4. Click **New**.
5. Specify appropriate (properties of the output port).
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

Configuring an input port

Use this task to change the properties of an input port for use by an application that uses extended messaging.

To change the properties of an input port, complete the following steps:

Steps for this task

1. Start the WebSphere Administrative console.
2. In the navigation pane, select **Resources-> Extended messaging provider**

   This displays resources for extended messaging in the content pane.
3. In the Additional Properties table of the content pane, select **Input ports**

   This displays a list of the input ports in the content pane.
4. Select the input port that you want to change.
5. Specify appropriate (properties of the input port).
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

Configuring an output port

Use this task to change the properties of an output port for use by an application that uses extended messaging.

To change the properties of an output port, complete the following steps:

Steps for this task

1. Start the WebSphere Administrative console.
2. In the navigation pane, select **Resources-> Extended messaging provider**

   This displays resources for extended messaging in the content pane.
3. In the Additional Properties table of the content pane, select **Output ports**

   This displays a list of the output ports in the content pane.
4. Select the output port that you want to change.
5. Specify appropriate (properties of the output port).
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

Extended messaging service settings

Use this page to enable or disable the extended messaging service.

The Extended Messaging Service provides run-time service for the support of extended messaging.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Extended Messaging Service** .

Startup

Specifies whether the server will attempt to start the extended messaging service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the extended messaging service automatically.

Cleared
The server does not try to start the extended messaging service. If extended messaging is to be used in applications that run on this server, the system administrator must start the extended messaging service manually or select this property then restart the server.

Late response handling extension collection

Use this page to view the configuration properties of late response handling extensions.

Late response handling extensions enable the handling of late responses with extended messaging

To view this administrative console page, click **Application Servers >** *server_name* **> Extended Messaging Service > Listener Port Extensions** .

Enabled

Specifies whether the handling of late responses is enabled.

Range

Selected
Handling of late responses is enabled.

Cleared
Handling of late responses is not enabled.

Request Interval

Specifies the interval that elaspes between checking for late responses.

Data type
Integer

Units milliseconds

Default
5

Range An integer number of milliseconds, greater than or equal to 0:

- 0 indicates that the late response handler continually checks for requests
- Other values are an integer number of milliseconds between checks for requests.

Request Timeout

Specifies the duration of time after which to give up waiting for a response.

Data type
Integer

Units seconds

Default
0

Range An integer number of milliseconds, greater than or equal to -1:
- -1 indicates that requests to handle late responses are never discarded.
- Other values are an integer number of milliseconds after which requests are discarded.

Listener Ports

Specifies the name of the listener port to be used to handle late responses.

Late response handling extension settings

Use this page to configure late response handling extensions.

To view this administrative console page, click **Application Servers >** *server_name* **> Extended Messaging Service > Listener Port Extensions >** *extension_name* .

Configuration tab

Enabled
Specifies whether the handling of late responses is enabled.

Range

Selected
Handling of late responses is enabled.

Cleared
Handling of late responses is not enabled.

Request Interval
Specifies the interval that elaspes between checking for late responses.

Data type
Integer

Units milliseconds

Default
5

Range An integer number of milliseconds, greater than or equal to -1:
- -1 indicates that requests to handle late responses are never discarded.
- Other values are an integer number of milliseconds after which requests are discarded.

Request Timeout
Specifies the duration of time after which to give up waiting for a response.

Data type
Integer

Units Seconds

Default
0

Range An integer number of milliseconds, greater than or equal to -1:
- -1 indicates that requests to handle late responses are never discarded.
- Other values are an integer number of milliseconds after which requests are discarded.

Listener Ports
Specifies the name of the listener port to be used to handle late responses.

Extended messaging provider settings

Use this page to manage extended messaging providers.

The extended messaging provider manages resources defined for use with extended messaging.

To view this administrative console page, click **Resources > Extended Messaging Providers** .

Name

The name of the resource provider.

Data type
String

Range 1 through 30 ASCII characters

Description

An optional description for the resource factory.

Data type
String

Input port collection

Use this page to view the configuration properties of input ports..

An input port specifies the properties needed by receiver beans as session beans. Receiver beans as message-driven beans do not need an input port, because the properties needed are associated with the deployed message-driven bean and the Message Listener service.

To view this administrative console page, click **Resources > Extended Messaging Providers > Input Port** .

Name: The name by which the input port is known for administrative purposes.

Data type
String

Units En_US ASCII characters

JNDI Name: The JNDI name for the resource.

Data type
String

Description: A description of the input port, for administrative purposes.

Data type
String

Category: A string that can be used to classify or group the resource.

Data type
String

Range 1 through 30 ASCII characters

JMS Connection Factory JNDI Name: The JNDI name for the JMS connection factory to be used by the input port; for example, `jms/connFactory1`.

Data type
String

JMS Destination JNDI Name: The JNDI name for the destination to be used by the input port; for example, `jms/destn1`.

Data type
String

JMS Acknowledgement Mode: JMS acknowledgment mode to be used for acknowledging messages.

This property applies only to message-driven beans that use bean-managed transaction demarcation (**Transaction type** is set to `Bean`).

Default
Auto Acknowledge

Range

Auto Acknowledge
The session automatically acknowledges a message in either of the following cases:
- When the session has successfully returned from a call to receive a message.
- When the session has called a message listener to process the message and received a successful response from that listener.

Dups OK Acknowledge
The session acknowledges only the delivery of messages. This is likely to result in the delivery of some duplicate messages if JMS fails, so it should be used only by consumers that are tolerant of duplicate messages.

Destination Type: The type of the JMS resource.

Default
Queue

Range

Queue
The receiver bean receives messages from a queue destination.

Topic The receiver bean receives messages from a topic destination.

Subscription durability: [Topic destinations only.] Specifies whether a JMS topic subscription is durable or non-durable.

Default
Durable

Range

Durable
A subscriber registers a durable subscription with a unique identity that is retained by JMS. Subsequent subscriber objects with the same identity resume the subscription in the state it was left in by the earlier subscriber. If there is no active subscriber for a durable subscription, JMS retains the subscription's messages until they are received by the subscription or until they expire.

Nondurable
Nondurable subscriptions last for the lifetime of their subscriber object. This means that a client sees the messages published on a topic only while its subscriber is active. If the subscriber is not active, the client is missing messages published on its topic.

Reply JMS Connection Factory JNDI Name: JNDI name of the JMS Connection Factory to be used for replies.

Data type
String

Reply JMS Destination JNDI Name: JNDI name of the JMS Destination to be used for replies.

Data type
String

Inport port settings

Use this page to configure an input port.

To view this administrative console page, click **Resources > Extended Messaging Providers > Input Port >** *inputport_name*.

Configuration tab

Name The name by which the input port is known for administrative purposes.

Data type
String

Units En_US ASCII characters

JNDI Name
The JNDI name for the resource.

Data type
String

Description
A description of the input port, for administrative purposes.

Data type
String

Category
A string that can be used to classify or group the resource.

Data type
String

Range 1 through 30 ASCII characters

JMS Connection Factory JNDI Name
The JNDI name for the JMS connection factory to be used by the input port; for example, `jms/connFactory1`.

Data type
String

JMS Destination JNDI Name
The JNDI name for the destination to be used by the input port; for example, `jms/destn1`.

Data type
String

JMS Acknowledgement Mode
JMS acknowledgment mode to be used for acknowledging messages.

This property applies only to message-driven beans that use bean-managed transaction demarcation (**Transaction type** is set to `Bean`).

Default
Auto Acknowledge

Range **Auto Acknowledge**

The session automatically acknowledges a message in either of the following cases:

- When the session has successfully returned from a call to receive a message.
- When the session has called a message listener to process the message and received a successful response from that listener.

Dups OK Acknowledge

The session acknowledges only the delivery of messages. This is likely to result in the delivery of some duplicate messages if JMS fails, so it should be used only by consumers that are tolerant of duplicate messages.

Destination Type
The type of the JMS resource.

Default
Queue

Range

Queue
The receiver bean receives messages from a queue destination.

Topic The receiver bean receives messages from a topic destination.

Subscription durability
[Topic destinations only.] Specifies whether a JMS topic subscription is durable or non-durable.

Default
Durable

Range

Durable

A subscriber registers a durable subscription with a unique identity that is retained by JMS. Subsequent subscriber objects with the same identity resume the subscription in the state it was left in by the earlier subscriber. If there is no active subscriber for a durable subscription, JMS retains the subscription's messages until they are received by the subscription or until they expire.

Nondurable

Nondurable subscriptions last for the lifetime of their subscriber object. This means that a client sees the messages published on a topic only while its subscriber is active. If the subscriber is not active, the client is missing messages published on its topic.

Reply JMS Connection Factory JNDI Name

JNDI name of the JMS Connection Factory to be used for replies.

Data type

String

Reply JMS Destination JNDI Name

JNDI name of the JMS Destination to be used for replies.

Data type

String

Output port collection

Use this page to view the configuration properties of output ports.

The Output port defines the parameters required by the extended messaging sender bean. These properties define the destination for the message being sent, together with optional details if a response is expected.

To view this administrative console page, click **Resources > Extended Messaging Providers > Output Port** .

Name: The name by which the output port is known for administrative purposes.

Data type

String

JNDI Name: The JNDI name for the output port.

Data type

String

Description: A description of the output port, for administrative purposes.

Data type

String

Category: A string that can be used to classify or group the resource.

Data type

String

JMS Connection factory JNDI name: The JNDI name for the JMS connection factory to be used by the output port; for example, `jms/connFactory1`.

Data type
String

Units En_US ASCII characters

Range 1 through 30 ASCII characters

JMS Destination JNDI name: The JNDI name for the destination to be used by the output port; for example, `jms/destn1`.

Data type
String

JMS Delivery Mode: Specifies whether all messages sent to the destination are persistent or non-persistent.

Default
Persistent

Range

Persistent
Messages put onto the destination are persistent.

Nonpersistent
Messages put onto the destination are not persistent.

JMS Priority: The message priority for this queue destination.

Data type
Integer

Default
4

Range 0 to 9

JMS Time To Live: The time in milliseconds after which messages on this queue expire.

Data type
Integer

Units Milliseconds

Default
0

Range 0 to*n*

0	messages never time out.
n	messages time out in *n* milliseconds.

JMS Disable Message I.D.: Specifies that the system should not generate a JMS message ID.

Default
Cleared

Range

Selected
The system does not generate message IDs.

Cleared
The system generates message IDs automatically.

JMS Disabled Message Time Stamp: Specifies that the system should not generate a JMS message timestamp.

Default
Cleared

Range

Selected
Message time stamps are added automatically to messages sent.

Cleared
Message time stamps are not added automatically to messages sent.

Response JMS Connection Factory JNDI name: The JNDI name for the JMS connection factory to be used for response messages handled by the output port; for example, `jms/connFactory1`.

Data type
String

Units En_US ASCII characters

Range 1 through 30 ASCII characters

Response JMS Destination JNDI name: The JNDI name for the destination to be used for response messages handled by the output port; for example, `jms/destn1`.

Data type
String

Output port settings

Use this page to configure an output port.

An output port specifies the properties needed by sender beans to define the destination for the message being sent, and other optional properties if a response is expected. The output port is associated with the sender bean at deployment time.

To view this administrative console page, click **Resources > Extended Messaging Providers > Output Port >** *outputport_name* .

Configuration tab

Name The name by which the output port is known for administrative purposes.

Data type
String

JNDI Name
The JNDI name for the output port.

Data type
String

Description
A description of the output port, for administrative purposes.

Data type
String

Category
A string that can be used to classify or group the resource.

Data type
String

JMS Connection factory JNDI name
The JNDI name for the JMS connection factory to be used by the output port; for example, `jms/connFactory1`.

Data type
String

Units En_US ASCII characters

Range 1 through 30 ASCII characters

JMS Destination JNDI name
The JNDI name for the destination to be used by the output port; for example, `jms/destn1`.

Data type
String

JMS Delivery Mode
Specifies whether all messages sent to the destination are persistent or non-persistent.

Default
Persistent

Range

Persistent
Messages put onto the destination are persistent.

Nonpersistent
Messages put onto the destination are not persistent.

JMS Priority
The message priority for this queue destination.

Data type
Integer

Default
4

Range 0 to 9

JMS Time To Live
The time in milliseconds after which messages on this queue expire.

Data type
Integer

Units Milliseconds

Default
0

Range 0 to *n*

0 messages never time out.

n messages time out in *n* milliseconds.

JMS Disable Message I.D.
Specifies that the system should not generate a JMS message ID.

Default
Cleared

Range

Selected
The system does not generate message IDs.

Cleared
The system generates message IDs automatically.

JMS Disabled Message Time Stamp
Specifies that the system should not generate a JMS message timestamp.

Default
Cleared

Range

Selected
Message time stamps are added automatically to messages sent.

Cleared
Message time stamps are not added automatically to messages sent.

Response JMS Connection Factory JNDI name
The JNDI name for the JMS connection factory to be used for response messages handled by the output port; for example, `jms/connFactory1`.

Data type
String

Units En_US ASCII characters

Range 1 through 30 ASCII characters

Response JMS Destination JNDI name
The JNDI name for the destination to be used for response messages handled by the output port; for example, `jms/destn1`.

Data type
String

Troubleshooting extended messaging

Use this overview task to help resolve a problem that you think is related to the extended messaging service.

The extended messaging service uses the standard IBMWebSphere Application Server RAS facilities. If you encounter a problem that you think might be related to the extended messaging service, complete the following stages:

Steps for this task

1. Check for extended messaging service messages in the application server's SystemOut log at *was_home*\logs*server*\SystemOut.

Any error messages associated with the extended messaging service are labelled with **EMSG**. The error message indicates the nature of the problem and provides some detail. The extended messaging service issues EMSG error messages if it fails to initialize, parse its configuration file, or encounters some runtime error.

2. Check for more messages in the application server's SystemOut log.

   If the JMS server is running, but you have problems accessing JMS resources, check the SystemOut log file, which should contain more error messages and extra details about the problem.
3. Check the Release Notes for specific problems and workarounds

   Tthe Release Notes, available from the IBM WebSphere Application Server library web site, is updated regularly to contain information about known defects and their workarounds. Check the latest version of the Release Notes for any information about your problem. If the Release Notes does not contain any information about your problem, you can also search the Technotes database on the IBM WebSphere Application Server web site.
4. Check for problems with the WebSphere Messaging functions or message-driven beans

   For more information about troubleshooting WebSphere Messaging, see the related topics listed at the bottom of this file.
5. **(Optional)** Get a detailed exception dump for extended messaging.

   If the information obtained in the preceding steps is still inconclusive, you can enable the application server debug trace for the ″Messaging″ group to provide a detailed exception dump.

Extended Messaging: Resources for learning

Use the following links to find relevant supplemental information about Extended Messaging. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- "Programming model and decisions"
- "Programming specifications"
- "Other" on page 33

Programming model and decisions
- Sun's Java Message Service (JMS) specification documentation

  http://developer.java.sun.com/developer/technicalArticles/Networking/messaging/

Programming specifications
- Java Message Service API, 1.0.2

  http://java.sun.com/products/jms/
- Enterprise JavaBeans Technology Downloads & Specifications

  http://java.sun.com/products/ejb/docs.html

Other

- WebSphere Application Server Enterprise Version 5 Overview: Extended J2EE Development Accelerators

  http://www-3.ibm.com/software/info1/websphere/index.jsp?tab=products/appserv_enterprise#extended
- Listing of PDF files to learn about WebSphere Application Server Version 5

  http://www-3.ibm.com/software/webservers/appserv/appserv_v5.html
- Listing of all IBM WebSphere Application Server Redbooks

  http://publib-b.boulder.ibm.com/Redbooks.nsf/Portals/WebSphere
- Listing of all IBM WebSphere Application Server Whitepapers

  http://www-4.ibm.com/software/webservers/appserv/whitepapers.html
- WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide

  http://www.redbooks.ibm.com/redbooks/SG246504.html

Chapter 2. Developing Web Services Gateway extensions

Use this task to develop your own extensions for the gateway

Before you begin

This information is intended for use by Java programmers.

To extend the functionality of the Web Services Gateway, you can write your own Java programs. The gateway does not provide any application programming interfaces, but there are system level interfaces that you can use. Specific guidance on how to do this in key areas is given in the following topics:

- "Writing a filter for the Web Services Gateway"
- "Using a filter to select a target service and port" on page 43
- "Capturing Web service invocation information from the Web Services Gateway" on page 45
- "Handling exceptions for the Web Services Gateway" on page 46

For additional technical details of the Web Services Gateway, see the Web Services Gateway Javadoc located in the InfoCenter.

Writing a filter for the Web Services Gateway

Use this task to write a filter for the Web Services Gateway.

Before you begin

To use this information you should be familiar with using a J2EE session bean development environment such as IBM WebSphere Studio Application Developer.

A Web Services Gateway filter is essentially a J2EE session bean implementing specific Home and Remote interfaces.

To write a filter using IBM WebSphere Studio Application Developer, complete the following steps. For more detailed information on writing session beans, see the WebSphere Studio Application Developer documentation topic entitled, "Developing enterprise beans - overview" (which can be accessed only when this topic is installed in a WebSphere Studio documentation plug-in)

Steps for this task

1. Open the J2EE perspective.
2. To create a new EJB application project, complete the following steps:
   a. Select **File -> New -> Enterprise Application Project**.
      The Project Creation wizard opens.
   b. In the Project Creation wizard, complete the following steps:
      1) Select the version of the J2EE specification that you want to use, then click **Next**.
      2) Type your project name.
      3) Clear the **Application client module** check box.

4) Clear the **Web module** check box.
5) Click **Finish**.

Your new EJB application project is created.

3. To add the extra JAR files that your EJB module needs that are not already in the Enterprise Application Server `/lib` directory, complete the following steps:
   a. Select **File -> Import**.
   b. Select the input source **File system**, then click **Next**.
   c. In the Import window, complete the following steps:
      1) Select `<em>WSGW_root</em>/client` as the source directory.

         where *WSGW_root* is the root directory for your installation of the gateway.
      2) Select `wsgwejb.jar`.
      3) Select the root directory of your new project as the **destination for imported resources**.
      4) Click **Finish**.
   d. **(Optional)** Repeat the previous **File -> Import** process to add any other extra JAR files that your EJB module needs.
   e. In the J2EE Hierarchy view, from the pop-up menu for your EJB module, select **Open With -> JAR Dependency Editor**.
   f. In the JAR Dependencies window, select all the JAR files listed.
   g. Close the JAR Dependencies window, then click **Yes** in the Save Resource window to save your changes.
4. To add extra JAR files to the Java build path for your EJB module, complete the following steps:
   a. In the J2EE Hierarchy view, select your EJB module's **Properties**.
   b. In the Properties window, ensure that the following JAR files are included on the **Java Build Path**:
      - `<em>WAS_root</em>/lib/jrom.jar`
      - `<em>WAS_root</em>/lib/qname.jar`
      - `<em>WAS_root</em>/lib/wsdl4j.jar`
      - `<em>WAS_root</em>/lib/wsif.jar`
      - `<em>WSGW_root</em>/client/wsgwejb.jar`

      where *WAS_root* is the root directory for your installation of IBM WebSphere Application Server Enterprise, and *WSGW_root* is the root directory for your installation of the gateway.
   c. Add any other JAR files or projects that you need for compiling your filter.
   d. Click **OK**.
5. To create the session bean, complete the following steps:
   a. Select **File -> New -> Enterprise Bean**.

      The Enterprise Bean Creation wizard opens.
   b. In the Enterprise Bean Creation wizard, complete the following steps:
      1) Select your EJB project, then click **Next**.
      2) Ensure that **Session Bean** is selected.
      3) Enter a name for the bean.
      4) Enter the package name for the bean as the **Default package**.
      5) Click **Next**.

c. In the Enterprise Bean Details window, complete the following steps:
   1) Accept the defaults offered for **Session type** (Stateless)and **Transaction type** (Container).
   2) Accept the defaults offered for **Bean supertype** (<none>), **Bean class** and **EJB binding name**.
   3) Confirm that **Local client view** is not enabled.
   4) For the **Remote client view: Remote Home Interface**, click **Class...** then select the `com.ibm.wsgw.beans.FilterHome` interface.
   5) For the **Remote client view: Remote Interface**, click **Class...** and select the `com.ibm.wsgw.beans.FilterRemote` interface.
   6) Click **Next**.

d. In the EJB Java Class window, complete the following steps:
   1) Select **Add Package**, then add these packages to the import statements:
      - `com.ibm.wsgw`
      - `com.ibm.wsgw.beans`
      - `org.apache.wsif`
   2) Click **Finish**.

Your new session bean is created.

6. The generated java code for your session bean does not implement the Filter. To update the code, complete the following steps:
   a. In the J2EE Hierarchy view, expand your session bean to show Java code entries for the Home interface, the Remote interface and for the session bean itself.
   b. In the J2EE Hierarchy view, double-click the entry for the session bean code. In the editor view, the generated code opens for editing.
   c. In the editor view, add ″, Filter″ to the end of the ″implements javax.ejb.SessionBean″ statement.
   d. Select **File -> Save** to save the file. Ignore any errors at this stage.
7. To add the unimplemented methods of the Filter interface to your session bean, complete the following steps:
   a. Open the Outline view (select **Window -> Show View -> Outline**).
   b. In the Outline view, from the pop-up menu for your session bean, select **Override Methods**.
   c. In the Override Methods window, select all the Filter methods to override then click **OK**.

   The methods of the Filter interface are added to your session bean.
8. Select **File -> Save** to save the file. Any errors from the previous **File -> Save** are resolved.
9. Develop your filter.

   The exact steps that you take to develop your filter depend upon what you want it to do. However to develop any filter, you use the following resources
   - The Filter interface.
   - The gateway Javadoc for the Filter interface.
   - The additional information on the Filter interface that is in “Web Services Gateway - the Filter interface” on page 39.

- The gateway message context. (This contains the context values for each message that comes into the Gateway. These are the values that your filter acts upon.)
- The gateway Javadoc for the GatewayContextNames class. (To use the gateway message context values, you import the GatewayContextNames class.)
- The additional information on the gateway message context fields that is in “Web Services Gateway - the gateway message context fields” on page 41.
- The Gateway WorkArea. (Filters use this to get and set the message context values, as described in “Web Services Gateway - the Filter interface” on page 39.)
- The WSIF Javadoc for the following WSIF objects:
  - WSIFRequest
  - WSIFResponse
  - WSIFMessage
  - WSIFException

  (the methods of the Filter interface use these objects, as described in “Web Services Gateway - the Filter interface” on page 39.)
- The example code below.

Note: You must observe the J2EE programming model, and ensure that any non-gateway services you use are available on all platforms that the filter might be expected to run on. For example, you should not use static variables to store state information because on certain platforms, or in certain configurations (such as a cluster), a filter might be invoked in a different JVM for each request.

Usage scenario

This example shows you how to access the context and get values in the filterRequest method of a filter.

```
import javax.naming.InitialContext;
import javax.naming.NamingException;

import com.ibm.websphere.workarea.UserWorkArea;
import com.ibm.websphere.workarea.WorkAreaException;

import com.ibm.wsgw.GatewayContextNames;

...

try
{
// Lookup the WorkArea Gateway context in JNDI
InitialContext ctx = new InitialContext();
UserWorkArea wsgwContext = (UserWorkArea)ctx.lookup("services:websphere/WSGW/workarea");

// Get the currently selected port name
String  Ptype = (wsgwContext.get( GatewayContextNames.TARGET_PORT_NAME)).getClass().getName();
String ThePortname = (String) wsgwContext.get( GatewayContextNames.TARGET_PORT_NAME);

// Get the currently selected target service WSDL location
String  Xtype = (wsgwContext.get( GatewayContextNames.TARGET_SERVICE_LOCATION)).getClass().getName();
TargetServiceLocation WSDLObject = (TargetServiceLocation)
wsgwContext.get(GatewayContextNames.TARGET_SERVICE_LOCATION);

String ServiceLocation = WSDLObject.serviceLocation;
int ServiceLocationType = WSDLObject.serviceLocationType;
String ServiceName = WSDLObject.serviceName;  String ServiceNamespace = WSDLObject.serviceNamespace;

}
catch ( NamingException e )
{
```

```
// Handle any exceptions thrown by the InitialContext here
}
catch ( WorkAreaException e )
{
// Handle exceptions thrown by UserWorkArea here
}
```

...

Note: In the preeding example, the line that begins with `TargetServiceLocation` wrapped due to the width of the page. Normally, it is one continuous line.

What to do next

After you have developed your filter, you need to generate deployment code and export the enterprise application. To do this using IBM WebSphere Studio Application Developer, complete the following steps:

1. Open the J2EE perspective.
2. In the J2EE Hierarchy view, from the pop-up menu for your EJB module, select **Generate -> Deploy and RMIC code**.
3. In the Generate Deploy and RMIC Code window, select the beans for which you want to generate code, then click **Finish**.
4. To configure the deployment descriptor properties for your bean, complete the following steps:
   a. In the J2EE Hierarchy view, from the pop-up menu for your bean, select **Open With -> EJB Deployment Descriptor**.
   b. On the **Beans** tab, set the JNDI name to the Filter class name. This name will be used as the ″Home Location″ when the filter is deployed to the gateway.
   c. Close the EJB Deployment Descriptor window, then click **Save** to save the changes.
5. In the J2EE Hierarchy view, from the pop-up menu for your project, select **Export EAR file** to export the enterprise application.

You are now ready to install your filter (as described in the penultimate step of Installing the gateway into a deployment manager cell and Installing the gateway into a standalone application server), then deploy your filter.

Web Services Gateway - the Filter interface

This topic gives more information on using each of the methods of the Filter interface. It supplements the information given in the following Javadoc:

- The gateway Javadoc for the Filter interface.
- The WSIF Javadoc for
  - WSIFRequest
  - WSIFResponse
  - WSIFMessage
  - WSIFException

The Filter interface represents an object which is called during service invocation. A bean which implements this interface can be registered to be called just before a request invocation, or just after response receipt for a particular service.

Gateway filters use the Gateway WorkArea to get and set the “Web Services Gateway - the gateway message context fields” on page 41. You get the gateway’s

WorkArea partition from `services:websphere/WSGW/workarea` in JNDI, then use the UserWorkArea API to get and set data within the message context.

These are the two main methods you use for developing your filter:
- FilterAction filterRequest(org.apache.wsif.WSIFRequest request, org.apache.wsif.WSIFResponse response).
- FilterAction filterResponse(org.apache.wsif.WSIFRequest request, org.apache.wsif.WSIFResponse response).

Another important method that requires a specific value to be returned is getContextVersion().

If you want your filter to change the WSIFReponse and WSIFRequest messages, then note that changes to messages are only recognized if the setUpdatedRequest and setUpdatedResponse methods are called on the returned FilterAction object. The FilterAction object can also dictate whether processing of the message should continue by calling the setContinueProcess method.

filterRequest Method

The filterRequest method is called by the gateway Manager just before a request is sent to a target service. The return value from the method can indicate that the request should not be sent.

The request parameter contains the request WSIFMessage. This consists of a set of named parts. Each part has a value which is encoded as an instance of an appropriate Java object. Filters can change the values of the Java object instances, but should not add or remove parts, or replace the values of parts with ones of a different type.

The Filter might decide that the request should not proceed. In that case it has three options:
- Throw a FilterException. The Gateway logs the exception but continues processing filters and the request invocation.
- Throw a WSGWException. The Gateway logs and rethrows the exception, and processing of filters and the request is stopped. The exeption then goes back to the receiving channel, and the channel must determine what to do with the exception (in the case of SOAP-based channels, this results in a Fault message back to the client). This should only be done for unexpected errors in the filter.
- Return a FilterAction object with the continueProcessing flag set to false. In this case the response message in the FilterAction can also be set, and is sent to the originator of the request. No further filters are invoked.

If the request or response is modified, then it must be returned in an instance of the FilterAction class. If this is not done, any change to the response is ignored by the Web Services Gateway.

filterResponse Method

The filterResponse method is called by the gateway Manager just after a response has been received from a target service. The response parameter contains the response or fault WSIFMessage. This consists of a set of named parts.

Each part has a value which is encoded as an instance of an appropriate Java object. Filters can change the values of the Java object instances, but should not add or remove parts, or replace the values of parts with ones of a different type.

The Filter may decide that the response should not proceed. In that case it has three options:

- Throw a FilterException. The Gateway logs the exception but continues processing filters and the response invocation.
- Throw a WSGWException. The Gateway logs and rethrows the exception, and processing of filters and the response is stopped. The exeption then goes back to the receiving channel, and the channel must determine what to do with the exception (in the case of SOAP-based channels, this results in a Fault message back to the client). This should only be done for unexpected errors in the filter.
- Return a FilterAction object with the continueProcessing flag set to false. In this case the fault WSIF Message is set in the response message in the FilterAction, and is sent to the originator of the request. No further filters are invoked.

If the response is modified, then it must be returned in an instance of the FilterAction class. If this is not done, any change to the response is ignored by the Web Services Gateway.

If the Filter throws a FilterException, it is logged, but the gateway continues to process other filters. If it throws a WSGWException, processing of the response is stopped.

getVersionString Method

The getVersionString method returns a string form of the version of the filter implementation. This is used by the gateway when logging events relating to the filter so that the exact version of the filter implementation is known.

getContextVersion Method

The getContextVersion method indicates the approach that this filter uses to access context information. To access the message context information for IBM WebSphere Application Server Version 5, this method must be implemented to return the value: `Filter.CONTEXT_VERSION_WORKAREA`.

init Method

The init method tells the filter that it has been configured with the Web Services Gateway.

This method is called by the gateway when it has been asked to add a filter.

destroy Method

The destroy method tells the filter that it is no longer configured with the Web Services Gateway. This method is called by the gateway when it has been asked to remove a filter.

Web Services Gateway - the gateway message context fields

The gateway message context contains the context information for each incoming message.

You can use the **Context Field Constant** values if you import the class `com.ibm.wsgw.GatewayContextNames`.

For basic information on the fields that are available in the context, see the Javadoc for the GatewayContextNames class. Additional information on all of these fields except `AUTH_SUBJECT` and `copyright` is given in the table below.

Note: In this version of the gateway you should treat all of the context fields as **Read only**. If your filter attempts to write to a context field, you do not get an error message (because the write does not actually fail) but subsequent system behavior is not readily predictable.

If you want to change the target service location and port name fields, then you should use the Routing interface to get the list of valid target service locations and to select the target service location. For more information, see "Using a filter to select a target service and port" on page 43.

Context Field	Context Field Constant	Description
WSGWAuthPassword	AUTH_PASSWORD	Read the password from the incoming HTTP request (where available)
WSGWAuthUserName	AUTH_USER_NAME	Read the user name from the incoming HTTP request (where available).
WSGWGatewayServiceName	GATEWAY_SERVICE_NAME	Name of the gateway Service for which the request was received.
WSGWReceivingChannelName	RECEIVING_CHANNEL_NAME	Name of the channel on which the request was received.
WSGWRetryCount	RETRY_COUNT	Number of retries for the request. NOT CURRENTLY USED.
WSGWSoapHeaders	SOAP_HEADERS	Retrieve the SOAP headers for an inbound soap request; SOAP Headers are returned as a Vector of Nodes.
WSGWTargetPortName	TARGET_PORT_NAME	Currently selected port name. Not set until after service invocation, and therefore can only be got by response filters and not by request filters. See also "Using a filter to select a target service and port" on page 43.
WSGWTargetServiceName	TARGET_SERVICE_LOCATION	Gives the location of the currently selected target service's WSDL, Name and Namespace. See also the Javadoc for the TargetServiceLocation class.
WSGWTimeoutTime	TIMEOUT_TIME	Time-out value for the response. NOT CURRENTLY USED.

Context Field	Context Field Constant	Description
WSGWMessageID	MESSAGE_ID	Set by the channel and is a server-unique ID that can be used to correlate messages, for example in trace. Can be made globally unique by prefixing with host name.

Using a filter to select a target service and port

Use this task to write a filter to select a target service and port.

When a request is received by the gateway, it must determine what the target service really is, and what port to use to access that service.

The Gateway represents each exported service as a gateway service. Each gateway service can map to one or more target services, but without filters there is no point in mapping multiple targets as the gateway will always pick the first one. If you want to map multiple targets, you also need to write pluggable filters (configured for each gateway service) that can select the target service from those available.

You write a filter as described in "Writing a filter for the Web Services Gateway" on page 35. Your filter can get the list of potential target services from the Routing service. It needs to select the target service, then call the Routing service to set the target service (note that doing this clears any prior selection of a target service's port). Your filter might also use the Routing service to select the target port for the service invocation.

The Routing service provides a non-standard interface which is defined in the topic "Web Services Gateway - the Routing interface" on page 44. The implementation of the Routing service is not pluggable.

The Home object for this service must implement the `com.ibm.wsgw.beans.RoutingHome` interface and be located in JNDI at `websphere/WSGW/Routing`.

The sequence of events for a filter to determine and set the target service is as follows:
1. The filter is called with a WSIFRequest.
2. The filter obtains the list of potential target services from the Routing service.
3. The filter selects the target service.
4. The filter calls the Routing service to set the target service (note that doing this clears any prior selection of a target service's port).

The Filter can also use the Routing service to select the target port for the service invocation.

Each target service is identified by the target service definition location (which is unique) and target service identity information (which might not be unique). So to select the target service, your filter can either get the table of mappings from target service location to identity information, then choose a target service to use; or it can call **setSelectedTargetServiceIdentity** with the required identity string (relying on the target service identity information being carefully defined). The routing service then selects the first target service it finds (for the current gateway service) with identity information that matches that specified (using String.equals()).

Note: When you use Routing to set the target service or the target port, the Routing service updates the request context. Because the request context has changed, you then need to set the request object in the **FilterAction** object that you return from the **filterRequest** method (see “Web Services Gateway - the Filter interface” on page 39).

Web Services Gateway - the Routing interface

This topic gives more information on using each of the methods of the Routing interface. It supplements the information given for this interface in the gateway Javadoc.

The Routing interface encapsulates a service which manages routing for requests. Filters can use this service to select the target service and port.

Note: The set methods all return a WSIFRequest object that contains the updated context information.

To get information on the currently selected target service, use the following Routing interface methods:

getSelectedTargetServiceLocation
This method returns the currently selected target service location for the request.

getTargetPortName
This method returns the currently selected target port name for the request.

Note: In this version of the gateway, this method always returns blank.

getTargetServiceDefinition
This method returns the currently selected target service definition for the request.

To set the target port, use the following Routing interface method:

setTargetPortName
This method sets the selected target port name for the request.

To get information on all potential target services, use the following Routing interface method:

getTargetServices
This method returns the set of target service names which are mapped by the gateway service on which the request was received.

To set the target service, use either of the following Routing interface methods:

setSelectedTargetServiceLocation
This method sets the selected target service location for the request. The selected port name if any is reset by this call.

setSelectedTargetServiceIdentity
This method sets the selected target service identity for the request. Target service identity need not be unique, so the first target service found with matching identity information is set. If none is found that matches, the method throws a WSGWException.

Capturing Web service invocation information from the Web Services Gateway

Use this task to help you to capture Web service invocation information from the Web Services Gateway.

The Web Services Gateway has not implemented a service that stores operational messages, but the gateway does contain an interface (the **MessageWarehouse** interface) to encapsulate such a service. This interface is driven by channels on receipt of requests and before sending responses.

If you have your own system for handling (classifying, storing and retrieving) operational messages, you can potentially use it to log the gateway's operational messages through "Web Services Gateway - the MessageWarehouse interface".

The Home object for this service must implement the `com.ibm.wsgw.beans.MessageWarehouseHome` interface and be located in JNDI at `websphere/WSGW/MessageWarehouse`.

Web Services Gateway - the MessageWarehouse interface

This topic gives more information on using each of the methods of the MessageWarehouse interface. It supplements the information given for this interface in the gateway Javadoc.

The MessageWarehouse interface encapsulates a service which stores messages for archiving. This interface is used by the channels to log incoming requests for the purposes of non-repudiation.

A default implementation of this interface is not provided by the Web Services Gateway. If no implementation is present the interface is not used.

logRequest

This method stores a request, along with information about the channel and originator of the request.

It is called by a channel when a request is received, after the user has been authenticated and the message decrypted. The channel may provide information to identify the originator of the request, and to identify the channel itself. The request itself is logged as a WSIFMessage.

Additional information regarding receipt of the request (for example any associated digital certificates) can also be logged as Serializable objects.

logResponse

This method stores a response, along with information about the channel and destination of the response.

It is called by a channel when a response is about to be sent, before the response is encrypted. The channel may provide information to identify the destination of the response, and to identify the channel itself. The response itself is logged as a WSIFMessage.

Additional information regarding sending of the response (for example any associated digital certificates) can also be logged as Serializable objects.

logException

This method stores a request, along with information about the channel and originator of the request in the event that an exception is thrown to the channel while the request is being processed. This method allows the exact request and exception information to be logged before the channel decides what actions to take as a result of the exception.

Additional information regarding request and exception (for example any associated digital certificates) can also be logged as Serializable objects.

Handling exceptions for the Web Services Gateway

Use this task to help you to capture information on the gateway's exception handling activities.

During normal processing of a Web service invocation, a fault message might be generated by the target service, and is passed back to the channel to be sent to the originator. As far as the Web Services Gateway is concerned there is no difference between processing a normal output message and processing a fault message.

But when an exception occurs during processing of a request, the channel needs some way to decide what to do with the exception. What is needed is a service that provides a pluggable handler that can look at the message, exception and other information to decide whether the exception should be thrown back to the originator, or whether a fault message should be constructed.

This service is not provided with the Web services Gateway, but the gateway does contain an interface to encapsulate such a service. "Web Services Gateway - the ExceptionHandler interface" allows channels to call an exception handling service, and allows the exceptions to be reported to a third party for analysis.

The Home object for this service must implement the `com.ibm.wsgw.beans.ExceptionHandlerHome` interface and be located in JNDI at `websphere/WSGW/ExceptionHandlerService`.

Web Services Gateway - the ExceptionHandler interface

This topic gives more information on using each of the methods of the ExceptionHandler interface. It supplements the information given in the gateway Javadoc for the ExceptionHandler interface.

The ExceptionHandler interface encapsulates a service which takes actions when exceptions occur during request/response processing in the gateway. It is intended for use by channels to allow a centralized facility to report and take actions when exceptions occur within the gateway.

handleException

This method is called by a channel when an exception is caught as a result of processing a message.

The return value indicates what action the channel should take. The actions include:

- Re-throw the original exception.
- Throw a new exception (this is thrown by the handler itself).
- Convert the exception into a fault message.

Note: If there is no ExceptionHandler installed, the “Capturing Web service invocation information from the Web Services Gateway” on page 45 (if any) is always used to log the exception, then the exception is rethrown.

Web Services Gateway: Resources for learning

Use the following links to find supplementary information about getting started with the Web Services Gateway. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Because the concept of a Web services gateway is so new, there is (as yet) very little supplementary information about it.

- Inside WebSphere Application Server 5

  http://advisor.com/doc/09808

  This article from WebSphere Advisor Magazine (August 2002) mentions the gateway as part of a general discussion of the new features in this version of WebSphere Application Server.
- Web Services Gateway

  http://www.alphaworks.ibm.com/tech/wsgw

  The Web Services Gateway area on the IBM alphaWorks® Web site. This area provides a discussion forum for early adopters.
- The IBM Web Services Gateway: Technical Overview

  http://www-3.ibm.com/software/integration/busconn/gateway.html

  A different version of the gateway is available as a component of a product called IBM WebSphere Business Connection. This brief technical summary from WebSphere Business Connection applies equally well to the version of the gateway in IBM WebSphere Application Server.

Chapter 3. Using EJB query

The EJB query language is used to specify a query over container-managed entity beans. The language is similar to SQL. An EJB query is independent of the bean's mapping to a persistent store.

An EJB query can be used in three situations:
- To define a finder method of an EJB entity bean.
- To define a select method of an EJB entity bean.
- To dynamically specify a query using the `executeQuery()` dynamic API.

Finder and select queries are specified in the bean's deployment descriptor using the `<ejb-ql>` tag. Queries specified in the deployment descriptor are compiled into SQL during deployment. Dynamic queries require the interface provided by IBM WebSphere Application Server Enterprise.

WebSphere's EJB query language is compliant with the EJB QL defined in Sun's EJB 2.0 specification and has additional capabilities as listed in the topic "Comparison of EJB 2.0 specification and WebSphere query language" on page 71.

In your WebSphere application, you can define an EJB query in the following ways:
- **Application Assembly Tool.** When assembling an EJB 2.0 entity bean, specify the `<ejb-ql>` tag for the finder() or select() method.
- **WebSphere Studio Application Developer**. When defining an entity bean, specify the `<ejb-ql>` tag for the finder or select method.
- **Dynamic query service.** Add the executeQuery() method to your application. The dynamic query API is provided as an Enterprise Extension to IBM WebSphere Application Server.

Before using EJB query, familiarize yourself with query language concepts, starting with the topic, "EJB query language".

Usage scenario

See "Example: EJB queries" on page 50.

EJB query language

An EJB query is a string that contains the following elements:
- a "SELECT clause" on page 67 that specifies the EJBs or values to return
- a "FROM clause" on page 52 that names the bean collections
- an optional "WHERE clause" on page 55 that contains search predicates over the collections
- an optional GROUP BY and HAVING clause (see "Aggregation functions" on page 66)
- an optional "ORDER BY clause" on page 67 that specifies the ordering of the result collection

The SELECT clause is optional in order to maintain compatibility with IBM WebSphere Application Server, Version 4.

Collections of entity beans are identified in EJB queries through the use of their abstract schema name in the query FROM clause.

The elements of EJB query language are discussed in more detail in the following related topics.

Example: EJB queries

Here is an example EJB schema, followed by a set of example queries:

Table 1. DeptBean schema

Entity bean name (EJB name)	DeptEJB (not used in query)
Abstract schema name	DeptBean
Implementation class	com.acme.hr.deptBean (not used in query)
Persistent attributes (cmp fields)	• deptno - Integer (key) • name - String • budget - BigDecimal
Relationships	• emps - 1:Many with EmpEJB • mgr - Many:1 with EmpEJB

Table 2. EmpBean schema

Entity bean name (EJB name)	EmpEJB (not used in query)
Abstract schema name	EmpBean
Implementation class	com.acme.hr.empBean (not used in query)
Persistent attributes (cmp fields)	• empid - Integer (key) • name - String • salary - BigDecimal • bonus - BigDecimal • hireDate - java.sql.Date • birthDate - java.util.Calendar • address - com.acme.hr.Address
Relationships	• dept - Many:1 with DeptEJB • manages - 1:Many with DeptEJB

Address is a serializable object used as cmp field in EmpBean. The definition of address is as follows:

```
   public class com.acme.hr.Address  extends Object implements Serializable {
public String street;
public String state;
public String city;
public Integer zip;
     public double distance (String start_location) { ... } ;
     public  String format ( ) { ... } ;
}
```

The following query returns all departments:

```
SELECT OBJECT(d) FROM DeptBean d
```

The following query returns departments whose name begins with the letters ″Web″. Sort the result by name:

```
SELECT OBJECT(d) FROM DeptBean d WHERE  d.name LIKE  'Web%' ORDER BY d.name
```

The keywords SELECT and FROM are shown in uppercase in the examples but are case insensitive. If a name used in a query is a reserved word, the name must be enclosed in double quotes to be used in the query. There is a list of reserved words later in this document. Identifiers enclosed in double quotes are case sensitive. This example shows how to use a cmp field that is a reserved word:

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d."select" > 5
```

The following query returns all employees who are managed by Bob. This example shows how to navigate relationships using a path expression:

```
SELECT OBJECT (e) FROM EmpBean e WHERE e.dept.mgr.name='Bob'
```

A query can contain a parameter which referes to the corresponding value of the finder or select method. Query parameters are numbered starting with 1:

```
SELECT OBJECT (e) FROM EmpBean e WHERE e.dept.mgr.name= ?1
```

This query shows navigation of a multivalued relationship and returns all departments that have an employee that earns at least 50000 but not more than 90000:

```
SELECT OBJECT(d) FROM DeptBean d,  IN (d.emps) AS e
WHERE e.salary BETWEEN 50000 and 90000
```

There is a join operation implied in this query between each department object and its related collection of employees. If a department has no employees, the department does not appear in the result. If a department has more than one employee that earns more than 50000, that department appears multiple times in the result.

The following query eliminates the duplicate departments:

```
SELECT DISTINCT OBJECT(d) from DeptBean d,  IN (d.emps) AS e  WHERE e.salary > 50000
```

Find employees whose bonus is more than 40% of their salary:

```
SELECT OBJECT(e) FROM EmpBean e where e.bonus > 0.40 * e.salary
```

Find departments where the sum of salary and bonus of employees in the department exceeds the department budget:

```
SELECT OBJECT(d) FROM DeptBean d where d.budget <
( SELECT SUM(e.salary+e.bonus) FROM IN(d.emps) AS e )
```

A query can contain DB2 style date-time arithmetic expressions if you use java.sql.* datatypes as CMP fields and your datastore is DB2. Find all employees who have worked at least 20 years as of January 1st, 2000:

```
SELECT OBJECT(e) FROM EmpBean e where year(  '2000-01-01' - e.hireDate ) >= 20
```

If the datastore is not DB2 or if you prefer to use java.util.Calendar as the CMP field, then you can use the java millsecond value in queries. The following query finds all employees born before Jan 1, 1990:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e.birthDate <  631180800232
```

Find departments with no employees:

```
SELECT OBJECT(d) from DeptBean d where d.emps IS EMPTY
```

Find all employees whose earn more than Bob:

```
SELECT OBJECT(e) FROM EmpBean e, EmpBean b
WHERE b.name = 'Bob' AND e.salary + e.bonus > b.salary + b.bonus
```

Find the employee with the largest bonus:

```
SELECT OBJECT(e) from EmpBean e  WHERE e.bonus =
(SELECT MAX(e1.bonus) from EmpBean e1)
```

The above queries all return EJB objects. A finder method query must always return an EJB Object for the home. A select method query can in addition return CMP fields or other EJB Objects not belonging to the home.

The following would be valid select method queries for EmpBean. Return the manager for each department:

```
SELECT  d.mgr FROM DeptBean d
```

Return department 42 manager's name:

```
SELECT  d.mgr.name FROM DeptBean d WHERE  d.deptno = 42
```

Return the names of employees in department 42:

```
SELECT e.name FROM EmpBean e WHERE  e.dept.deptno=42
```

Another way to write the same query is:

```
SELECT e.name from DeptBean d, IN (d.emps) AS e WHERE d.deptno=42
```

Finder and select queries allow only a single CMP field or EJBObject in the SELECT clause.

The dynamic query api allows multiple expressions in the SELECT clause. The following query would be a valid dynamic query, but not a valid select or finder query:

```
SELECT  e.name, e.salary+e.bonus as total_pay , object(e), e.dept.mgr
FROM  EmpBean e
ORDER BY 2
```

The following dynamic query returns the number of employees in each department:

```
SELECT e.dept.deptno as department_number , count(*) as employee_count
FROM  EmpBean e
GROUP BY  by e.dept.deptno
ORDER BY 1
```

The dynamic query api allows queries that contain bean or value object methods:

```
SELECT object(e), e.address.format( )
FROM EmpBean e EmpBean e
```

FROM clause

The FROM clause specifies the collections of objects to which the query is to be applied. Each collection is identified either by an abstract schema name and an identification variable, called a range variable, or by a collection member declaration that identifies a multivalued relationship and an identification variable.

Conceptually, the semantics of the query is to first form a temporary collection of tuples R. Tuples are composed of elements from the collections identified in the FROM clause. Each tuple contains one element from each of the collections in the FROM clause. All possible combinations are formed subject to the constraints imposed by the collection member declarations. If any schema name identifies a collection for which there are no records in the persistent store, then the temporary collection R will be empty.

Example: FROM clause

`DeptBean` contains records 10, 20 and 30 in the persistent store. `EmpBean` contains records 1, 2 and 3 that are related to department 10 and records 4, 5 that are related to department 20. Department 30 has no related employees.

```
FROM DeptBean d, EmpBean e
```

This forms a temporary collection R that contains 15 tuples.

```
FROM  DeptBean d,  DeptBean d1
```

This forms a temporary collection R that contains 9 tuples.

```
FROM  DeptBean d,  IN (d.emps) AS e
```

This forms a temporary collection R that contains 5 tuples. Department 30 because it contains no employees will not be in R. Department 10 will be contained in R three times and department 20 will be contained in R twice.

After forming the temporary collection the search conditions of the WHERE clause will be applied to R and this will yield a new temporary collection R1. The ORDER BY and SELECT clauses are applied to R1 to yield the final result set.

An identification variable is a variable declared in the FROM clause using the operator IN or the optional AS.

```
FROM DeptBean AS d,  IN (d.emps) AS e
```

is equivalent to:

```
FROM DeptBean d, IN (d.emps) e
```

An identification variable that is declared to be an abstract schema name is called a range variable. In the query above ″d″ is a range variable. An identification variable that is declared to be a multivalued path expression is called a collection member declaration. ″d″ and ″e″ in the example above are collection member declarations.

Note that the following path expression is illegal as a collection member declaration because it is not multivalued:

```
e.dept.mgr
```

Inheritance in EJB query

If an EJB inheritance hierarchy has been defined for an abstract schema, using the abstract schema name in a query statement implies the collection of objects for that abstract schema as well as all subtypes.

Example: Inheritance

Suppose that bean `ManagerBean` is defined as a subtype of `EmpBean` and `ExecutiveBean` is a subtype of `ManagerBean` in an EJB inheritance hierarchy. The following query returns employees as well as managers and executives:

```
SELECT OBJECT(e) FROM EmpBean e
```

Path expressions

An identification variable followed by the navigation operator (.) and a cmp or relationship name is a path expression.

A path expression that leads to a cmr field can be further navigated if the cmr field is single-valued. If the path expression leads to a multi-valued relationship, then the path expression is terminal and cannot be further navigated. If the path expression leads to a cmp field whose type is a value object, it is possible to navigate to attributes of the value object.

Example: Value object

Assume that `address` is a cmp field for `EmpBean`, which is a value object.

```
SELECT  object(e)  FROM EmpBean e
WHERE  e.address.distance('San Jose') < 10  and e.address.zip = 95037
```

It is best to use the composer pattern to map value object attributes to relational columns if you intend to search on value attributes. If you store value objects in serialized format, then each value object must be retrieved from the database and deserialized. Value object methods can only be done in dynamic queries.

A path expression can also navigate to a bean method. The method must be defined on either the remote or local bean interface. Methods can only be used in dynamic queries. You cannot mix both remote and local methods in a single query statement.

If the query contains remote methods, the dynamic query must be executed using the query remote interface. Using the query remote interface causes the query service to activate beans and create instances of the remote bean interface

Likewise, a query statement with local bean methods must be executed with the query local interface. This causes the query service to activate beans and local interface instances.

Do not use get methods to access cmp and cmr fields of a bean.

If a method has overloaded definitions, the overloaded methods must have different number of parameters.

Methods must have non-void return types and method arguments and return types must be either primitive types byte, short, int, long, float, double, boolean, char or wrapper types from the following list:

Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Date

If any input argument to a method is NULL, it is assumed the method returns a NULL value and the method is not invoked.

A collection valued path expression can be used in the FROM clause as a collection member declaration, and with the IS EMPTY, MEMBER OF, and EXISTS predicates in the WHERE clause.

`FROM EmpBean e WHERE e.dept.mgr.name='Bob'`	OK

`FROM EmpBean e WHERE e.dept.emps.name='BOB'`	INVALID — cannot navigate through emps because it is multivalued
`FROM EmpBean e,  IN (e.dept.emps) e1 WHERE e1.name='BOB'`	OK
`FROM EmpBean e WHERE e.dept.emps IS EMPTY`	OK

WHERE clause

The WHERE clause contains search conditions composed of the following:

- literal values
- input parameters
- expressions
- basic predicates
- quantified predicates
- BETWEEN predicate
- IN predicate
- LIKE predicate
- NULL predicate
- EMPTY collection predicate
- MEMBER OF predicate
- EXISTS predicate
- IS OF TYPE predicate

If the search condition evaluates to TRUE, the tuple is added to the result set.

Literals

A string literal is enclosed in single quotes. A single quote that occurs within a string literal is represented by two single quotes; For example: 'Tom''s'. A string literal cannot exceed the maximum length that is supported by the underlying persistent datastore.

A numeric literal can be any of the following:

- an exact value such as 57, -957, +66
- any value supported by Java long
- a decimal literal such as 57.5, -47.02
- an approximate numeric value such as 7E3, -57.4E-2

A decimal or approximate numeric value must be in the range supported by Java double.

A boolean literal can be the keyword TRUE or FALSE and is case insensitive.

Input parameters

Input parameters are designated by the question mark followed by a number; For example: `?2`

Input parameters are numbered starting at 1 and correspond to the arguments of the finder or select method; therefore, a query must not contain an input parameter that exceeds the number of input arguments.

An input parameter can be a primitive type of byte, short, int, long, float, double, boolean, char or wrapper types of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Char, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp or an EJBObject.

An input parameter must not have a NULL value. To search for the occurrence of a NULL value the "NULL predicate" on page 61 should be used.

Expressions

Conditional expressions can consist of comparison operators and logical operators (AND, OR, NOT).

Arithmetic expressions can be used in comparison expressions and can be composed of arithmetic operations and functions, path expressions that evaluate to a numeric value and numeric literals and numeric input parameters.

String expressions can be used in comparison expressions and can be composed of string functions, path expressions that evaluate to a string value and string literals and string input parameters. A cmp field of type char is handled as if it were a string of length 1.

Boolean expressions can be used with = and <> comparison and can be composed of path expressions that evaluate to a boolean value and TRUE and FALSE keywords and boolean input parameters.

Reference expressions can be used with = and <> comparison and can be composed of path expressions that evaluate to a cmr field, an identification variable and an input parameter whose type is an EJB reference

Four different expression types are supported for working with date-time types. For portability the java.util.Calendar type should be used. DB2® style date, time and timestamp expressions are supported if the datastore is DB2 and the CMP field is of type java.util.Date, java.sql.Date, java.sql.Time or java.sql.Timestamp.

A Calendar type can be compared to another Calendar type, an exact numeric literal or input parameter of type long whose value is the standard Java long millisecond value.

The following query finds all employees born before Jan 1, 1990:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e.birthDate <  631180800232
```

Date expressions can be used in comparison expressions and can be composed of operators + - , date duration expressions and date functions, path expressions that evaluate to a date value, string representation of a date and date input parameters.

Time expressions can be used in comparison expressions and can be composed of operators + - , time duration expressions and time functions, path expressions that evaluate to a time value, string representation of time and time input parameters.

Timestamp expressions can be used in comparison expressions and can be composed of operators + - , timestamp duration expressions and timestamp functions, path expressions that evaluate to a timestamp value, string representation of a timestamp and timestamp input parameters.

Standard bracketing ( ) for ordering expression evaluation is supported.

The operators and their precedence order from highest to lowest are:

- Navigation operator (.)
- Arithmetic operators in precedence order:
  - + - unary
  - * / multiply, divide
  - + - add, subtract
- Comparison operators: =, >, <, >=, <=, <>(not equal)
- Logical operator NOT
- Logical operator AND
- Logical operator OR

Null value semantics: The following describe the semantics of NULL values:

- Comparison or arithmetic operations with an unknown (NULL) value yield an unknown value
- Path expressions that contain NULL evaluate to NULL
- The IS NULL and IS NOT NULL operators can be applied to path expressions and return TRUE or FALSE. Boolean operators AND, OR and NOT use three valued logic.

AND	**True**	**False**	**Unknown**
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

OR	**True**	**False**	**Unknown**
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

	NOT
True	False
False	True
Unknown	Unknown

Example: Null value semantics

```
select object(e) from EmpBean where e.salary > 10  and e.dept.budget > 100
```

If salary is NULL the evaluation of `e.salary > 10` returns unknown and the employee object is not returned. If the cmr field dept or budget is NULL evalution of `e.dept.budget > 100` returns unknown and the employee object is not returned.

```
select object(e) from EmpBean where e.dept.budget is null
```

If dept or budget is NULL evaluation of `e.dept.budget` is null returns TRUE and the employee object is returned.

```
select object(e) from EmpBean e ,  in (e.dept.emps) e1 where e1.salary > 10
```

If dept is NULL, then the multivalued path expression `e.dept.emps` results in an empty collection (not a collection that contains a NULL value). An employee with a null dept value will not be returned.

```
select object(e) from EmpBean e where  e.dept.emps is empty
```

If dept is NULL the evaluation of the predicate in unknown and the employee object is not returned.

```
select object(e) from EmpBean e , EmpBean e1  where e member of  e1.dept.emps
```

If dept is NULL evaluation of the member of predicate returns unknown and the employee is not returned.

Date time arithmetic and comparisons: DATE, TIME and TIMESTAMP values may be compared with another value of the same type. Comparisons are chronological. Date time values can also be incremented, decremented, and subtracted.

If the datastore is DB2, then DB2 string representation of DATE, TIME and TIMESTAMP types can also be used. A string representation of a date or time can use ISO, USA, EUR or JIS format. A string representation of a timestamp uses ISO format.

Format	Date format	Date examples	Time format	Time examples
ISO	yyyy-mm-dd	1987-02-24 1987-2-24	hh.mm.ss	13.50.00 13.50
USA	mm/dd/yyyy	2/24/1987	hh:mm AM or PM	1:50 pm 02:10 AM
EUR	dd.mm.yyyy	24.02.1987 24.2.1987	hh.mm.ss	13.50.00 13.55
JIS	yyyy-mm-dd	1987-02-24	hh:mm:ss	13:50 13:50:05

Example 1: Date time arithmetic comparisons

```
e.hiredate > '1990-02-24'
```

The timestamp of February 24th, 1990 1:50 pm can be represented as follows:

```
'1990-02-24-13.50.00.000000'  or
'1990-02-24-13.50.00'
```

If the datastore is DB2, DB2 decimal durations can be used in expressions and comparisons. A date duration is a decimal(8,0) number that represents the difference between two dates in the format YYYYMMDD. A time duration is a decimal(6,0) number that represents the difference between two time values as HHMMSS. A timestamp duration is a decimal(20,6) number representing the differences between two timestamp values as YYYYMMDDHHMMSS.ZZZZZZ (ZZZZZZ is the number of microseconds and is to the right of the decimal point) .

Two date values (or time values or timestamp values) can be subtracted to yield a duration. If the second operand is greater than the first the duration is a negative decimal number. A duration can be added or subtracted from a datetime value to yield a new datetime value.

Example 2: Date time arithmetic comparisons

`DATE('3/15/2000') - '12/31/1999'` results in a decimal number 215 which is a duration of 0 years, 2 months and 15 days.

Durations are really decimal numbers and can be used in arithmetic expressions and comparisons.

`( DATE('3/15/2000') - '12/31/1999' ) + 14 > 215` evaluates to TRUE.

`DATE('12/31/1999') + DECIMAL(215,8,0)` results in a date value 3/15/2000.

`TIME('11:02:26') - '00:32:56'` results in a decimal number 102930 which is a time duration of 10 hours, 29 minutes and 30 seconds.

`TIME('00:32:56') + DECIMAL(102930,6,0)` results in a time value of 11:02:26.

`TIME('00:00:59') + DECIMAL(240000,6,0)` results in a time value of 00:00:59.

`e.hiredate + DECIMAL(500,8,0) > '2000-10-01'` means compare the hiredate plus 5 months to the date 10/01/2000.

Basic predicates

Basic predicates can be of two forms

```
expression-1  comparison-operator  expression-2
expression-3  comparison-operator ( subselect )
```

The subselect must not return more than one value and the subselect can not return a type of an EJB reference. Boolean types and reference types only support = and <> comparisons.

Example: Basic predicates

```
d.name='Java Development'
e.salary > 20000
e.salary > ( select avg(e.salary) from EmpBean e)
```

Quantified predicates

A quantified predicate compares a value with a set of values produced by a subselect.

```
expression   comparison-operator   SOME  |  ANY  |   ALL    ( subselect )
```

The expression must not evaluate to a reference type.

When SOME or ANY is specified the result of the predicate is as follows:

- TRUE if the comparison is true for at least one value returned by the subselect.
- FALSE if the subselect is empty or if the comparison is false for every value returned by the subselect.
- UNKNOWN if the comparison is not true for all of the values returned by the subselect and at least one of the comparisons is unknown because of a null value.

When ALL is specified the result of the predicate is as follows:

- TRUE if the subselect returns empty or if the comparison is true to every value returned by the subselect.
- FALSE if the comparison is false for at least one value returned by the subselect.
- UNKNOWN if the comparison is not false for all values returned by the subselect and at least one comparison is unknown because of a null value.

BETWEEN predicate

The BETWEEN predicate determines whether a given value lies between two other given values.

```
expression  [NOT]  BETWEEN expression-2  AND expression-3
```

Example: BETWEEN predicate

```
e.salary BETWEEN 50000 AND 60000
```

is equivalent to:

```
e.salary >= 50000  AND e.salary <= 60000
e.name NOT BETWEEN 'A' AND 'B'
```

is equivalent to:

```
e.name < 'A'  OR e.name > 'B'
```

IN predicate

The IN predicate compares a value to a set of values and can have one of two forms:

```
expression [NOT] IN  ( subselect )
expression [NOT] IN  ( value1, value2,  .... )
```

ValueN can either be a literal value or an input parameter. The expression can not evaluate to a reference type.

Example: IN predicate

```
e.salary IN ( 10000, 15000 )
```

is equivalent to

```
( e.salary = 10000  OR  e.salary = 15000 )
e.salary IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary = ANY   ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
e.salary NOT IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary <> ALL    ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

LIKE predicate

The LIKE predicate searches a string value for a certain pattern.

```
string-expression   [NOT]  LIKE  pattern    [ ESCAPE escape-character ]
```

The pattern value is a string literal or parameter marker of type string in which the underscore (_) stands for any single character and percent (%) stands for any sequence of characters (including empty sequence). Any other character stands for itself. The escape character can be used to search for character _ and %. The escape character can be specified as a string literal or an input parameter.

If the string-expression is null, then the result is unknown.

If both string-expression and pattern are empty, then the result is true.

Example: LIKE predicate

- '' LIKE '' is true

- `''` `LIKE` `'%'` is true
- `e.name LIKE '12%3'` is true for '123' '12993' and false for '1234'
- `e.name LIKE 's_me'` is true for 'some' and 'same', false for 'soome'
- `e.name LIKE '/_foo' escape '/'` is true for '_foo', false for 'afoo'
- `e.name LIKE '//_foo' escape '/'` is true for '/afoo' and for '/bfoo'
- `e.name LIKE '///_foo' escape '/'` is true for '/_foo' but false for '/afoo'

NULL predicate

The NULL predicate tests for null values.

```
single-valued-path-expression IS [NOT] NULL
```

Example: NULL predicate

```
e.name IS NULL
```

```
e.dept.name IS NOT NULL
```

```
e.dept IS NOT NULL
```

EMPTY collection predicate

To test if a multivalued relationship is empty, use the following syntax:

```
collection-valued-path-expression  IS [NOT]  EMPTY
```

Example: Empty collection predicate

To find all departments with no employees:

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d.emps IS EMPTY
```

MEMBER OF predicate

This expression tests whether the object reference specified by the single valued path expression or input parameter is a member of the designated collection. If the collection valued path expression designates an empty collection the value of the MEMBER OF expression is FALSE.

```
{ single-valued-path-expression | input_parameter } [ NOT ] MEMBER [ OF ]
collection-valued-path-expression
```

Note: The preceeding example wrapped onto two lines due to the width of the page.

Example: MEMBER OF predicate

Find employees that are not members of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Find employees whose manager is a member of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e, DeptBean d
WHERE e.dept.mgr MEMBER  OF d.emps  and d.deptno=?1
```

EXISTS predicate

The exists predicate tests for the presence or absence of a condition specified by a subselect.

```
EXISTS ( subselect )
```

```
EXISTS collection-valued-path-expression
```

The result of EXISTS is true if the subselect returns at least one value or the path expression evaluates to a nonempty collection, otherwise the result is false.

To negate an EXISTS predicate, precede it with the logical operator NOT.

Example: EXISTS predicate

Return departments that have at least one employee earning more than 1000000:

```
SELECT  OBJECT(d) FROM  DeptBean d
WHERE EXISTS ( SELECT  1  FROM IN (d.emps) e WHERE  e.salary > 1000000 )
```

Return departments that have no employees:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS  ( SELECT 1 FROM IN (d.emps) e)
```

The above query can also be written as follows:

```
SELECT OBJECT(d) FROM DeptBean d WHERE NOT EXISTS d.emps
```

IS OF TYPE predicate

The IS OF TYPE predicate is used to test the type of an EJB reference. It is similar in function to the Java instance of operator. IS OF TYPE is used when several abstract beans have been grouped into an EJB inheritance hierarchy. The type names specified in the predicate are the bean abstract names. The ONLY option can be used to specify that the reference must be exactly this type and not a subtype.

```
identification-variable IS OF TYPE ( [ONLY] type-1, [ONLY] type-2, ..... )
```

Example: IS OF TYPE predicate

Suppose that bean `ManagerBean` is defined as a subtype of `EmpBean` and `ExecutiveBean` is a subtype of `ManagerBean` in an EJB inheritance hierarchy.

The following query returns employees as well as managers and executives:

```
SELECT  OBJECT(e) FROM EmpBean e
```

If you are interested in objects which are employees and not managers and not executives:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e IS OF TYPE( ONLY EmpBean )
```

If you are interested in object which are managers or executives:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e IS OF TYPE( ManagerBean)
```

The above query is equivalent to the following query:

```
SELECT  OBJECT(e) FROM ManagerBean e
```

If you are interested in managers only and not executives:

```
SELECT  OBJECT(e) FROM  EmpBean e WHERE e IS OF TYPE( ONLY ManagerBean)
```

or:

```
SELECT  OBJECT(e) FROM  ManagerBean e
WHERE  e IS OF TYPE (ONLY ManagerBean)
```

Scalar functions

EJB query contains scalar built-in functions for doing type conversions, string manipulation, and for manipulating date-time values. The list of scalar functions is documented in the topic “EJB query: Scalar functions” on page 63.

Example: Scalar functions

Find employees hired in 1999:

```
SELECT OBJECT(e) FROM EmpBean e where YEAR(e.hireDate) = 1999
```

The only scalar functions that are guaranteed to be portable across backend datastore vendors are the following:

- ABS
- SQRT
- CONCAT
- LENGTH
- LOCATE
- SUBSTRING

The other scalar functions should be used only when DB2 is the backend datastore.

EJB query: Scalar functions

EJB query contains scalar built-in functions, as listed below, for doing type conversions, string manipulation, and for manipulating date-time values.

Numeric functions

```
ABS ( < any numeric datatype > ) -> < any numeric datatype >
```

```
SQRT ( < any numeric datatype > ) -> Double
```

Type conversion functions

```
CHAR ( < any  numeric datatype > ) ->  string
CHAR ( <  string  > ) ->  string
CHAR ( < any datetime  datatype >  [, Keyword k ]) ->  string
```

Datetime datatype is converted to its string representation in a format specified by the keyword k. The valid keywords values are ISO, USA, EUR or JIS. If k is not specified the default is ISO.

```
BIGINT ( < any numeric datatype > ) -> Long
BIGINT ( < string > ) -> Long
```

The following function converts the argument to an integer n by truncation and returns the date that is n-1 days after January 1, 0001:

```
DATE ( < date string > ) -> Date
DATE (  < any numeric datatype>) -> Date
```

The following function returns date portion of a timestamp:

```
DATE( timestamp ) -> Date
DATE ( < timestamp-string > ) -> Date
```

The following function converts number to decimal with optional precision p and scale s.

```
DECIMAL ( < any numeric datatype > [, p [ ,s ] ] ) -> Decimal
```

The following function converts string to decimal with optional precision p and scale s.

```
DECIMAL ( < string > [ , p [ , s ] ] ) -> Decimal
```

```
DOUBLE ( < any numeric datatype > ) -> Double
DOUBLE ( < string > ) -> Double
```

```
FLOAT ( < any numeric datatype > ) -> Double
FLOAT ( < string > ) -> Double
```

Float is a synonym for DOUBLE.

```
INTEGER ( < any numeric datatype > ) -> Integer
INTEGER ( < string > ) -> Integer
```

```
REAL ( < any numeric datatype > ) -> Float
```

```
SMALLINT ( < any numeric datatype ) -> Short
SMALLINT ( < string > ) -> Short
```

```
TIME ( < time > ) -> Time
TIME ( < time-string  > ) -> Time
TIME ( < timestamp > ) -> Time
TIME ( < timestamp-string  > ) -> Time
```

```
TIMESTAMP ( < timestamp > ) -> Timestamp
TIMESTAMP ( < timestamp-string > ) -> Timestamp
```

String functions

```
CONCAT ( <string>, <string>  ) -> String
```

The following function returns a character string representing absolute value of the argument not including its sign or decimal point. For example, `digits( -42.35)` is "4235".

```
DIGITS ( Decimal d  ) -> String
```

The following function returns the length of the argument in bytes. If the argument is a numeric or datetime type, it returns the length of internal representation.

```
LENGTH ( < string >  ) -> Integer
```

The following function returns a copy of the argument string where all upper case characters have been converted to lower case.

```
LCASE ( < string > ) -> String
```

The following function returns the starting position of the first occurrence of argument 1 inside argument 2 with optional start position. If not found, it returns 0.

```
LOCATE ( String s1 , String s2  [, Integer start ] ) -> Integer
```

The following function returns a substring of s beginning at character m and containing n characters. If n is omitted, the substring contains the remainder of string s. The result string is padded with blanks if needed to make a string of length n.

```
SUBSTRING ( String s ,  Integer m [ , Integer n ] ) -> String
```

The following function returns a copy of the argument string where all lower case characters have been converted to upper case.

```
UCASE ( < string > ) -> String
```

Date - time functions

The following function returns the day portion of its argument. For a duration, the return value can be -99 to 99.

```
DAY (  Date ) ->  Integer
DAY ( < date-string > ) ->  Integer
DAY ( < date-duration > ) -> Integer
DAY ( Timestamp  ) ->  Integer
DAY ( < timestamp-string > ) ->  Integer
DAY ( < timestamp-duration > ) -> Integer
```

The following function returns one more than number of days from January 1, 0001 to its argument.

```
DAYS ( Date  ) ->  Integer
DAYS ( < Date-string > ) ->  Integer
DAYS ( Timestamp  ) ->  Integer
DAYS ( < timestamp-string > ) ->  Integer
```

The following function returns the hour part of its argument. For a duration, the return value can be -99 to 99.

```
HOUR ( Time ) -> Integer
HOUR ( < time-string > ) -> Integer
HOUR ( < time-duration > ) ->  Integer
HOUR ( Timestamp ) -> Integer
HOUR ( < timestamp-string > ) -> Integer
HOUR ( < timestamp-duration >  ) -> Integer
```

The following function returns the microsecond part of its argument.

```
MICROSECOND ( Timestamp ) -> Integer
MICROSECOND ( < timestamp-string > ) -> Integer
MICROSECOND ( < timestamp-duration >  ) -> Integer
```

The following function returns the minute part of its argument. For a duration, the return value can be -99 to 99.

```
MINUTE ( Time ) -> Integer
MINUTE ( < time-string > ) -> Integer
MINUTE ( < time-duration > ) ->  Integer
MINUTE ( Timestamp ) -> Integer
MINUTE ( < timestamp-string > ) -> Integer
MINUTE ( < timestamp-duration >  ) -> Integer
```

The following function returns the month portion of its argument. For a duration, the return value can be -99 to 99.

```
MONTH (  Date ) ->  Integer
MONTH ( < date-string > ) ->  Integer
MONTH ( < date-duration > ) -> Integer
MONTH ( Timestamp  ) ->  Integer
MONTH ( < timestamp-string > ) ->  Integer
MONTH ( < timestamp-duration > ) -> Integer
```

The following function returns the second part of its argument. For a duration, the return value can be -99 to 99.

```
SECOND ( Time ) -> Integer
SECOND ( < time-string > ) -> Integer
SECOND ( < time-duration > ) ->  Integer
SECOND ( Timestamp ) -> Integer
SECOND ( < timestamp-string > ) -> Integer
SECOND ( < timestamp-duration >  ) -> Integer
```

The following function returns the year portion of its argument. For a duration, the return value can be -9999 to 9999.

```
YEAR (  Date ) ->  Integer
YEAR ( < date-string > ) ->  Integer
YEAR ( < date-duration > ) -> Integer
YEAR ( Timestamp  ) ->  Integer
YEAR ( < timestamp-string > ) ->  Integer
YEAR ( < timestamp-duration > ) -> Integer
```

Aggregation functions

Queries that return aggregate values can only be used with the dynamic query interface available in IBM WebSphere Application Server Enterprise. However, aggregation functions can be used in non-dynamic queries if the aggregation function is used in a subselect or HAVING clause.

Aggregation functions operate on a set of values to return a single scalar value. The following is an example of an aggregation:

```
SELECT  SUM (e.salary + e.bonus) FROM EmpBean e WHERE e.dept.deptno =20
```

This computes the total salary and bonus for department 20.

The aggregation functions are avg, count, max, min and sum. The syntax of an aggregation function is as follows:

```
aggregation-function  (    [ ALL |  DISTINCT ]  expression )
```

or:

```
COUNT( * )
```

The DISTINCT option eliminates duplicate values before applying the function. ALL is the default and does not eliminate duplicates. Null values are ignored in computing the aggregate function except for COUNT(*) which returns a count of all elements in the set.

MAX and MIN can apply to any numeric, string or datetime datatype and returns the same datatype. SUM and AVG take a numeric type as input. SUM and AVG return numeric type. The actual numeric type returned by SUM and AVG depends on the underlying datastore. COUNT can take any datatype and returns an integer.

The set of values that is used for the aggregate function is determined by the collection that results from the FROM and WHERE clause of the subquery. This set can be divided into groups and the aggregation function applied to each group. This is done by using a GROUP BY clause in the subquery. The GROUP BY clause defines grouping members which is a list of path expressions. Each path expression specifies a field that is a primitive type of byte, short, int, long, float, double, boolean, char, or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time or java.sql.Timestamp.

Finder or select queries can not return aggregation function values. In other words, aggregation functions can not appear in the top level SELECT of a finder or select query but can be used in subqueries.

Example: Aggregation functions

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e GROUP BY e.dept.deptno
```

The above query computes the average salary for each department.

In dividing a set into groups, a NULL value is considered equal to another NULL value.

Just as the WHERE clause filters tuples from the FROM clause, the groups can be filtered using a HAVING clause that tests group properties involving aggregate functions or grouping members:

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING  COUNT(*) > 3  AND  e.dept.deptno > 5
```

This query returns average salary for departments that have more than 3 employees and the department number is greater than 5.

It is possible to have a HAVING clause without a GROUP BY clause in which case the entire set is treated as a single group to which the HAVING clause is applied.

SELECT clause

For finder and select queries, the syntax of the SELECT clause is as follows:

```
SELECT  [  ALL | DISTINCT  ]
 { single-valued-path-expression  |  OBJECT ( identification-variable )  }
```

The SELECT clause consists of either a single identification variable that is defined in the FROM clause or a single valued path expression that evaluates to a object reference or CMP value. The keyword DISTINCT can be used to eliminate duplicate references.

For a query that defines a finder method the query must return an object type consistent with the home for which the finder method associated with the query. In other words, a finder method for a department home can not return employee objects.

For dynamic queries the syntax is as follows:

```
SELECT { ALL | DISTINCT }  [ selection , ]*  selection
selection  ::= { expression [[AS] id ]  |  scalar-subselect }
```

A scalar-subselect is a subselect that returns a single value.

Example: SELECT clause

Find all employees that earn more than John:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean e
WHERE  ej.name = 'John'  and e.salary > ej.salary
```

Find all departments that have one or more employees who earn less than 20000:

```
SELECT DISTINCT  e.dept  FROM EmpBean e where e.salary < 20000
```

A select method query can have a path expression that evaluates to an arbitrary value:

```
SELECT  e.dept.name  FROM EmpBean e where e.salary < 2000
```

The above query returns a collection of name values for those departments having employees earning less than 20000.

Example: Valid dynamic queries

The following are examples of dynamic queries:

```
SELECT e.name, e.salary+e.bonus as total_pay from EmpBean e
```

```
SELECT  SUM( e.salary+e.bonus) from EmpBean e where e.dept.deptno = ?1
```

ORDER BY clause

The ORDER BY clause specifies an ordering of the objects in the result collection:

```
ORDER BY  [ order_element ,]* order_element
order_element ::= { path-expression | integer } [ ASC | DESC ]
```

The path expression must specify a single valued field that is a primitive type of byte, short, int, long, float, double, char or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp.

ASC specifies ascending order and is the default. DESC specifies descending order.

Integer refers to a selection expression in the SELECT clause.

Example: ORDER BY clause

Return department objects in decreasing deptno order:

```
SELECT  OBJECT(d) FROM  DeptBean d ORDER BY  d.deptno DESC
```

Return employee objects sorted by department number and name:

```
SELECT  OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC,  e.name DESC
```

The following is a valid dynamic query:

```
SELECT OBJECT(e), e.salary+e.bonus as total_pay FROM  EmpBean e ORDER BY  2 DESC
```

Subqueries

A subquery can be used in quantified predicates, EXISTS predicate or IN predicate. A subquery should only specify a single element in the SELECT clause. When a path expression appears in a subquery, the identification variable of the path expression must be defined either in the subquery, in one of the containing subqueries, or in the outer query. A scalar subquery is a subquery that returns one value. A scalar subquery can be used in a basic predicate and in the SELECT clause of a dynamic query.

Example: Subqueries

```
SELECT  OBJECT(e) FROM  EmpBean e
WHERE e.salary > ( SELECT  AVG(e1.salary) FROM  EmpBean e1)
```

The above query returns employees who earn more than average salary of all employees.

```
SELECT  OBJECT(e) FROM EmpBean e WHERE  e.salary >
( SELECT AVG(e1.salary) FROM  IN  (e.dept.emps) e1  )
```

The above query returns employees who earn more than average salary of their department.

```
SELECT  OBJECT(e) FROM EmpBean e WHERE e.salary =
( SELECT MAX(e1.salary) FROM IN (e.dept.emps) e1  )
```

The above query returns employees who earn the most in their department.

```
 SELECT OBJECT(e) FROM EmpBean e
WHERE e.salary > ( SELECT AVG(e.salary) FROM EmpBean e1
WHERE YEAR(e1.hireDate) =  YEAR(e.hireDate)  )
```

The above query returns employees who earn more than the average of employees hired in same year.

EJB query restrictions

An EJB query is compiled into an SQL query and executed against the underlying datastore based on schema mapping of the abstract bean to the datastore schema. The semantics of comparison and arithmetic operations are that of the underlying datastore. In the case of SQL, note that two strings are equal if the shorter string padded with blanks equals the longer string. For example, `'A'` is equal to `'A '`. This differs from the equality of strings in the Java language. Arithmetic overflow operations are an error in SQL.

A cmp field can not be used in comparison operations or predicates (except for LIKE) if that cmp field is mapped to a long varchar or large objects (LOB) column or any other column type for which the database server does not support predicates or comparison operations.

A cmp field of any type can be used in a SELECT clause. Fields that can be used in predicates, grouping, or ordering operations must be of the types listed below:

- Primitive types : byte, short, int, long, float, double, boolean, char
- Object types: Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Char, java.util.Calendar , java.util.Date
- JDBC types: java.sql.Date, java.sql.Time, java.sql.Timestamp

The field must be mapped to a table column that is compatible in type either by using a ″top-down″ default mapping generated by the WebSphere deploy tool, or using a ″meet-in-the-middle″ mapping between compatible types.

In order to search on attributes of a cmp field that is a user-defined value object, you should use a ″meet-in-the-middle″ mapping and use a composer to map each attribute to a compatible column. The default ″top-down″ mapping stores the object as a serialized object in a column of type blob, which does not allow searching.

If a cmp field is mapped to a column using a ″meet-in-the-middle″ mapping with a converter, that field can only be used with the NULL predicate or with basic predicates of the following form:

```
 path-expression  <comparison>  literal_value
 path-expression  <comparison>  input_parameter
```

In this situation, the converter method toData( ) is called to convert the right-hand side of the predicate to an SQL value.

Example of allowable predicate on a cmp field with user defined converter:

```
e.name = 'Chris'
e.name > ?1
e.name IS NULL
```

Examples of unallowable predicates:

```
substring( e.name, 1, 3 ) = 'ABC'
e.salary >  d.budget
```

A converter should preserve equality, collating sequence and null values when doing a conversion. Otherwise cmp fields created by the converter should not be used in WHERE, GROUP, HAVING or ORDER clauses of a query.

EJB Query: Reserved words

The following words are reserved in WebSphere EJB query:

all, as, distinct, empty, false, from, group, having, in, is, like, select, true, union, where

Avoid using identifiers that start with underscore (for example, _integer) as these are also reserved.

EJB query: BNF syntax

```
EJB QL ::= [select_clause] from_clause [where_clause]
          [group_by_clause] [having_clause] [order_by_clause]

from_clause::=FROM identification_variable_declaration
              [,identification_variable_declaration]*

identification_variable_declaration::=collection_member_declaration |
     range_variable_declaration

collection_member_declaration::=
     IN ( collection_valued_path_expression ) [AS] identifier

range_variable_declaration::=abstract_schema_name [AS] identifier

single_valued_path_expression ::=
      {single_valued_navigation | identification_variable}. ( cmp_field |
             method  |  cmp_field.value_object_attribute |  cmp_field.value_object_method )
       | single_valued_navigation

single_valued_navigation::=
       identification_variable.[  single_valued_cmr_field.  ]*
            single_valued_cmr_field

collection_valued_path_expression ::=
       identification_variable.[  single_valued_cmr_field.  ]*
           collection_valued_cmr_field

select_clause::= SELECT { ALL | DISTINCT }  {single_valued_path_expression |
                         identification_variable |  OBJECT ( identification_variable) }

select_clause_eex ::= SELECT { ALL | DISTINCT }  [ selection , ]*  selection

selection  ::= { expression [[AS] id ]  |  subselect }

order_by_clause::= ORDER BY [ {single_valued_path_expression | integer}  [ASC|DESC],]*
      {single_valued_path_expression | integer}[ASC|DESC]

where_clause::= WHERE conditional_expression

conditional_expression ::= conditional_term |
                           conditional_expression OR conditional_term

conditional_term ::= conditional_factor |
                     conditional_term AND conditional_factor

conditional_factor ::= [NOT] conditional_primary

conditional_primary::=simple_cond_expression | (conditional_expression)

simple_cond_expression ::= comparison_expression | between_expression |
       like_expression | in_expression | null_comparison_expression |
       empty_collection_comparison_expression | quantified_expression |
       exists_expression | is_of_type_expression  |  collection_member_expression

between_expression ::= expression [NOT] BETWEEN expression AND expression

in_expression ::= single_valued_path_expression [NOT] IN
          { (subselect) |  ( atom ,]* atom) }

atom = { string-literal | numeric-constant | input-parameter }

like_expression ::= expression [NOT] LIKE
                {string_literal | input_parameter}
                [ESCAPE {string_literal | input_parameter}]

null_comparison_expression ::=
       single_valued_path_expression IS [ NOT ] NULL

empty_collection_comparison_expression ::=
       collection_valued_path_expression IS [NOT] EMPTY

collection_member_expression  ::=
       { single_valued_path_expression |  input_paramter }  [ NOT ] MEMBER [ OF ]
         collection_valued_path_expression
```

```
quantified_expression ::=
       expression comparison_operator  {SOME | ANY | ALL} (subselect)
exists_expression ::= EXISTS {collection_valued_path_expression |  (subselect)}
subselect ::= SELECT [{ ALL | DISTINCT }]  expression  from_clause [where_clause]
       [group_by_clause] [having_clause]
group_by_clause::= GROUP BY [single_valued_path_expression,]*
                   single_valued_path_expression
having_clause ::= HAVING conditional_expression
is_of_type_expression ::= identifier  IS OF TYPE
            ([[ONLY] abstract_schema_name,]* [ONLY] abstract_schema_name)
comparison_expression ::=  expression   comparison_operator { expression |  ( subquery ) }
comparison_operator ::=    = | > | >= | < | <= | <>
method ::=  method_name( [[expression , ]* expression ] )
expression ::= term |   expression {+|-} term
term ::=  factor |  term {*|/} factor
factor ::= {+|-} primary
primary ::= single_valued_path_expression | literal |
       ( expression ) |  input_parameter | functions
```

<pre/> functions ::= ABS(expression) | AVG([ALL|DISTINCT] expression) | BIGINT(expression) | CHAR({expression [,{ISO|USA|EUR|JIS}]) | CONCAT (expression , expression) | COUNT({[ALL|DISTINCT] expression | *}) | DATE(expression) | DAY({expression) | DAYS(expression) | DECIMAL(expression [,integer[,integer]]) DIGITS(expression) | DOUBLE(expression) | FLOAT(expression) | HOUR (expression) | INTEGER(expression) | LCASE (expression) | LENGTH(expression) | LOCATE(expression, expression [, expression]) | MAX([ALL|DISTINCT] expression) | MICROSECOND(expression) | MIN([ALL|DISTINCT] expression) | MINUTE (expression) | MONTH(expression) | REAL(expression) | SECOND(expression) | SMALLINT(expression) | SQRT (expression) | SUBSTRING(expression, expression[, expression]) | SUM([ALL|DISTINCT] expression) | TIME(expression) | TIMESTAMP(expression) | UCASE (expression) | YEAR(expression)

Comparison of EJB 2.0 specification and WebSphere query language

Item	EJB 2.0 specification	WebSphere Query	WebSphere Enterprise (Dynamic) Query
Bean methods	no	no	yes
Calendar comparisons	yes	yes	yes
Delimited identifiers	no	yes	yes
Dependent Value attributes	no	yes	yes
Dependent Value methods	no	no	yes
Dynamic Query APIs	no	no	yes
EXISTS predicate	no	yes	yes
Inheritance	no	yes	yes
Multiple element select clauses	no	no	yes
Order by	no	yes	yes
Scalar functions	yes *	yes	yes
Select clause	required	optional	optional

Item	EJB 2.0 specification	WebSphere Query	WebSphere Enterprise (Dynamic) Query
SQL Date/time expressions	no	yes	yes
String comparisons	= and <> only	= <> > <	= <> > <
Subqueries, aggregations, group by, and having clauses	no	yes	yes

* EJB 2.0 defines the following scalar functions: abs, sqrt, concat, length, locate and substring. WebSphere query and dynamic query support additional scalar functions as listed in the topic, "EJB query: Scalar functions" on page 63.

Using the dynamic query service

Before you begin

Consider using the dynamic query service (available with IBM WebSphere Application Server Enterprise) when any of the following are true:
- You do not know the query search criteria until application runtime.
- You need to return multiple cmp or cmr fields from a query (deployment queries allow only a single element to be specified in the SELECT clause).
- You want to perform aggregation in the query (deployment queries do not allow use of aggregation function SUM, AVG, COUNT, MAX, MIN in the top level SELECT of a query).
- You want to use value object methods or bean methods in the query statement.
- You want to interactively test an EJB query during development but do not want to repeatedly deploy your application each time you update a finder or select query.

If you have a query that has a high frequency of execution you should define it as a finder or select method and consider using SQLJ as a deployment option for best performance. The dynamic query service always uses JDBC and must parse and process the EJB query at runtime.

If you need security control over which queries a user can execute, you need to define the queries as finder or select methods and use EJB method authorization. The dynamic query service does not have fine grain security control at this time. You can control who is permitted access to the remote query bean and the local query bean, but once authorized a user can execute any valid query and return any data in the server.

The dynamic query API is a stateless session bean. Using the dynamic query API is similar to using any other J2EE EJB application bean.

The default JNDI name for the query bean is as follows, but your system administrator can change this name:

```
com/ibm/websphere/ejbquery/Query
```

The system administrator might need to install the query.ear application into the application server. The WebSphere product install does this only for the default server.

The query bean has both a remote and a local interface to support both remote and local clients.

- remote interface = `com.ibm.websphere.ejbquery.Query`
- remote home interface = `com.ibm.websphere.ejbquery.QueryHome`
- local interface = `com.ibm.websphere.ejbquery.QueryLocal`
- local home interface = `com.ibm.websphere.ejbquery.QueryLocalHome`

To use the local interface of the query bean, you must configure your server to use the following:

```
Application Classloader Policy = SINGLE
```

Using a value of MULTI may result in your application being unable to find the local interface for the query bean home.

The following Jar files comprise the query service:

Jar	Location	Usage
query.jar	server - AppServer/lib	query parser
qjcup.jar	server - AppServer/lib	auxillary classes for parser
querybean.jar	server - installedApps	query session bean
qryclient.jar	server and client	client stubs and classes
querymd.jar	server - AppServer/lib	auxillary classes for query
queryws.jar	server - AppServer/lib	auxillary classes for query

Steps for this task

1. To execute a query, have your client do a JNDI lookup for the QueryHome and create an instance of the query bean. The query bean contains the executeQuery() method, which takes as parameters the query statement in the form of a string, and input parameters in the form of an array of java.lang.Object values. Remote clients also pass as arguments the size of the result set to return.

   The results of the query are returned for remote clients as:

   ```
   com.ibm.websphere.ejbquery.QueryIterator
   ```

   or for local clients as:

   ```
   com.ibm.websphere.ejbquery.QueryLocalIterator
   ```

   If you want to return remote EJB references from the query, or if the query statement contains remote methods, you must use the query remote interface.

   If you want to return local EJB references from the query, or if the query statement contains local methods, you must use the query local interface.

   Calling the next() method on the iterator returns an intance of com.ibm.websphere.ejbquery.IQueryTuple, which contains the actual data values or object references. The iterator also contains the following methods:

```
getFieldName( int i)
getFieldsCount( );
```

2. Compile and run your client program with the file **qryclient.jar** in the classpath.

   For more details, see the Javadoc for package com.ibm.websphere.ejbquery.

   Security Considerations. WebSphere does not have security access control for CMP and CMR fields. You must therefore secure the query bean methods executeQuery(), prepareQuery(), and executePlan(), and the create() method on the query home; otherwise, any user is able to perform a dynamic query and retrieve data from your application.

Example: Dynamic query remote client

```
  import com.ibm.websphere.ejbquery.QueryHome;
  import com.ibm.websphere.ejbquery.Query;
  import com.ibm.websphere.ejbquery.QueryIterator;
  import com.ibm.websphere.ejbquery.IQueryTuple;
  import com.ibm.websphere.ejbquery.QueryException;

try {
 String query =
        "select e.name, object(e) from EmpBean e where e.salary < ?1 ";

 InitialContext ic =  ...  // get initial context

 // the following jndi name may have to be changed  to match the
 // jndi name assigned to query session bean home.
 Object obj =  ic.lookup("com/ibm/websphere/ejbquery/Query" );

 QueryHome  qh =
     ( QueryHome) javax.rmi.PortableRemoteObject.narrow( obj, QueryHome.class );
 Query qb = qh.create();

 // pass in a query parameter of 10000

 Object[] parms = new Object[] { new Integer(10000) } ;

 // return first 99 result tuples
 QueryIterator it = qb.executeQuery(query, parms, null ,0, 99 );

 qb.remove();

 // display the query result
 // each tuple contains the values for e.name and object(e) because
 // the query was "select e.name, object(e) from ... "
 //
 // the number of fields and their names can be obtained by calling
 //    getFieldsCount()
 //    getFieldName( int i)

 while (it.hasNext() ) {
    IQueryTuple t  = (IQueryTuple) it.next();
    System.out.print( it.getFieldName(1) +"="+ t.getObject(1) );
    System.out.println( it.getFieldName(2) +"="+
               t.getObject(2).getPrimaryKey().toString() );
 }


} catch (QueryException qe) {
        System.out.println("Query Exception "+ qe.getMessage() );
  }
```

Example: Dynamic query from local client

```
import com.ibm.websphere.ejbquery.QueryLocalHome;
  import com.ibm.websphere.ejbquery.QueryLocal;
  import com.ibm.websphere.ejbquery.QueryLocalIterator;
  import com.ibm.websphere.ejbquery.IQueryTuple;
  import com.ibm.websphere.ejbquery.QueryException;

try {
 String query =
       "select e.name, object(e) from EmpBean e where e.salary < ?1 ";

 InitialContext ic =  new InitialContext();
 // replace "query" with the ejb-local-ref defined for your appl
 // in the following line
 Object obj =  ic.lookup( "java:comp/ejb/query" );

 QueryLocalHome  qh =  ( LocalQueryHome) obj;
 QueryLocal qb = qh.create();

 // pass in a query parameter of 10000

 Object[] parms = new Object[] { new Integer(10000) } ;

 // the local query iterator requires a transaction context

 userTransaction.begin();

 QueryLocalIterator it = qb.executeQuery(query, parms, null);

 qb.remove();

 // display the query result
 // each tuple contains the values for e.name and object(e) because
 // the query was "select e.name, object(e) from ... "
 //
 // the number of fields and their names can be obtained by calling
 //    getFieldsCount()
 //    getFieldName( int i)

 while (it.hasNext() ) {
    IQueryTuple t  = (IQueryTuple) it.next();
    System.out.print( it.getFieldName(1) +"="+ t.getObject(1) );
    System.out.println( it.getFieldName(2) +"="+
               t.getObject(2).getPrimaryKey().toString() );
 }

 userTransaction.commit();

} catch (QueryException qe) {
        System.out.println("Query Exception "+ qe.getMessage() );
 }
```

Chapter 4. Using the internationalization service

Before you begin

The internationalization service adds APIs and tooling that enable J2EE applications to manage the distribution of internationalization information, or *internationalization context*, necessary to perform localizations within server-side application components. This topic summarizes the steps involved in using the internationalization service.

Steps for this task

1. If you have an application that uses the WebSphere Version 4.0 internationalization service, review the topic "Migrating internationalized applications" on page 81.
2. "Assembling internationalized applications" on page 82.

   Use the Application Assembly Tool to configure the internationalization type and any container internationalization attributes for the servlets and enterprise beans of your application.

   Internationalization type specifies the internationalization policy applicable to a servlet or an enterprise bean and, in particular, indicates whether the application component or its hosting J2EE container will manage internationalization context. Container internationalization attributes can be specified for container-managed servlet and enterprise bean business methods. These attributes tailor a policy by indicating which context the container will scope to an invocation. Configuring internationalization policies declaratively prescribes, by means of the application's deployment descriptor, the distribution and management of context throughout an application.
3. "Using the internationalization context API" on page 90.

   Use the internationalization context API within application components to obtain or manage internationalization context.

   Servlet and enterprise bean business methods can use internationalization context to perform locale- and time zone-sensitive localizations. EJB client applications, and server components configured to manage internationalization context, must use the internationalization context API to set the context elements scoped to their invocations.
4. "Managing the internationalization service" on page 112.

   Use the administrative console to enable the service on all application servers.

   By default, the service is enabled within the J2EE client environment, but is disabled on application servers. You must enable the service on all application servers hosting your application's servlets and enterprise beans in order to use internationalization context.
5. "Troubleshooting the internationalization service" on page 114.

   Use the administrative console to enable the trace service to log internationalization service messages when debugging your applications.

Internationalization

An application that can present information to users according to regional cultural conventions is said to be *internationalized*: The application can be configured to interact with users from different localities in culturally appropriate ways. In an internationalized application, a user in one region sees error messages, output, and interface elements in the requested language. Date and time formats, as well as currencies, are presented appropriately for users in the specified region. A user in another region sees output in the conventional language or format for that region.

Historically, the creation of internationalized applications has been restricted to large corporations writing complex systems. Internationalization techniques have traditionally been expensive and difficult to implement, so they have been applied only to major development efforts. However, given the rise in distributed computing and in the use of the World Wide Web, application developers have been pressured to internationalize a much wider variety of applications. This requires making internationalization techniques much more accessible to application developers.

In an application that is not internationalized, the interface that the user sees is unalterably written into the application code. On the other hand, localizing the displayed strings adds a layer of abstraction into the design of the application. Instead of simply printing an error message, an internationalized application represents the error message with some language-neutral information; in the simplest case, each error condition corresponds to a key. To print a usable error message, then, the application looks up the key in the configured message catalog. Each message catalog is a list of keys with associated strings. Different message catalogs provide strings for the different languages supported. The application looks up the key in the appropriate catalog, retrieves the corresponding error message in the requested language, and prints this string for the user.

Localization of text can be used for far more than translating error messages. For example, by using keys to represent each element in a graphical user interface (GUI) and by providing the appropriate message catalogs, the GUI itself (buttons, menus, and so on) can support multiple languages. Extending support to additional languages requires that you provide message catalogs for those languages; in many cases, the application itself needs no further modification.

Internationalization of an application is driven by two variables, the time zone and the locale. The time zone indicates how to compute the local time as an offset from a standard time like Greenwich Mean Time. The locale is a collection of information about language, currency, and the conventions for presenting information like dates. In a localized application, the locale also indicates the message catalog from which an application is to retrieve message strings. A time zone can cover many locales, and a single locale can span time zones. With both time zone and locale, the date, time, currency, and language for users in a specific region can be determined.

IBM WebSphere Application Server Enterprise provides an Internationalization Service, which manages the distribution of locale and time-zone information, or *internationalization context*, within Java applications.

Internationalization service: Overview

In a distributed client-server environment, application processes can run on different machines, configured to different locales, corresponding to different cultural conventions; they can also be located across geographical boundaries. For an understanding of how these differences impact application development, review the topic “Internationalization challenges in distributed applications” on page 80.

The J2EE platform provides support for application components executing on computers with differing endian architecture and code sets, but does not provide dedicated support for application components that run on computers having different locales or time zones.

The conventional method for solving locale and time zone mismatch across remote application components is to pass one or more extra parameters on all business methods needed to convey the client-side locale or time zone to the server. Although simple, this technique has the following limitations when used in EJB applications:

- It is intrusive because it requires that one or more parameters be added to all bean methods in the call chain to locale-sensitive or time zone-sensitive methods.
- It is inherently error-prone.
- It is impracticable within applications that do not afford modification, such as legacy applications.

The internationalization service solution

The WebSphere internationalization service addresses the challenges posed by locale and time zone mismatch without incurring the limitations of conventional techniques. It does this by systematically managing the distribution of internationalization contexts across the various components of EJB applications, including client applications, enterprise beans, and servlets.

The service works by associating an “Internationalization context” on page 101 with every thread of execution within an application. When a client-side component invokes a business method, the internationalization service interposes by obtaining the internationalization context associated with the current thread of the client-side process and attaching that context to the outgoing request. On the server-side, the internationalization service again interposes by detaching the context from the incoming request and associating it with the thread of the server-side process on which the business method will execute, effectively scoping the context to the business method. The service propagates internationalization context on subsequent business method invocations in the same manner and thus distributes the context of the originating request over the entire chain of business method invocations.

This basic operation of “Internationalization context: Propagation and scope” on page 102 is defined precisely by “Internationalization context: Management policies” on page 104. Every application component has a default policy, which can be overridden and tailored for servlets and enterprise beans at development time using WebSphere’s Application Assembly Tool (AAT). Internationalization policies specify whether an application component or its hosting J2EE container are to manage internationalization context. For container-managed components, the policy indicates which internationalization context the container will scope to invocations on that component. Server components configured to manage internationalization context, as well as EJB clients, must use the internationalization context API to manage the internationalization context elements scoped to their invocations.

At execution time, application components can use the "Internationalization context API: Programming reference" on page 99 to get any element of the internationalization contexts scoped to an invocation. To programmatically access context elements, application components first resolve an internationalization context API reference, then invoke the appropriate API method to access the various context elements, such as the caller locale or the invocation time zone. These elements can be used in calls to Java 2 SDK internationalization API methods; for example, to perform localizations such as formatting messages, configuring dates, or comparing strings.

Internationalization challenges in distributed applications

With the advent of Internet-based business computational models, such as eCommerce, applications increasingly comprise clients and servers that operate in different locales and geographical regions. These differences introduce challenges to the task of designing a sound client-server infrastructure. Specifically, clients and servers can:

- Execute on computers having different endian architectures or code sets.
- Be located in different locales.
- Be located in different time zones.

The following sections describe these three challenges in more detail.

Computers with differing endian architectures or code sets

Clients and servers can reside in computers having different endian architectures: a client could reside in a little-endian CPU, while the server code runs in a big-endian one. As a more complex instance, a client might want to invoke a business method on a server running in a code set different from that of the client.

A client-server infrastructure must define precise endian and code set tracking and conversion rules. Both CORBA and J2EE have addressed the problems of endian and code set mismatches. The language-neutral CORBA formalism uses byte order indicator in all marshalled data streams to indicate the byte order of the originating machine; in case of an endian mismatch, the receiving side can perform byte swapping for endian correction. The code set mismatch is addressed by CORBA using a comprehensive framework for code set conversion.

J2EE has nearly eliminated these problems in a unique way by relying on its Java Virtual Machine (JVM), which encodes all string data in UCS-2 format and externalizes everything in big-endian format. The JVM employs a set of platform-specific programs for interfacing with the native platform. These programs perform any necessary code set conversions between UCS-2 and the native code set of a platform.

Computers located in different locales

Client and server processes can execute in geographical locations having different locales. For example, a Spanish client might invoke a business method upon an object residing on an American server. Some business methods can be locale-sensitive in nature; for example, given a business method that returns a sorted list of strings, the Spanish client expects that list to be sorted according to the Spanish collating sequence, not in the server's English collating sequence. Since data retrieval and sorting procedures run on the server, the locale of the client must be available in order to perform a legitimate sort.

A similar consideration applies in instances where the server has to return strings containing date, time, currency, exception messages, and so on, formatted according to the client's cultural expectations. Neither the CORBA nor the J2EE specifications have architecturally addressed the locale mismatch problem and other options involving extra parameters are not practical or have limitations. For example, requiring an extra parameter could require interface changes, which is a serious concern for deployed applications.

Computers located in different time zones

Client and server processes can execute in geographical locations having different time zones. To date, all internationalization literature and resources have concentrated mainly on code set and locale-related issues. They have generally ignored the time zone issue, even though business methods can be sensitive to time zone as well as to locale.

For example, suppose that a vendor makes the claim that ″orders received before 2:00 PM will be processed by 5:00 PM the same day″. The times given, of course, are in the time zone of the server that is processing the order. It is important to know the client's time zone in order to give customers in other time zones the correct times for same-day processing.

Other time zone-sensitive operations include time stamping messages logged to a server, and accessing file or database resources. The concept of Daylight Savings Time (DST) further complicates the time zone issue. Neither the CORBA nor the J2EE specifications address time zone issues adequately, and conventional methods of solving this problem are limited.

Migrating internationalized applications

Applications that used the internationalization service in IBM WebSphere Application Server, Version 4.0 can use the service in Version 5.0 with no modification. The packaging and structure of the internationalization context API remain identical across releases. Most importantly, the semantics of the API remain as well.

In Version 4.0, the internationalization service did not provide internationalization deployment descriptor policy information to direct how the service manages internationalization context across the various application components. Rather, the service employed the implicit client-side internationalization (CSI) and server-side internationalization (SSI) policies, which dictated how the service managed context according to the type of J2EE container hosting a component. For details, refer to the WebSphere Application Server Version 4.0 Integrated InfoCenter. Briefly, all server components in Version 4.0 were SSI, and all EJB client applications were CSI.

In Version 5.0, the internationalization type setting of all server components is configured to ″Container″ by default. The internationalization service assigns the default container internationalization attribute, ″RunAsCaller″, to any container-managed (CMI) servlet or enterprise bean invocation lacking a container internationalization attribute. Hence, invocations of server components lacking internationalization policy information in the deployment descriptor run under the policy, `[CMI, RunAsCaller]`, which is semantically equivalent to the SSI internationalization policy of Version 4.0; EJB client applications run under the logical policy `[AMI, RunAsServer]`, which is equivalent to the CSI policy of Version 4.0.

When migrating a Version 4.0 application to Version 5.0, it is unnecessary to configure the internationalization deployment descriptor information during application assembly because all component invocations execute under semantically equivalent internationalization context management policies.

Assembling internationalized applications

Use the Application Assembly Tool to configure internationalization deployment descriptor information for servlets and enterprise beans.

Steps for this task

1. (Optional) Set the **internationalization type**.

   All servlets and enterprise beans have an internationalization type setting that specifies whether internationalization context is managed by the application component or by its hosting J2EE container during invocations of their respective lifecycle and business methods. Internationalization type can be configured for all server application components except entity beans, which are container-managed only.

   By default, all server components employ container-managed internationalization (CMI). The default setting should suffice in most cases; when it does not, modify the internationalization type setting by completing the steps described in one of the following topics:

   - "Setting the internationalization type for servlets"
   - "Setting internationalization type for enterprise beans" on page 86
2. (Optional) Set the **container internationalization attribute**.

   You can associate CMI servlets, and business methods of CMI enterprise beans, with a container internationalization attribute that specifies which of three internationalization contexts - **Caller**, **Server**, or **Specified** - the container is to scope to an invocation. When running as specified, the container internationalization attribute also specifies the custom internationalization context elements.

   Named container internationalization attributes can be associated with sets of servlets or with sets of EJB business methods. Initially, CMI servlets and business methods implicitly run as caller and do not associate with a container internationalization attribute. When the implicit behavior or an associated attribute setting is unsuitable, configure an attribute by completing the steps described in one of the following topics:

   - "Configuring container internationalization for servlets" on page 83
   - "Configuring container internationalization for enterprise beans" on page 87

Setting the internationalization type for servlets

This task sets the internationalization type for a servlet within a Web module.

Steps for this task

1. Start the Application Assembly Tool.
2. View the servlets of your Web module by selecting *application_name* > **Web Modules** > *webmodule_name* > **Web Components** in the left-hand pane.

3. Select your servlet from the left-hand pane.
4. On the **WAS Enterprise** tab on the right, select either **Container** or **Application** from the **Internationalization type** drop-down menu.
5. Click **Apply**.

Results

The internationalization type setting is assigned to the servlet.

Configuring container internationalization for servlets

This task configures container internationalization for a servlet within a Web module. Use this procedure to relate one or more servlets to a container internationalization attribute.

Steps for this task

1. Start the Application Assembly Tool.
2. In the left-hand panel, expand your Web module.
3. Select **Internationalization**.
4. To configure a new attribute, right-click on **Internationalization** and select **New**; otherwise, skip to step 5.
   a. On the **New Container Internationalization Attribute** panel, enter a **Description** that uniquely identifies the policy.
   b. Click **Add**. From the **Add Servlets** list, select one or more servlets to which the attribute will apply and click **OK** to exit the panel. The servlet(s) appear in the Web Components list.
   c. Click **OK**.

   The new attribute description is displayed in the **Description** list in the right-hand panel.
5. Select a named **Container Internationalization Attribute** from the Description list.

   The fields of the selected attribute are displayed.
6. If desired, re-enter a description (name) that uniquely identifies this attribute.

   A description is an arbitrary character string.
7. Click the **Add** button adjacent to the **Web Components** list.
8. From the **Add Servlets** list, select one or more servlets to which the attribute applies.
9. Click **OK**.

   The servlets appear in the Web Components list. Repeat these steps for any remaining servlets you want configure.
10. Complete the **Run as** field by selecting **Caller**, **Server**, or **Specified**.

    If the servlet is to run as **Specified**, complete the following steps to specify the context elements that the container will scope to servlet invocations; otherwise, click **OK** to exit this panel.
11. (Optional) Enter the **Description** of the specified localization context.

    A description is an arbitrary character string.
12. Complete the **Time Zone ID** fields:
    a. (Optional) Enter the time zone ID **Description**.

       A description is an arbitrary character string.

   b. Enter the **Time Zone ID**.

      A time zone ID is an arbitrary, non-empty string that identifies a time zone supported by the Java SDK type, java.util.SimpleTimeZone. Refer to the topic Container internationalization attributes for details.

13. Complete the **Locales** fields; perform the following steps to construct an ordered list of locales:
    a. Click **Add**.

       The **Add Locales** panel is displayed.
    b. [Optional] Enter the locale **Description**.

       A description is an arbitrary character string.
    c. Enter the **Language Code**.

       A language code is an arbitrary string. A valid locale must contain at least a language code or a country code.
    d. Enter the **Country Code**.

       A country code is an arbitrary string. A valid locale must contain at least a language code or a country code.

    Refer to the section on locales in the topic "Container internationalization attributes" on page 107 for details about language codes, country codes, and variants.
14. Click **Apply** to apply your changes and exit this panel.

Results

The servlets are now configured to run under the associated internationalization settings.

Internationalization assembly properties for Web modules

Use this page to specify which internationalization context the Web container will scope to servlet service method invocations.

By default, a Web container scopes the caller's internationalization context to service method invocations of servlets and Java Server Pages (JSPs) configured to the **Container** internationalization type. To override this default scoping behavior, create and configure at least one Container Internationalization attribute.

A Container Internationalization attribute associates those servlets (JSPs) that employ container-managed internationalization to one of three internationalization contexts: the **Caller** context, the **Server** context, or the context **Specified** in the attribute. For each servlet listed in the attribute, the container will scope the internationalization context indicated by the attribute's **Run as** field.

Use the information below to configure new and existing Container Internationalization attributes.

Description: A description that identifies the attribute.

Data type
String

Servlet: The set of servlets to which the **Run as** field applies.

Click the **Add** button to select the servlets specified by this attribute.

Run as: The Run as field specifies which invocation context the container will scope to every lifecycle method invocation of the servlet indicated in the Servlet field.

Default
Caller

Range

Caller

The container invokes the attribute's servlet with the locales (the accept-languages) of the incoming HTTP request. The container supplies GMT+00:00 for time zone. Select Caller when you want servlets to execute under the invocation context of the client request.

Server

The container invokes the servlet with the default context elements of the server JVM. Select Server when you want servlets to execute under the invocation context of the server JVM.

Specified

The container invokes the attribute's servlet with the context elements specified in the attribute's **Locales** and **Time zone ID** fields. Select Specified when you want servlets to execute under the invocation context elements specified in the attribute. Be sure to complete the **Locales** and **Time zone ID** fields.

Description
A description of the group of specified context elements.

Data type
String

Time zone: A time zone represents a temporal offset and computes daylight savings information.

Specify the time zone fields within the **Time Zone** panel according to the instructions below.

Description
A description of the specified time zone (ID).

Data type
String

ID A short-hand identifier for a time zone.

Enter a valid time zone ID. A valid ID represents any time zone supported by the SDK type, java.util.TimeZone. Specifically, an ID may be any of the IDs appearing in the list of time zone IDs returned by method java.util.TimeZone.getAvailableIds(), or a custom ID having the form GMT[+|-]hh[[:]mm]; for example, ″America/Los_Angeles″, ″GMT-08:00″ are valid time zone IDs.

Data type
String

Locales: An ordered list of locales, where a locale represents a specific geographical, cultural, or political region.

Description
A description of the specified locale.

Data type
String

Language code
A code identifying the language spoken within a particular region.

Enter the language code of the new locale. Ideally, language code is one of the lower-case, two-character codes defined by ISO-639. Language code is not restricted to ISO codes and is not a required field; however, a valid locale must contain either a language code or a country code.

Data type
String

Country code
A code identifying the country within a particular region.

Enter the country code of the new locale. Ideally, country code is one of the upper-case, two-character codes defined by ISO-3166. Country code is not restricted to ISO codes and is not a required field; however, a valid locale must specify either a language code or a country code.

Data type
String

Variant
A vendor-specific code.

Enter the variant of the new locale. Variant is not a required field and serves only to supplement the **Language code** and **Country code** fields according to application- or platform-specific requirements.

Data type
String

Setting internationalization type for enterprise beans

This task sets the internationalization type for an enterprise bean within an EJB module.

Steps for this task

1. Start the Application Assembly Tool.
2. View the beans of your EJB module by selecting *application_name* > **EJB Modules** > *EJBmodule_name* > **Session Beans | Message-Driven Beans** in the left-hand panel.

   Recall that internationalization type cannot be configured on entity beans, which are CMI only.
3. Select your bean from the left-hand panel.
4. On the **WAS Enterprise** tab of the right-hand panel, select either **Container** or **Application** from the **Internationalization type** drop-down menu.
5. Click **Apply**.

Results

The internationalization type is assigned to the bean.

Configuring container internationalization for enterprise beans

This task configures container internationalization for enterprise bean business methods. Use this procedure to relate one or more business methods to a container internationalization attribute.

Steps for this task

1. Start the Application Assembly Tool.
2. In the left-hand panel, expand your EJB module.
3. Select **Internationalization**.
4. To configure a new attribute, right-click on **Internationalization** and select **New**; otherwise, skip to step 5.
   a. On the **New Container Internationalization Attribute** panel, enter a **Description** that uniquely identifies the policy.
   b. Click **Add**. From the **Add Methods** list, select one or more methods to which the attribute will apply and click **OK** to exit the panel. The selections appear in the Methods list.
   c. Click **OK**.

   The new attribute description is displayed in the **Description** list.
5. Select a named **Container Internationalization Attribute** from the **Description** list.

   The fields of the selected attribute are displayed.
6. If desired, re-enter a description (name) that uniquely identifies this attribute.

   A description is an arbitrary character string.
7. To specify the list of bean methods to which the attribute applies, click the **Add** button adjacent to the **Methods** list.

   The **Add Methods** panel is displayed.
8. Select a bean method to which the attribute applies.
9. Click **Apply**.

   The method appears in the methods list. Repeat these steps for each remaining bean method that you want configure.
10. Click **OK** to exit the Add Methods panel.
11. Complete the **Run As** field by selecting **Caller**, **Server**, or **Specified**.

    If the bean is to run as **Specified**, complete the following steps to specify the context elements that the container will scope to bean method invocations; otherwise, click **OK** to exit the panel.
12. (Optional) Enter a **Description** of the specified context.

    A description is an arbitrary character string.
13. Complete the **Time Zone ID** fields:
    a. (Optional) Enter the **Time zone ID Description**.

       A description is an arbitrary character string.
    b. Enter the **Time Zone ID**.

       A time zone ID is an arbitrary, non-empty string that identifies a time zone supported by the Java SDK type, java.util.SimpleTimeZone. Refer to the topic "Container internationalization attributes" on page 107 for details.
14. Complete the **Locales** fields. Complete the following steps to construct an ordered list of locales:
    a. Click Add.

The **Add Locales** panel is displayed.

b. (Optional) Enter the locale **Description**.

   A description is an arbitrary character string.

c. Enter the **Language code**.

   A language code is an arbitrary string. A valid locale must contain at least a language code or a country code.

d. Enter the **Country code**.

   A country code is an arbitrary string. A valid locale must contain at least a language code or a country code.

Refer to the topic "Container internationalization attributes" on page 107 for details.

15. Click **OK** to apply your changes and to exit this panel.

Results

The bean methods are now configured to run under the associated internationalization settings.

Internationalization assembly settings for EJB modules

Use this page to specify which internationalization context the EJB container will scope to enterprise bean business method invocations.

By default, the EJB container scopes the caller's internationalization context to business method invocations of enterprise beans configured to the **Container** internationalization type. To override this default scoping behavior, create and configure at least one **Container Internationalization** attribute.

A Container Internationalization attribute associates business methods of those enterprise beans that employ container-managed internationalization to one of three internationalization contexts: the **Caller** context, the **Server** context, or the context **Specified** in the attribute. For each method listed in the attribute, the container will scope the internationalization context indicated by the attribute's **Run as** field.

Use the information below to configure new and existing Container Internationalization attributes.

Description: A description that identifies the attribute.

Data type
String

Methods: The set of EJB methods to which the **Run as** field applies. Click the **Add** button, then select the methods to be specified in the attribute.

Run as: The specific internationalization context that the EJB container will scope to invocations of the business methods indicated in the **Methods** field.

Default
Caller

Range

Caller

The container invokes the attribute’s methods with the locales and time zone of the incoming client request. For any missing context element, the container supplies the corresponding default element of the server JVM. Select Caller when you want bean methods to execute under the invocation context of the client request.

Server

The container invokes the attribute’s methods with the default context elements of the server JVM. Select Server when you want bean methods to execute under the invocation context of the server JVM.

Specified

The container invokes the attribute’s methods with the context elements specified in the attribute’s **Locales** and **Time zone ID** fields. Select Specified when you want bean methods to execute under the invocation context specified in the attribute. Be sure to complete the **Locales** and **Time zone ID** fields.

Description
A description of the group of specified context elements.

Time zone: Represents a temporal offset and computes daylight savings information.

Description
A description of the specified time zone (ID).

Data type
String

ID A short-hand identifier for a time zone.

Enter a valid time zone ID. A valid ID represents any time zone supported by the SDK type java.util.TimeZone. Specifically, an ID can be any of the IDs appearing in the list of time zone IDs returned by method java.util.TimeZone.getAvailableIds(), or a custom ID having the form GMT[+|-]hh[[:]mm]; for example, ″America/Los_Angeles″, ″GMT-08:00″ are valid time zone IDs.

Data type
String

Locales: An ordered list of locales, where a locale represents a specific geographical, cultural, or political region.

Description
A description of the specified locale.

Data type
String

Language code
A code identifying the language spoken within a particular region.

Enter the language code of the new locale. Ideally, language code is one of the lower-case, two-character codes defined by ISO-639. Language code is not restricted to ISO codes and is not a required field; however, a valid locale must contain either a language code or a country code.

Data type
String

Country code

A code identifying the country within a particular region.

Enter the country code of the new locale. Ideally, country code is one of the upper-case, two-character codes defined by ISO-3166. Country code is not restricted to ISO codes and is not a required field; however, a valid locale must specify either a language code or a country code.

Data type

String

Variant

A vendor-specific code.

Enter the variant of the new locale. Variant is not a required field and serves only to supplement the **Language code** and **Country code** fields according to application- or platform-specific requirements.

Data type

String

Using the internationalization context API

Before you begin

EJB client applications, servlets, and enterprise beans can programmatically obtain and manage internationalization context using the internationalization context API

The java.util and com.ibm.websphere.i18n.context packages contain all classes necessary to use the internationalization service within an EJB application. Classes specific to the internationalization service reside in the file *WAS_HOME*/lib/i18nctx.jar. Before compiling application components that import internationalization service classes, add the i18nctx.jar file to your CLASSPATH.

Steps for this task

1. "Gaining access to the internationalization context API"

   Gaining access to the internationalization context API.
2. "Accessing caller locales and time zone" on page 92

   Accessing caller locales and time zone.
3. "Accessing invocation locales and time zone" on page 93

   Accessing invocation locales and time zone.

Usage scenario

Each EJB application component uses the internationalization context API differently. Three code examples are provided that illustrate how to use the API within each application type. Differences in API usage, as well as other coding tips, are noted in comments that precede the relevant statement blocks.

Gaining access to the internationalization context API

This topic describes how to access the internationalization service by resolving a reference to the internationalization context API.

Recommendation: Resolve internationalization context API references once over the lifecycle of an application component, within the initialization method of that component (for example, within the init() method of servlets, or within the SetXxxContext() method of enterprise beans.)

Steps for this task

1. Resolve a reference to the UserInternationalization interface by performing a lookup on the following JNDI name:

   ```
   java:comp/websphere/UserInternationalization
   ```

   For example:

   ```
   //------------------------------------------------------------------
   // Internationalization context imports.
   //------------------------------------------------------------------
   import com.ibm.websphere.i18n.context.*;
   import javax.naming.*;
   ...

   public class MyApplication {
     ...

     //------------------------------------------------------------------
     // Resolve a reference to the UserInternationalization interface.
     //------------------------------------------------------------------
     InitialContext initCtx = null;
     UserInternationalization userI18n = null;
     final String UserI18nUrl = "java:comp/websphere/UserInternationalization";
     try {
       initCtx = new InitialContext();
       userI18n = (UserInternationalization)initCtx.lookup(UserI18nUrl);
     }
     catch (NamingException ne) {
       // UserInternationalization URL is unavailable.
     }
   ```

   If the UserInternationalization object is unavailable due to an anomaly or a restriction, the JNDI lookup invocation throws a javax.naming.NameNotFoundException containing the java.lang.IllegalStateException.
2. Use the UserInternationalization reference to create references to the CallerInternationalization or InvocationInternationalization objects, which provide access to elements of the Caller or Invocation internationalization contexts, respectively.

   The CallerInternationalization reference can be bound to the Internationalization interface, only; the InvocationInternationalization reference can be bound to either the Internationalization or the InvocationInternationalization interfaces, depending on whether the application requires read-only or read-write access to invocation context.

   For example:

   ```
     ...
     //------------------------------------------------------------------
     // Resolve references to the Internationalization and
     // InvocationInternationalization interfaces.
     //------------------------------------------------------------------
     Internationalization callerI18n = null;
     InvocationInternationalization invocationI18n = null;
     try {
       callerI18n = userI18n.getCallerInternationalization();
       invocationI18n = userI18n.getInvocationInternationalization();
   ```

```
  }
  catch (IllegalStateException ise) {
    // An Internationalization interface(s) is unavailable.
  }
```

Accessing caller locales and time zone

Before you begin

To access caller locales and time zone, an application component first resolves a reference to the CallerInternationalization object and binds it to the Internationalization interface. Instructions for resolving this reference can be found in the topic "Gaining access to the internationalization context API" on page 90.

Every remote invocation of an application component has an associated caller internationalization context associated with the thread running that invocation. Caller context is propagated by the internationalization service and middleware to the target of a request, such as an EJB business method or servlet service method.

Steps for this task

1. Obtain the desired caller context elements:

   ```
   java.util.Locale [] myLocales  = null;
   try {
     myLocales  = callerI18n.getLocales();
   }
   catch (IllegalStateException ise) {
     // The Caller context is unavailable;
     // is the service started and enabled?
   }
   ...
   ```

   The Internationalization interface contains the following methods to get caller internationalization context elements:

   - **Locale [] getLocales()** Returns the list of caller locales associated with the current thread.
   - **Locale getLocale()** Returns the first in the list of caller locales associated with the current thread.
   - **TimeZone getTimeZone()** Returns the caller SimpleTimeZone associated with the current thread.

   The Internationalization interface allows read-only access to internationalization context within application components. Methods of the Internationalization interface are available to all EJB application components and are used in the same manner for each, but the method semantics vary according to the component's type. For instance, when obtaining caller locale within an EJB client application, the interface returns the default locale of the host JVM; in contrast, when obtaining caller context within a servlet service method (for example, doPost() or doGet() methods), the interface returns the first locale (accept-language) propagated within the corresponding HTML request. See the topic "Internationalization context" on page 101 for a discussion of how the service propagates internationalization context throughout an application.
2. Use the caller context elements to localize computations under a locale or time zone of the calling process:

   ```
   DateFormat df = DateFormat.getDateInstance(myLocale);
   String localizedDate = df.getDateInstance().format(aDateInstance);
   ...
   ```

Accessing invocation locales and time zone

Before you begin

To access invocation locales and time zone, an application component must first resolve a reference to the InvocationInternationalization object and bind it to the InvocationInternationalization interface of the internationalization context API. Instructions for resolving this reference can be found in the topic "Gaining access to the internationalization context API" on page 90.

Every remote invocation of a servlet service or EJB business method has an invocation internationalization context associated with the thread running that invocation. Invocation context is the internationalization context under which servlet and business method implementations execute; it is propagated on subsequent invocations by the internationalization service and middleware.

Steps for this task

1. Obtain the desired invocation context elements.

   ```
   java.util.Locale myLocale;
   try {
     myLocale = invocationI18n.getLocale();
   }
   catch (IllegalStateException ise) {
     // The invocation context is unavailable;
     // is the service started and enabled?
   }
   ...
   ```

   The InvocationInternationalization interface contains the following methods to both get and set invocation internationalization context elements:

   - **Locale [] getLocales()**. Returns the list of invocation locales associated with the current thread.
   - **Locale getLocale()**. Returns the first in the list of invocation locales associated with the current thread.
   - **TimeZone getTimeZone()**. Returns the invocation SimpleTimeZone associated with the current thread.
   - **setLocales(Locale [])**. Sets the list of invocation locales associated with the current thread to the supplied list.
   - **setLocale(Locale)**. Sets the list of invocation locales associated with the current thread to a list containing the supplied locale.
   - **setTimeZone(TimeZone)**. Sets the invocation time zone associated with the current thread to the supplied SimpleTimeZone.
   - **setTimeZone(String)**. Sets invocation time zone associated with the current thread to a SimpleTimeZone having the supplied ID.

   The InvocationInternalization interface allows read and write access to invocation internationalization context within application components. However, according to internationalization context management policies, only components configured to manage internationalization context (AMI components) have write access to invocation internationalization context elements. Calls to set invocation context elements within CMI application components result in a java.lang.IllegalStateException. Any differences in how application components can use InvocationInternationalization methods are explained in the topic "Internationalization context" on page 101.

2. Use the invocation context elements to localize a computation under a locale or time zone of the invoking process:

```
DateFormat df = DateFormat.getDateInstance(myLocale);
  String localizedDate = df.getDateInstance().format(aDateInstance);
  ...
```

Usage scenario

In the following code example, locale (en,GB) and simple time zone (GMT) transparently propagate on the call to the myBusinessMethod() method. Server-side application components, such as myEjb, can use the InvocationInternationalization interface to obtain these context elements.

```
...
//--------------------------------------------------------------------
// Set the invocation context under which the business method or
// servlet will execute and propagate on subsequent remote business
// method invocations.
//--------------------------------------------------------------------
try {
  invocationI18n.setLocale(new Locale("en", "GB"));
  invocationI18n.setTimeZone(SimpleTimeZone.getTimeZone("GMT"));
}
catch (IllegalStateException ise) {
  // Is the component CMI; is the service started and enabled?
}
myEjb.myBusinessMethod();
```

Within CMI application components, the Internationalization and InvocationInternationalization interfaces are semantically equivalent, and either of these interfaces can be used to obtain the context associated with the thread on which that component is running. For instance, both interfaces can be used to obtain the list of locales propagated to the servlet doPost() service method.

Example: Internationalization context in a contained EJB client

This code example illustrates how to use the internationalization context API within a contained EJB client program.

```
//------------------------------------------
// Basic Example: J2EE EJB client.
//------------------------------------------
package examples.basic;

//------------------------------------------
// INTERNATIONALIZATION SERVICE: Imports.
//------------------------------------------
import com.ibm.websphere.i18n.context.UserInternationalization;
import com.ibm.websphere.i18n.context.Internationalization;
import com.ibm.websphere.i18n.context.InvocationInternationalization;

import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import java.util.Locale;
import java.util.SimpleTimeZone;

public class EjbClient {

  public static void main(String args[]) {

    //--------------------------------------------------
    // INTERNATIONALIZATION SERVICE: API references.
    //--------------------------------------------------
    UserInternationalization userI18n = null;
    Internationalization callerI18n = null;
    InvocationInternationalization invocationI18n = null;

    //--------------------------------------------------
    // INTERNATIONALIZATION SERVICE: JNDI name.
    //--------------------------------------------------
    final String UserI18NUrl =
        "java:comp/websphere/UserInternationalization";

    //--------------------------------------------------
    // INTERNATIONALIZATION SERVICE: Resolve the API.
    //--------------------------------------------------
    try {
     Context initialContext = new InitialContext();
     userI18n = (UserInternationalization)initialContext.lookup(
         UserI18NUrl);
     callerI18n = userI18n.getCallerInternationalization();
     invI18n = userI18n.getInvocationInternationalization ();
    } catch (NamingException ne) {
      log("Error: Cannot resolve UserInternationalization: Exception: " + ne);
    } catch (IllegalStateException ise) {
      log("Error: UserInternationalization is not available: " + ise);
    }
    ...

    //--------------------------------------------------------------------
    // INTERNATIONALIZATION SERVICE: Set invocation context.
    //
    // Under Application-managed Internationalization (AMI), contained EJB
    // client programs may set invocation context elements. The following
    // statements associate the supplied invocation locale and time zone
    // with the current thread. Subsequent remote bean method calls will
    // propagate these context elements.
    //--------------------------------------------------------------------
    try {
      invocationI18n.setLocale(new Locale("fr", "FR", ""));
      invocationI18n.setTimeZone("ECT");
    } catch (IllegalStateException ise) {
      log("An anomaly occurred accessing Invocation context: " + ise );
    }
    ...

    //--------------------------------------------------------------------
    // INTERNATIONALIZATION SERVICE: Get locale and time zone.
    //
    // Under AMI, contained EJB client programs can get caller and
    // invocation context elements associated with the current thread.
    // The next four statements return the invocation locale and time zone
    // associated above, and the caller locale and time zone associated
    // internally by the service. Getting a caller context element within
    // a contained client results in the default element of the JVM.
    //--------------------------------------------------------------------
    Locale invocationLocale = null;
    SimpleTimeZone invocationTimeZone = null;
    Locale callerLocale = null;
    SimpleTimeZone callerTimeZone = null;
    try {
      invocationLocale = invocationI18n.getLocale();
      invocationTimeZone =
          (SimpleTimeZone)invocationI18n.getTimeZone();
      callerLocale = callerI18n.getLocale();
      callerTimeZone = SimpleTimeZone)callerI18n.getTimeZone();
    } catch (IllegalStateException ise) {
      log("An anomaly occurred accessing I18n context: " + ise );
    }
```

```
    ...
  } // main

  ...
  void log(String s) {
    System.out.println (((s == null) ? "null" : s));
  }
} // EjbClient
```

Example: Internationalization context in an EJB servlet

The following code example illustrates how to use the internationalization context API within a servlet. Note the init() and doPost() methods.

```
...
//--------------------------------------------------------------------
// INTERNATIONALIZATION SERVICE: Imports.
//--------------------------------------------------------------------
import com.ibm.websphere.i18n.context.UserInternationalization;
import com.ibm.websphere.i18n.context.Internationalization;
import com.ibm.websphere.i18n.context.InvocationInternationalization;

import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import java.util.Locale;

public class J2eeServlet extends HttpServlet {

  ...
  //------------------------------------------------------------------
  // INTERNATIONALIZATION SERVICE: API references.
  //------------------------------------------------------------------
  protected UserInternationalization userI18n = null;
  protected Internationalization i18n = null;
  protected InvocationInternationalization invI18n = null;

  //------------------------------------------------------------
  // INTERNATIONALIZATION SERVICE: JNDI name.
  //------------------------------------------------------------
  public static final String UserI18NUrl =
      "java:comp/websphere/UserInternationalization";

  protected Locale callerLocale = null;
  protected Locale invocationLocale = null;

  /**
   * Initialize this servlet.
   * Resolve references to the JNDI initial context and the
   * internationalization context API.
   */
  public void init() throws ServletException {

    //------------------------------------------------------------------
    // INTERNATIONALIZATION SERVICE: Resolve API.
    //
    // Under Container-managed Internationalization (CMI), servlets have
    // read-only access to invocation context elements. Attempts to set these
    // elements result in an IllegalStateException.
    //
    // Suggestion: cache all internationalization context API references
    // once, during initialization, and use them throughout the servlet
    // lifecycle.
    //------------------------------------------------------------------
    try {
      Context initialContext = new InitialContext();
      userI18n = (UserInternationalization)initialContext.lookup(UserI18nUrl);
      callerI18n = userI18n.getCallerInternationalization();
      invI18n = userI18n.getInvocationInternationalization();
    } catch (NamingException ne) {
      throw new ServletException("Cannot resolve UserInternationalization" + ne);
    } catch (IllegalStateException ise) {
      throw new ServletException ("Error: UserInternationalization is not
         available: " + ise);
    }
    ...
  } // init

  /**
   * Process incoming HTTP GET requests.
   * @param request Object that encapsulates the request to the servlet
   * @param response Object that encapsulates the response from the
   *    Servlet.
   */
  public void doGet(
      HttpServletRequest  request,
      HttpServletResponse response)
    throws ServletException, IOException {
    doPost(request, response);
  } // doGet

  /**
   * Process incoming HTTP POST requests
   * @param request Object that encapsulates the request to
   *    the Servlet.
   * @param response Object that encapsulates the response from
   *    the Servlet.
   */
  public void doPost(
      HttpServletRequest  request,
      HttpServletResponse response)
    throws ServletException, IOException {

    ...
    //----------------------------------------------------------------------
    // INTERNATIONALIZATION SERVICE: Get caller context.
    //
    // The Internationalization service extracts the accept-languages
    // propagated in the HTTP request and associates them with the
    // current thread as a list of locales within the caller context.
    // These locales are accessible within HTTP Servlet service methods
    // using the caller internationalization object.
    //
    // If the incoming HTTP request does not contain accept languages,
    // the service associates the server's default locale. The service
    // always associates the GMT time zone.
    //
    //----------------------------------------------------------------------
    try {
      callerLocale = callerI18n.getLocale();   // caller locale
      // the following code enables you to get invocation locale,
      // which depends on the Internationalization policies.
      invocationLocale = invI18n.getLocale();  // invocation locale
    } catch (IllegalStateException ise) {
      log("An anomaly occurred accessing Invocation context: " + ise);
    }
    // NOTE: Browsers may propagate accept-languages that contain a
    // language code, but lack a country code, like "fr" to indicate
    // "French as spoken in France."  The following code supplies a
    // default country code in such cases.
    if (callerLocale.getCountry().equals(""))
      callerLocale = AccInfoJBean.getCompleteLocale(callerLocale);
    // Use iLocale in JDK locale-sensitive operations, etc.
    ...
  } // doPost

  ...
  void log(String s) {
    System.out.println (((s == null) ? "null" : s));
  }
} // CLASS J2eeServlet
```

Example: Internationalization context in an EJB session bean

The following code example illustrates how to perform a localized operation using the internationalization service within an EJB session bean.

```
...
//------------------------------------------------------------
// INTERNATIONALIZATION SERVICE: Imports.
//------------------------------------------------------------
import com.ibm.websphere.i18n.context.UserInternationalization;
import com.ibm.websphere.i18n.context.Internationalization;
import com.ibm.websphere.i18n.context.InvocationInternationalization;

import javax.naming.InitialContext;
import javax.naming.Context;
import javax.naming.NamingException;
import java.util.Locale;

/**
 * This is a stateless Session Bean Class
 */
public class J2EESessionBean implements SessionBean {

  //------------------------------------------------------------
  // INTERNATIONALIZATION SERVICE: API references.
  //------------------------------------------------------------
  protected UserInternationalization       userI18n = null;
  protected InvocationInternationalization  invI18n  = null;

  //------------------------------------------------------------
  // INTERNATIONALIZATION SERVICE: JNDI name.
  //------------------------------------------------------------
  public static final String UserI18NUrl =
      "java:comp/websphere/UserInternationalization";
  ...

  /**
   * Obtain the appropriate internationalization interface
   * reference in this method.
   * @param ctx javax.ejb.SessionContext
   */
  public void setSessionContext(javax.ejb.SessionContext ctx) {

    //------------------------------------------------------------
    // INTERNATIONALIZATION SERVICE: Resolve the API.
    //------------------------------------------------------------
    try {
      Context initialContext = new InitialContext();
      userI18n = (UserInternationalization)initialContext.lookup(
          UserI18NUrl);
      invI18n = userI18n.getInvocationInternationalization();
    } catch (NamingException ne) {
      log("Error: Cannot resolve UserInternationalization: Exception: " + ne);

    } catch (IllegalStateException ise) {
      log("Error: UserInternationalization is not available: " + ise);
    }
  } // setSessionContext

 /**
  * Set up resource bundle using I18n Service
  */
  public void setResourceBundle()
  {
    Locale invLocale   = null;

    //------------------------------------------------------------
    // INTERNATIONALIZATION SERVICE: Get invocation context.
    //------------------------------------------------------------
    try {
      invLocale = invI18n.getLocale();
    } catch (IllegalStateException ise) {
      log ("An anomaly occurred while accessing Invocation context: " + ise );
    }
    try {
      Resources.setResourceBundle(invLocale);
      // Class Resources provides support for retrieving messages from
      // the resource bundle(s). See Currency Exchange sample source code.
    } catch (Exception e) {
      log("Error: Exception occurred while setting resource bundle: " + e);
    }
  } // setResourceBundle

  /**
   * Pass message keys to get the localized texts
   * @return java.lang.String []
   * @param key java.lang.String []
   */
  public String[] getMsgs(String[] key) {
    setResourceBundle();
    return Resources.getMsgs(key);
  }

  ...
  void log(String s) {
    System.out.println(((s == null) ? ";null" : s));
  }
} // CLASS J2EESessionBean
```

Internationalization context API: Programming reference

Application components programmatically manage internationalization context through the UserInternationalization, Internationalization, and InvocationInternationalization interfaces in the com.ibm.websphere.i18n.context package. The following code example introduces the internationalization context API:

```
public interface UserInternationalization {
  public Internationalization getCallerInternationalization();
  public InvocationInternationalization
  getInvocationInternationalization();
}

public interface Internationalization {
  public java.util.Locale[] getLocales();
  public java.util.Locale getLocale();
  public java.util.TimeZone getTimeZone();
}

public interface InvocationInternationalization
    extends Internationalization {
  public void setLocales(java.util.Locale[] locales);
  public void setLocale(java.util.Locale jmLocale);
  public void setTimeZone(java.util.TimeZonetimeZone);
  public void setTimeZone(String timeZoneId);
}
```

UserInternationalization interface

The UserInternationalization interface provides factory methods for obtaining references to the CallerInternationalization and InvocationInternationalization context objects. Use these references to access elements of the caller and invocation contexts correlated to the current thread.

Methods of the UserInternationalization interface:

Internationalization getCallerInternationalization()
: Returns a reference implementing the Internationalization interface that allows access to elements of the caller internationalization context correlated to the current thread. If the service is disabled, this method throws an IllegalStateException.

InvocationInternationalization getInvocationInternationalization()
: Returns a reference implementing the InvocationInternationalization interface. If the service is disabled, this method throws an IllegalStateException.

Internationalization interface

The Internationalization interface declares methods affording read-only access to internationalization context. Given a caller or invocation internationalization context object created with the UserInternationalization interface, bind the object to the Internationalization interface in order to get elements of that context type. Observe that caller internationalization context can be accessed only through this interface.

Methods of the Internationalization interface:

Locale[] getLocales()
: Returns the chain of locales within the internationalization context (object) bound to the interface, provided the chain is not null; otherwise this method returns a chain of length(1) containing the default locale of the JVM.

Locale getLocale()
: Returns the first in the chain of locales within the internationalization context (object) bound to the interface, provided the chain is not null; otherwise this method returns the default locale of the JVM.

TimeZone getTimeZone()
: Returns the caller time zone (that is, the SimpleTimeZone) associated with the current thread, provided the time zone is non-null; otherwise this method returns the process time zone.

InvocationInternationalization interface

The InvocationInternationalization interface declares methods affording read and write access to InvocationInternationalization context. Given an invocation internationalization context object created with the UserInternationalization interface, bind the object to the InvocationInternationalization interface in order to get and set elements of the invocation context.

Note: According to the container-managed internationalization (CMI) policy, all set methods, setXxx(), throw an IllegalStateException when called within a CMI servlet or enterprise bean.

Methods of the InvocationInternationalization interface:

void setLocales(java.util.Locale[] locales)
: Sets the chain of locales to the supplied chain, *locales*, within the invocation internationalization context. The supplied chain can be null or have length(>= 0). When the supplied chain is null or has length(0), the service sets the chain of invocation locales to an array of length(1) containing the default locale of the JVM. Null entries can exist within the supplied locale list, for which the service substitutes the default locale of the JVM on remote invocations.

void setLocale(java.util.Locale locale)
: Sets the chain of locales within the invocation internationalization context to an array of length(1) containing the supplied locale, *locale*. The supplied locale can be null, in which case the service instead sets the chain to an array of length(1) containing the default locale of the JVM.

void setTimeZone(java.util.TimeZone timeZone)
: Sets the time zone within the invocation internationalization context to the supplied time zone, *time zone*. If the supplied time zone is not an exact instance of java.util.SimpleTimeZone or is null, the service instead sets the invocation time zone to the default time zone of the JVM.

void setTimeZone(String timeZoneId)
: Sets the time zone within the invocation internationalization context to the java.util.SimpleTimeZone having the supplied ID, *timeZoneId*. If the supplied time zone ID is null or invalid (that is, it does not appear in the list of IDs returned by the java.util.TimeZone.getAvailableIds() method) the service sets the invocation time zone to the simple time zone having an ID of GMT, an offset of 00:00, and otherwise invalid fields.

Internationalization context

An ″internationalization context″ is a distributable collection of internationalization information containing an ordered list, or chain, of locales and a single time zone, where the locales and time zone are instances of Java SDK types, java.util.Locale and java.util.TimeZone. A locale chain is ordered according to the user’s preference.

The internationalization service manages and makes available two varieties of internationalization context: the ″caller context″ representing the caller’s localization environment; and the ″invocation context″ representing the localization environment under which a business method executes. Server application components use elements of the caller and invocation internationalization contexts to appropriately tailor locale-sensitive and time zone-sensitive computations.

Note: The internationalization service does not support time zone types other than Java SDK type java.util.SimpleTimeZone. Unsupported time zone types silently map to the default time zone of the JVM when supplied to internationalization context API methods. For a complete description of the java.util.Locale, java.util.TimeZone, and java.util.SimpleTimeZone types, refer the Java SDK API documentation.

Caller context

Caller internationalization context contains the locale chain and time zone received on incoming EJB business method and servlet service method invocations; it is the internationalization context propagated from the calling process. Use caller context elements within server application components to localize computations to the calling component. Caller context is read-only and can be accessed by all application components by using the **Internationalization** interface of the internationalization context API.

Caller context is computed in the following manner: On an EJB business method or servlet service method invocation, the internationalization service extracts the internationalization context from the incoming request and scopes this context to the method as the caller context. For any missing or null context element, the service inserts the corresponding default element of the JVM (for example, java.util.Locale.getDefault() or java.util.TimeZone.getDefault().)

Formally, caller context is the invocation context of the calling business method or application component.

Invocation context

Invocation internationalization context contains the locale chain and time zone under which EJB business methods and servlet service methods execute. It is managed by either the hosting container or the application component, depending on the applicable internationalization policy. On outgoing business method requests, it is the context that propagates to the target process. Use invocation context elements to localize computations under the specified settings of the current application component.

Invocation context is computed in the following manner: On an incoming business method or servlet service method invocation, the internationalization service queries the associated context management policy. If the policy is container-managed internationalization (CMI), the container scopes the context designated by the policy to the invocation; otherwise the policy is application-managed internationalization (AMI), and the container scopes a vacuous context to the invocation that can be altered by the method implementation.

Application components can access invocation context elements through both the **Internationalization** and **InvocationInternationalization** interfaces of the internationalization context API. Invocation context elements can be set (overwritten) under the application-managed internationalization policy only.

On an outgoing business method request, the service obtains the currently scoped invocation context and attaches it to the request. This outgoing exported context becomes the caller context of the target invocation. When supplying invocation context elements, either for export on outgoing requests or through the API, the internationalization service always provides the most recent element set using the API; also, the service supplies the corresponding default element of the JVM for any null invocation context element.

Internationalization context: Propagation and scope

The scope of internationalization context is implicit. Every EJB client application, servlet service method, and EJB business method invocation has two internationalization contexts under which it executes. For each application component invocation, the container enters the caller context and the invocation context, as indicated by the pertinent internationalization policy, into scope before the container delegates to the actual implementation. When the implementation returns, the service removes these contexts from scope. The internationalization service supplies no programmatic mechanism for components to explicitly manage the scope of internationalization context.

The service scopes internationalization context differently with respect to application component type:
- EJB client programs (contained)
- Servlets
- Enterprise beans

Internationalization context observes by-value semantics over remote method requests, meaning that changes to internationalization context elements scoped to an invocation do not affect the corresponding elements of the internationalization context scoped to the remote calling process. Also, modifications to context elements obtained using the internationalization context API do not affect the corresponding elements scoped to the invocation.

EJB client programs (contained)

Before it invokes the main() method of a client program, the J2EE client container introduces into scope invocation and caller internationalization contexts containing null elements. These contexts remain in scope throughout the life of program. EJB client programs are the base in a chain of remote business method invocations and, technically, do not have a logical caller context. Accessing a caller context element yields the corresponding default element of the client JVM. On outgoing EJB business method requests, the internationalization service propagates the invocation context to the target process. Any unset (null) invocation context elements are replaced with the default of the JVM when exported by the internationalization context API or by outgoing requests.

Tip: To propagate values other than the JVM defaults to remote business methods, EJB client programs, as well as AMI servlets or enterprise beans, must set (override) elements of the invocation context. To learn how to set invocation context elements, see the topic “Accessing invocation locales and time zone” on page 93.

Servlets

On every servlet service method (doGet(), doPost()) invocation, the J2EE web container introduces caller and invocation internationalization contexts into scope before delegating to the service method implementation. The caller context contains the accept-languages propagated in the HTTP servlet request, typically from a Web browser. The invocation context contains whichever context is indicated by the container internationalization attribute of the internationalization policy associated with the servlet. Any unset (null) invocation context elements are replaced with the default of the server JVM when exported by the internationalization context API or by outgoing requests. The caller and invocation contexts remain effective until immediately after the implementation returns, at which time the container removes them from scope.

Enterprise beans

On every EJB business method invocation, the J2EE EJB container introduces caller and invocation internationalization contexts into scope before delegating to the business method implementation. The caller context contains the internationalization context elements imported from the incoming IIOP request; if the incoming request lacks a particular internationalization context element, the container scopes a null element. The invocation context contains whichever context is indicated by the container internationalization attribute of the internationalization policy associated with the business method. On outgoing EJB business method requests, the service propagates the invocation context to the target process. Any unset (null) invocation context elements are replaced with the default of the server JVM when exported by the internationalization context API or by outgoing requests. The caller and invocation contexts remain effective until immediately after the implementation returns, at which time the container removes them from scope.

Consider a simple WebSphere EJB application having a Java client that invokes remote bean method, myBeanMethod(). On the client side, the application could use the Internationalization Service API to set invocation context elements. When the client calls myBeanMethod(), the service exports the client's invocation context to the outgoing request. On the server side, the service detaches the imported context from the incoming request and scopes it to the method as its caller context; it also scopes the invocation context to the method as indicated by the associated internationalization context management policy. The EJB container then calls the myBeanMethod(), which can use the internationalization context API to access elements of either the caller or invocation contexts. When myBeanMethod() returns, the EJB container removes these contexts from scope.

Thread association considerations

The Web and EJB containers scope internationalization contexts to a method by associating it with the thread that executes the method's implementation. Similarly, methods of the internationalization context API either associate context with, or obtain context associated with, the thread on which these methods execute. In cases where new threads are spawned within an application component (for instance, a user-generated thread inside the service() method of a servlet, or a system-generated event handling thread in an AWT client) the internationalization contexts associated with the parent thread does not automatically transfer to the newly-spawned thread. In such instances, the service exports the default locale and time zone of the JVM on any remote business method request and on any API calls executed on the new thread. If the default context is inappropriate, the desired invocation context elements must be explicitly associated to the new thread using the setXxx() methods of the **InvocationInternationalization** interface. Currently, internationalization context management policies allow invocation context to be set within EJB client programs, as well as within servlets, session beans, and message-driven beans employing application-managed internationalization.

Internationalization context: Management policies

Internationalization policies declaratively prescribe how J2EE application components or their hosting containers (the service) will manage internationalization context on component invocations. There are two internationalization context management policies applicable to all component types:

- Application-managed internationalization (AMI)
- Container-managed internationalization (CMI)

These policies are represented in two parts:

- Internationalization type

- Container internationalization attribute

The service defines a default, or implicit, internationalization policy for every application component type. At development time, assemblers can override the default policy for server component types by explicitly configuring their internationalization type, and optional container internationalization attributes, using the WebSphere Application Assembly Tool. Policies configured during assembly are preserved in the application's deployment descriptor.

All components have an internationalization type that indicates whether it is AMI or CMI; that is, whether a component is to deploy under the application-managed or the container-managed internationalization policy. Application assemblers can set the internationalization type for servlets, session beans, and message-driven beans. Entity beans are implicitly CMI and EJB clients are implicitly AMI; neither can be configured otherwise.

For CMI servlets and enterprise beans, optional container internationalization attributes can be specified to indicate which invocation internationalization context the container is to scope to service or business methods. A CMI service or business method invocation can run under the context of the caller's process, under the default context of the server's JVM, or under a custom context specified in the attribute. Assemblers can specify one container internationalization attribute per disjoint set of CMI servlets within a Web module, or one Attribute per disjoint set of business methods of CMI beans within an EJB module. In other words, a container internationalization attribute can be associated with more than one method, but a method cannot be associated with more than one attribute.

When a IBM WebSphere Application Server launches an application, the internationalization service collects policy information from the deployment descriptor, then uses this information to construct and associate an internationalization policy to every component invocation. A policy is denoted as:

```
[<Internationalization Type>,<Container Internationalization Attribute>]
```

There are several cases where the deployment descriptor appears to lack policy information, for example: EJB client applications have no configurable internationalization policy settings; AMI components do not have container internationalization attributes; and you are not required to specify container internationalization attributes for CMI components. When the service cannot obtain the explicit internationalization type and container attribute settings from a well-formed deployment descriptor, it implicitly inserts the appropriate setting into the policy.

The service observes the following conventions when applying policies to invocations:

- Servlets (service) and EJB business methods lacking all internationalization policy information in the deployment descriptor implicitly execute under policy `[CMI,RunAsCaller]`.
- CMI servlets and business methods lacking a container internationalization attribute in the deployment descriptor implicitly execute under policy `[CMI,RunAsCaller]`.
- AMI servlets and business methods always lack container internationalization attributes in the deployment descriptor, but implicitly execute under the logical policy `[AMI,RunAsServer]`.

- EJB clients always lack internationalization policy information in the deployment descriptor. By definition, EJB clients are implicitly AMI and run under the invocation context of the JVM; they execute under the logical policy `[AMI,RunAsServer]`.

For conditions other than these, such as a malformed deployment descriptor, refer to the topic "Internationalization service errors" on page 114.

Internationalization policies for EJB clients and HTTP clients cannot be configured using the Application Assembly Tool; HTTP clients do, however, run under the language priority settings of the hosting Web browser. These settings are configurable under the options dialog of most Web browsers; refer to your Web browser's documentation for details.

Internationalization type: Every server application component has an *internationalization type* setting that indicates whether the invocation internationalization context is to be managed by the component or by the hosting J2EE container.

Server application components can be deployed to use one of two types of internationalization context management:
- Application-managed internationalization (AMI)
- Container-managed internationalization (CMI)

A server component may be deployed as AMI or CMI, but not both; CMI is the default. The setting applies to the entire component, on every invocation. Use the Application Assembly Tool to configure the internationalization type for servlets, session beans, and message-driven beans; entity beans are CMI and cannot be configured otherwise. EJB client applications do not have an internationalization type setting, but are implicitly AMI.

Application-managed internationalization (AMI)

Under the AMI deployment policy, component developers assume complete control over the invocation internationalization context. AMI components can use the internationalization context API to programmatically set invocation context elements.

AMI components are expected to manage invocation context. Invocations of AMI components implicitly run under the default locale and time zone of the hosting JVM. Invocation context elements not set using the API default to the corresponding elements of the JVM when accessed through the API or when exported on business methods. To export context elements other than the JVM defaults, AMI servlets, AMI enterprise beans, and EJB client applications must set (overwrite) invocation elements using the internationalization context API. Moreover, the container logically suspends caller context imported on AMI servlet lifecycle method and AMI EJB business method invocations. To continue propagating the context of the calling process, AMI servlets and enterprise beans must use the API to transfer caller context elements to the invocation context.

Specify AMI for server components that have internationalization context management requirements not supported by container-managed internationalization (CMI).

Container-managed internationalization (CMI)

CMI is the preferred internationalization context management policy for server application components; it is also the default policy. Under CMI, the internationalization service collaborates with the Web and EJB containers to set the invocation internationalization context for servlets and enterprise beans. The service sets invocation context according to the container internationalization attribute of the policy associated with a servlet (service method) or an EJB business method.

A CMI policy contains a container internationalization attribute that indicates which internationalization context the container is to scope to an invocation. For details, see topic "Container internationalization attributes". By default, invocations of CMI components run under the caller's internationalization context; or rather, they adhere to the implicit policy `[CMI,RunasCaller]` whenever the servlet or business is not associated with an attribute in the deployment descriptor. For complete details, see the topic "Internationalization context: Management policies" on page 104.

Methods within CMI components can obtain elements of the invocation context using the internationalization context API, but cannot set them. Any attempt to set invocation context elements within CMI components results in a java.lang.IllegalStateException.

Specify container-managed internationalization for server application components requiring standard internationalization context management, then specify the container internationalization attributes for CMI servlets and for business methods of CMI enterprise beans that should not run under the caller's internationalization context.

Container internationalization attributes: The internationalization policy of every CMI servlet and EJB business method has a *container internationalization attribute* that specifies which internationalization context the container is to scope to its invocation.

The container internationalization attribute has three main fields:
- Run as
- Locales
- Time zone ID

As a convenience, developers can create named container internationalization attributes and associate them to subsets of CMI servlets within a Web module, or to subsets of business methods of CMI enterprise beans within an EJB module.

Run-as field

The Run-as field specifies one of three types of invocation context that a container can scope to a method. For servlet service and EJB business methods, the container constructs the invocation internationalization context according to the Run as field and associates this context to the current thread before delegating to the method's implementation.

Using the Application Assembly Tool, the Run as field is configurable for any CMI servlet and business method of a CMI enterprise bean. By default, invocations of servlet service methods and EJB business methods implicitly run as caller

(RunAsCaller) unless the Run as field of a policy's attribute specifies otherwise. EJB client applications and AMI server components always run as server (RunAsServer).

Invocation context types specifiable with the Run as field are:

Caller The container invokes the method under the internationalization context of the calling process. For any missing context element, the container supplies the corresponding default context element of the JVM. Select run as caller when you want the invocation to execute under the invocation context of the calling process.

Server The container invokes the method under the default locale and time zone of the JVM. Select run as server when you want the invocation to execute under the invocation context of the JVM.

Specified
The container invokes the method under the internationalization context specified in the attribute. Select run as specified when you want the invocation to execute under the custom invocation context specified in the policy, then provide the custom context elements by completing the Locales and Time zone ID fields.

Note: JMS messages do not contain internationalization context. Although container-managed message-driven beans can be configured to run as caller, the container associates the default elements of the server process when invoking the onMessage() method of any message-driven bean configured as `[CMI, RunAsCaller]`

Locales field

The Locales field specifies an ordered list of locales that the container scopes to an invocation. Using the Application Assembly Tool, the Locales field is configurable for CMI servlets and for business methods of CMI enterprise beans that run as specified.

A locale represents a specific geographical, cultural, or political region and contains three fields:

- **Language code**. Ideally, language code is one of the lower-case, two-character codes defined by ISO-639; however, language code is not restricted to ISO codes and is not a required field. A valid locale must specify a language code if it does not specify a country code.
- **Country code**. Ideally, country code is one of the upper-case, two-character codes defined by ISO-3166; however, country code is not restricted to ISO codes and is not a required field. A valid locale must specify a country code if it does not specify a language code.
- **Variant**. Variant is a vendor-specific code. Variant is not a required field and serves only to supplement the language and country code fields according to application- or platform-specific requirements.

A valid locale must specify at least a language code or a country code; the variant is always optional. The first locale of the list is returned when accessing invocation context using the internationalization context API method getLocale().

Time zone ID field

The Time zone ID field specifies a shorthand identifier for a time zone that the container scopes to an invocation. Using the Application Assembly Tool, the Time zone ID field is configurable for CMI servlets and for CMI EJB business methods that run as specified.

A time zone represents a temporal offset and computes daylight savings information. A valid ID indicates any time zone supported by the SDK type, java.util.TimeZone. Specifically, a valid ID is any of the IDs appearing in the list of time zone IDs returned by method java.util.TimeZone.getAvailableIds(), or a custom ID having the form GMT[+|-]hh[[:]mm]; for example, `America/Los_Angeles`, `GMT-08:00` are valid time zone IDs.

Internationalization type

Every server application component has an *internationalization type* setting that indicates whether the invocation internationalization context is to be managed by the component or by the hosting J2EE container.

Server application components can be deployed to use one of two types of internationalization context management:
- Application-managed internationalization (AMI)
- Container-managed internationalization (CMI)

A server component may be deployed as AMI or CMI, but not both; CMI is the default. The setting applies to the entire component, on every invocation. Use the Application Assembly Tool to configure the internationalization type for servlets, session beans, and message-driven beans; entity beans are CMI and cannot be configured otherwise. EJB client applications do not have an internationalization type setting, but are implicitly AMI.

Application-managed internationalization (AMI)

Under the AMI deployment policy, component developers assume complete control over the invocation internationalization context. AMI components can use the internationalization context API to programmatically set invocation context elements.

AMI components are expected to manage invocation context. Invocations of AMI components implicitly run under the default locale and time zone of the hosting JVM. Invocation context elements not set using the API default to the corresponding elements of the JVM when accessed through the API or when exported on business methods. To export context elements other than the JVM defaults, AMI servlets, AMI enterprise beans, and EJB client applications must set (overwrite) invocation elements using the internationalization context API. Moreover, the container logically suspends caller context imported on AMI servlet lifecycle method and AMI EJB business method invocations. To continue propagating the context of the calling process, AMI servlets and enterprise beans must use the API to transfer caller context elements to the invocation context.

Specify AMI for server components that have internationalization context management requirements not supported by container-managed internationalization (CMI).

Container-managed internationalization (CMI)

CMI is the preferred internationalization context management policy for server application components; it is also the default policy. Under CMI, the internationalization service collaborates with the Web and EJB containers to set the invocation internationalization context for servlets and enterprise beans. The service sets invocation context according to the container internationalization attribute of the policy associated with a servlet (service method) or an EJB business method.

A CMI policy contains a container internationalization attribute that indicates which internationalization context the container is to scope to an invocation. For details, see “Container internationalization attributes” on page 107. By default, invocations of CMI components run under the caller’s internationalization context; or rather, they adhere to the implicit policy [CMI,RunasCaller] whenever the servlet or business is not associated with an attribute in the deployment descriptor. For complete details, see “Internationalization context: Management policies” on page 104.

Methods within CMI components can obtain elements of the invocation context using the internationalization context API, but cannot set them. Any attempt to set invocation context elements within CMI components results in a java.lang.IllegalStateException.

Specify container-managed internationalization for server application components requiring standard internationalization context management, then specify the container internationalization attributes for CMI servlets and for business methods of CMI enterprise beans that should not run under the caller’s internationalization context.

Container internationalization attributes

The internationalization policy of every CMI servlet and EJB business method has a *container internationalization attribute* that specifies which internationalization context the container is to scope to its invocation.

The container internationalization attribute has three main fields:

- Run as
- Locales
- Time zone ID

As a convenience, developers can create named container internationalization attributes and associate them to subsets of CMI servlets within a Web module, or to subsets of business methods of CMI enterprise beans within an EJB module.

Run-as field

The Run-as field specifies one of three types of invocation context that a container can scope to a method. For servlet service and EJB business methods, the container constructs the invocation internationalization context according to the Run as field and associates this context to the current thread before delegating to the method’s implementation.

Using the Application Assembly Tool, the Run as field is configurable for any CMI servlet and business method of a CMI enterprise bean. By default, invocations of servlet service methods and EJB business methods implicitly run as caller (RunAsCaller) unless the Run as field of a policy’s attribute specifies otherwise. EJB client applications and AMI server components always run as server (RunAsServer).

Invocation context types specifiable with the Run as field are:

Caller The container invokes the method under the internationalization context of the calling process. For any missing context element, the container supplies the corresponding default context element of the JVM. Select run as caller when you want the invocation to execute under the invocation context of the calling process.

Server The container invokes the method under the default locale and time zone of the JVM. Select run as server when you want the invocation to execute under the invocation context of the JVM.

Specified
The container invokes the method under the internationalization context specified in the attribute. Select run as specified when you want the invocation to execute under the custom invocation context specified in the policy, then provide the custom context elements by completing the Locales and Time zone ID fields.

Note: JMS messages do not contain internationalization context. Although container-managed message-driven beans can be configured to run as caller, the container associates the default elements of the server process when invoking the onMessage() method of any message-driven bean configured as `[CMI, RunAsCaller]`.

Locales field

The Locales field specifies an ordered list of locales that the container scopes to an invocation. Using the Application Assembly Tool, the Locales field is configurable for CMI servlets and for business methods of CMI enterprise beans that run as specified.

A locale represents a specific geographical, cultural, or political region and contains three fields:

- **Language code.** Ideally, language code is one of the lower-case, two-character codes defined by ISO-639; however, language code is not restricted to ISO codes and is not a required field. A valid locale must specify a language code if it does not specify a country code.
- **Country code.** Ideally, country code is one of the upper-case, two-character codes defined by ISO-3166; however, country code is not restricted to ISO codes and is not a required field. A valid locale must specify a country code if it does not specify a language code.
- **Variant.** Variant is a vendor-specific code. Variant is not a required field and serves only to supplement the language and country code fields according to application- or platform-specific requirements.

A valid locale must specify at least a language code or a country code; the variant is always optional. The first locale of the list is returned when accessing invocation context using the internationalization context API method getLocale().

Time zone ID field

The Time zone ID field specifies a shorthand identifier for a time zone that the container scopes to an invocation. Using the Application Assembly Tool, the Time zone ID field is configurable for CMI servlets and for CMI EJB business methods that run as specified.

A time zone represents a temporal offset and computes daylight savings information. A valid ID indicates any time zone supported by the SDK type, java.util.TimeZone. Specifically, a valid ID is any of the IDs appearing in the list of time zone IDs returned by method java.util.TimeZone.getAvailableIds(), or a custom ID having the form GMT[+|-]hh[[:]mm]; for example, `America/Los_Angeles`, `GMT-08:00` are valid time zone IDs.

Managing the internationalization service

To use internationalization context in an EJB application, the internationalization service must be enabled in the run-time environments for all server-side components (servlets and enterprise beans) as well as all client-side components (EJB client applications).

Note: The internationalization service cannot be enabled for HTTP clients because support for internationalization in that case is provided by the browser, not by IBM WebSphere Application Server.

Steps for this task

1. "Enabling the internationalization service for servlets and enterprise beans".

   Use the administrative console to enable the service within all hosting IBM WebSphere Application Servers. The service is disabled by default within IBMWebSphere Application Server Enterprise.
2. "Enabling the internationalization service for EJB clients" on page 113.

   Enable the service within the hosting WebSphere J2EE client environments. The service is enabled by default within the WebSphere J2EE client container.

Enabling the internationalization service for servlets and enterprise beans

Any servlet or enterprise bean can use internationalization context if the internationalization service is enabled within the hosting IBM WebSphere Application Server.

Steps for this task

1. Start the administrative console.
2. Select **Servers** > **Application Servers** > *server_name* > **Internationalization Service**.
3. Enable the Internationalization Service:
   a. If not already selected, select the **Startup** checkbox.
   b. Click **OK**.

Results

When the Startup setting is selected, the application server automatically initializes, starts, and enables the internationalization service whenever the server starts. If you change this setting, be sure to restart the application server in order for the new setting to take effect.

To disable the service, clear the **Startup** checkbox. In this case, the internationalization service initializes, but is neither started nor enabled when the application server starts.

Alternatively, the internationalization service can be enabled from the command line using the wsadmin tool. To do this, start the wsadmin tool and enter the following commands

```
set x [$AdminConfig list I18NService]
$AdminConfig modify $x { { enable true } }
$AdminConfig save
exit
```

Again, if you enable or disable the internationalization service, be sure to restart the application server in order for the new setting to take effect.

Internationalization service settings

Use this page to enable or disable the internationalization service.

The internationalization service manages the implicit propagation and scoping of locale and time zone information, called internationalization context, within WebSphere Enterprise applications. When the service is enabled, server-side application components can use the internationalization context API to programmatically manage locale and time zone information, or to use this information with the J2SE Internationalization API to perform server-side localizations.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Internationalization Service** .

Startup: Specifies whether the server will attempt to start the internationalization service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the internationalization service automatically.

Cleared
The server does not try to start the internationalization service. If internationalization is to be used in applications that run on this server, the system administrator must select this property then restart the server.

Enabling the internationalization service for EJB clients

The internationalization service is enabled for use within EJB client applications whenever the **i18nctx.jar** file is in the CLASSPATH constructed by the launchclient utility. When invoking a Java client application, launchclient configures the CLASSPATH to include the i18nctx.jar file, then activates the WebSphere J2EE client (container), which initializes, starts, and enables the service before delegating to the specified application.

The internationalization service is enabled by default within the WebSphere J2EE client (container). To disable the service, ensure that the i18nctx.jar file is not included in the CLASSPATH. Because the launchclient utility constructs the CLASSPATH on behalf of the J2EE client, you must remove the i18nctx.jar file from the *WAS_HOME*/lib directory. This prevents the file from inadvertently being included in the CLASSPATH constructed by the launchclient utility.

Note: Removing the i18nctx.jar file from the *WAS_HOME*/lib directory prevents any application server in your installation from using the internationalization service. To selectively disable the service, submit the argument `-CCDI18NService.enable=false` or `-CCDI18NService.enable=no` when invoking the launchClient tool.

Troubleshooting the internationalization service

To have your application server emit trace statements for the internationalization service, specify the appropriate trace string to the server's diagnostic trace service.

Steps for this task

1. Start the administrative console.
2. Select **Servers > application servers >** *server_name* **> Diagnostic Trace Service**.
3. Select the **Enable Trace** checkbox.
4. In the **Trace Specification** field, type the following as a continuous string (no spaces and no line breaks):
   ```
   com.ibm.ws.i18n.context.*=all=enabled:
   com.ibm.websphere.i18n.context.*=all=enabled
   ```
5. Click **OK**.
6. Click **Save** on the taskbar.
7. Click **Save** in the **Save to Master Repository** panel.

Results

These settings enable the internationalization service trace when you start or restart the corresponding application server.

Internationalization service errors

The following conditions can occur while your internationalized application is running. These conditions might cause the internationalization service not to start, to throw instances of IllegalStateException, or to exercise default behaviors:

- The service is disabled
- The service is not started
- Invalid context element
- Missing context element
- Invalid policy
- Missing policy

If you encounter unexpected or exceptional behavior, the problem is likely related to one of these conditions. You need to examine the trace log to investigate these conditions, which requires that you configure the diagnostic trace service to generate messages about internationalization service function. To do this, see the topic "Troubleshooting the internationalization service".

The service is disabled

The internationalization service does not initialize and start when the service's startup setting is cleared. The service generates a message indicating whether it is enabled or disabled. Applications cannot access the internationalization API when the service is disabled. If an application attempts a JNDI lookup to obtain the UserInternationationlization reference, the lookup fails with a NamingException indicating the reference could not be found. In addition, the service does not scope (propagate) internationalization context on incoming (outgoing) business method invocations.

The service is not started

The internationalization service is operational whenever it is in the STARTED state. For example, if an application attempts to access internationalization context and the service is not started, the API throws an IllegalStateException. In addition, the service does not provide runtime support for servlets and enterprise beans.

As an application server progresses through its lifecycle, it initializes, starts, stops, and terminates (destroys) the internationalization service. If an anomaly occurs during initialization, the service does not start. Once the service has been started, its state can change to BLOCKED in the event that a serious error occurs. The service generates a message for every state change.

If a trace message indicates that the service is not STARTED, examine previous messages to determine the problem. For instance, the internationalization service does not start if the activity service is unavailable and a message is displayed to that effect during initialization of the internationalization service.

During startup, the following messages indicate potential configuration or run-time problems:

No ORB support
: The service could not obtain an instance of the ORB. This is a fatal error. Examine the logs for information.

No TCM support
: The service could not obtain an instance of its thread context manager. This is a fatal error. Examine the logs for information.

No IIOP (Activity service) support
: The service could not register with the Activity service. This is a fatal error. The internationalization service cannot propagate or receive context on IIOP requests without Activity service support. Review the logs for error conditions related to the Activity service.

No AsynchBeans support
: The service could not register into the AsynchBeans environment. This warning indicates that the AsynchBeans environment cannot support internationalization context. If the application server should have AsynchBeans support, verify that the asynchbeans.jar and asynchbeansimpl.jar files exist in the classpath and review the trace log for any AsynchBeans error conditions.

No EJB container support
: The service could not register with the EJB container. This is a warning that the internationalization service cannot support enterprise beans. Without EJB container support, internationalization, contexts do not scope properly to EJB business methods. Review the trace log for any EJB container-related error conditions.

No Web container support
: The service could not register with the Web container. This is a warning that the internationalization service cannot support servlets and Java Server Pages (JSPs). Without Web container support, internationalization

contexts do not scope properly to servlet service methods. Review the trace log for any Web container-related error conditions.

No Meta-data support
The service could not register with the meta-data service. This is a warning that the internationalization service cannot process the internationalization policies within application deployment descriptors. Without meta-data support, the service associates the default internationalization context management policy, [CMI, RunAsCaller], to every servlet lifecycle method and enterprise bean business method invocation. Review the trace log for any meta-data service-related error conditions.

No JNDI (Name service) support
The service could not bind the UserInternationalization object into the namespace. This is a fatal error. Application components are unable to access internationalization context API references, and are therefore unable to access internationalization context elements. Review the trace log for any Naming (JNDI) service-related error conditions.

No API support
The service could not obtain an instance of an internationalization context API object. This is a fatal error. Application components are unable to access internationalization context API references, and are therefore unable to access internationalization context elements.

Invalid context element

The service detected an invalid internationalization context element. For example, the internationalization service does not support TimeZone instances of a type other than java.util.SimpleTimeZone. If the service encounters an invalid element, it logs a message and substitutes the corresponding default element of the JVM.

Missing context element

The service detected a missing internationalization context element. Incoming requests, for example from application servers not supporting the internationalization service will lack internationalization context. When the service attempts to access a caller internationalization context element, which does not exist in this case, it logs a message and substitutes the corresponding default element of the JVM.

Whenever possible, the internationalization service should be enabled within all clients and hosting application servers comprising a WebSphere enterprise application. For more information see the topic “Managing the internationalization service” on page 112.

Invalid policy

The internationalization service detected a malformed internationalization policy in the application deployment descriptor. At execution, the service replaces the malformed attribute with the appropriate default. For instance, if the internationalization type for an entity bean is set to **Application** during the execution of a servlet or EJB business method invocation, the service logs the inconsistency and enforces the **Container** setting instead.

Also, AMI application components do have an implicit container internationalization attribute. By default they run as server. The service silently enforces the implicit policy, `[AMI, RunAsServer]`, and logs messages to this effect.

Invalid container internationalization attributes are likely to occur when specifying the Locales and Time zone ID fields. When encountering invalid Locales and Time zone ID within attributes, the service replaces each with the corresponding default element of the JVM. Be sure to follow the guidelines provided in the topic "Assembling internationalized applications" on page 82.

Missing policy

The service detected a missing internationalization policy. At execution, the service replaces the missing policy with the appropriate default. For instance, if the internationalization type is missing for a servlet or enterprise bean, the service sets the attribute to Container.

Container internationalization attributes are not mandatory for CMI application components. In the event that a CMI servlet or EJB business method lacks a container internationalization attribute, the service silently enforces the implicit policy `[CMI, RunAsCaller]`.

When an application lacks internationalization policies in its deployment descriptor, or meta-data support is unavailable, the service logs a message and applies the policy `[CMI, RunAsCaller]` on every servlet service method and EJB business method invocation.

For more information, see the following topics:
- "Migrating internationalized applications" on page 81
- "Assembling internationalized applications" on page 82
- "Container internationalization attributes" on page 107
- "Internationalization type" on page 106
- "No meta-data support" on page 116

Internationalization service exceptions

The internationalization service employs one exception: java.lang.IllegalStateException. This exception indicates one of the following things:
- An application component attempted an operation not supported by the service's programming model.

  IllegalStateException is thrown whenever a server application component whose internationalization type is set to container-managed Internationalization (CMI) attempts to set invocation context. This is a violation of the CMI policy, under which servlets and enterprise beans cannot modify their invocation internationalization context.
- An anomaly occurred that disabled the service.

  For instance, if the internationalization service does not properly initialize, the JNDI lookup on the UserInternationalization URL throws a javax.naming.NameNotFoundException containing an instance of IllegalStateException. Refer to the trace log to determine the reason for failure and, if necessary, contact your IBM support representative.

Internationalization: Resources for learning

Use the following links to find relevant supplemental information about internationalization. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to this product but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks™ that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- "Programming instructions and examples"
- "Programming specifications"

Programming instructions and examples
- Java internationalization tutorial

  http://java.sun.com/docs/books/tutorial/i18n/index.html

  An online tutorial that explains how to use the Java 2 SDK Internationalization API.

Programming specifications
- Java 2 SDK, Standard Edition Documentation: Internationalization

  http://java.sun.com/j2se/1.3/docs/guide/intl/

  The Java internationalization documentation from Sun Microsystems, including a list of supported locales and encodings.
- Making the WWW truly World Wide

  http://www.w3.org/International/

  The W3C's effort to make World Wide Web technology work with the many writing systems, languages, and cultural conventions of the global community.
- developerWorks - Unicode

  http://www.ibm.com/developerworks/unicode/

  Articles on various subjects relating to Unicode, from IBM's developerWorks.

Chapter 5. Application profiling

Application profiling enables you to configure multiple access intent policies on the same entity bean. Application profiling reflects the fact that different invocations against the same entity can require different kinds of support from the server run-time environment. For more information, see the topic "Application profiling: Overview".

Steps for this task

1. "Assembling applications for application profiling" on page 122.

   This topic describes how to configure tasks, create application profiles, and configure tasks on profiles, using the Application Assembly Tool.
2. "Managing application profiles" on page 135.

   This topic describes how to add and remove tasks from application profiles using the administrative console.
3. "Using the TaskNameManager interface" on page 137.

   This topic describes how to programmatically set the current task name, but you should use this technique sparingly. Wherever possible, use the declarative method instead, which results in more portable function.

Application profiling: Overview

Application profiling enables developers to identify particular units of work to the IBM WebSphere Application Server, Version 5 run-time environment. The run time can tailor its support to the exact requirements of that unit of work. Access intent is currently the only run time component that makes use of the application profiling functionality. For example, you can configure one transaction to load an entity bean with strong update locks and configure another transaction to load the same entity bean without locks.

Application profiling introduces two new concepts in order to achieve this function: ″tasks″ and ″profiles″.

Tasks A task is a named unit of work within a distributed application. Unit of work in this case means a unique path within the application that may or may not correspond to a transaction or activity session. The name of the path is typically assigned declaratively to a J2EE client or servlet, or to the method of an enterprise bean. This point of configuration marks the head of a graph or subgraph identified by the name of the task; the task name flows from the head of the graph downstream on all subsequent IIOP requests, identifying each subsequent invocation along the graph as belonging to the developer-configured task.

Profiles
A profile is simply a set of policies that are configured not only on the components of an application, but on a set of tasks as well. When an invocation on a bean (whether by a finder method, a cmr getter, or a dynamic query) requires data to be retrieved from the back-end system, the current task associated with the request is used to determine the exact requirement of the transaction; the same bean loads and behaves differently in the context of the task to profile mapping. Each profile provides the developer an opportunity to reconfigure the application's

access intent. If a request is operating in the absence of a task, the run-time environment uses the access intent configuration external to the application profiles.

Tasks

Tasks are named units of work. They are the mechanism by which the run-time environment determines which access intent policies to apply when an entity bean's data is loaded from the back-end system.

Application profiles enable developers to configure an entity bean with multiple access intent policies; if there are *n* instances of profiles in a given application, each bean can be configured with as many as *n* access intent policies.

A task is a unit of work that is given a name by a developer. A task is assigned to any thread executing within a J2EE component, then propagated implicitly across all IIOP requests. The IBM WebSphere Application Server run-time environment queries the task at the invocation of any entity bean, and establishes the appropriate access intent policy with which an entity instance will be associated. A task typically corresponds to the execution of a concrete and high-level job within the application.

If an entity bean is loaded in a unit of work that is not associated with a task, or is associated with a task that is unassociated with an application profile, the method-level access intent configuration is applied. If a unit of work is associated with a task that is configured with an application profile, the bean-level access intent configuration within the appropriate application profile is applied.

For example, consider a school district application that calls through a session bean in order to interact with student records. One method on the session bean allows administrators to modify the students' records; another method supports student requests to view their own records. Without application profiling, the two tasks would operate anonymously and the run-time environment would be unable to distinguish work operating on behalf of one task or the other. To optimize the application, a developer can configure one of the methods on the session bean with the task ″updateRecords″ and the other method on the session bean with the task ″readRecords″. When registered with an application profile that has the student bean configured with the appropriate locking access intent, the ″updateRecords″ task is assured that it is not unnecessarily blocking transactions that need to only read the records.

Developers declare tasks using the Application Assembly Tool. Tasks can be declared in three ways:

- Tasks can be associated with J2EE application clients, servlets, and JSPs; requests from these components are then associated with the appropriate task.
- Tasks can be associated with methods of an enterprise bean using a task run-as policy. There are two run-as policies:

  run as caller

  This is the default policy under which all methods run. If a request is already associated with a task, the configured method imports and runs under that task. If the request is not associated with a task, the request operates in the absence of a task and beans are controlled using the method-level access intent as configured outside of an application profile.

run as specified
Under the run as specified policy, requests to the configured methods never run under an imported task; instead, a specified task names the unit of work beginning with the method.

- Tasks can be associated with any point in the execution of a component by using the programmatic interface of the task name manager. Any task that can be applied programatically must first be declared for the component; attempts to set unknown task names result in an IllegalTaskNameException error.

Application profiles

An application profile is the set of access intent or query intent policies that should be selectively applied, as well as the list of tasks for which the policies should be applied.

The intention of application profiling is to enable an application to run under a different set of policies depending on the active task under which the application is operating.

Consider an application that centralizes the student records for a school district. These records are frequently accessed by the school district's central office in order to generate reports. The report generation process would be optimized if it held no locks with the back-end system, and if the records could be read into memory with as few back-end operations as possible. Occasionally, however, the records are updated by the students' instructors. Without the ability to distinguish between transactions, the developer is forced to assume a worst-case scenario and, wishing to use pessimistic concurrency, lock the records for all transactions.

Using the application profiling service, the developer can configure in as many ways as necessary the access intent under which the students' records are loade . Under one profile, the records can be configured with an exclusive pessimistic update intent, not only locking-out competing transactions but ensuring that the student is not removed from the system before the transaction completes. Under another profile, the records can be configured with an optimistic intent as part of an object graph that is read from the back-end system in a single database operation. Any task configured with the pessimistic profile receives the strong-locking semantics required for certain transactions, while tasks configured with the optimistic profile receive the performance benefits appropriate for other transactions.

Multiple tasks can be configured on a single profile, indicating that different units of work can have the same requirements on the application; however, the same task cannot be registered with multiple application profiles because the run-time environment would have to guess which set of policies the developer wanted to have applied.

Use the Application Assembly Tool (AAT) to create and configure application profiles. Application profiles span the entire scope of an application. When a profile is created within a module, that profile is automatically created within all other EJB modules and at the EAR file level of the application. Likewise, when a profile is created at the EAR file level of an application, the profile is automatically pushed down to all EJB modules of the application. When an EJB module is imported into an application, all profiles within the module, and the profiles already declared inside the application, are merged.

Assembling applications for application profiling

Before you begin

Application profiling enables multiple access intent policies to be configured on the same entity bean, to be applied for a particular unit of work. Before using application profiles, you need to first(apply access intent policies to entity beans). You can use the one of the default policies or create your own, as described in the topic, (Creating a custom access intent policy).

Steps for this task

1. Configuring tasks.

   Declaratively configure tasks using the Application Assembly Tool, as described in the following topics:

   - "Configuring a component task policy" on page 130.
   - "Configuring a container task policy" on page 130.

   On rare occasions, you might find it necessary to configure tasks *programatically*. Application profiling supports this requirement with a simple interface that enables both overriding of the current task associated with the thread of execution, and resetting of the current task to the original task. See the topic "Using the TaskNameManager interface" on page 137.
2. "Creating an application profile" on page 132.
3. "Configuring tasks on application profiles" on page 134.

Using access intent policies

You can use access intent policies to help the product run-time environment manage various aspects of Enterprise JavaBeans™ (EJB) persistence. You apply access intent policies to methods of EJB Version 2.0 entity beans by using the Application Assembly Tool. This product provides a set of default access intent policies. You can also create your own custom policies.

Steps for this task

1. Apply access intent policies to methods by "Applying access intent policies to methods" on page 126 of CMP entity beans.
2. **(Optional)** Create a custom access intent policy by using the Application Assembly Tool.
3. **(Optional)** Apply access intent policies to BMP entity bean methods by "Using the AccessIntent API" on page 127.
4. **(Optional)** Apply multiple access intent policies to methods by using Chapter 5, "Application profiling", on page 119.

Access intent policies

An access intent policy is a named set of properties (access intents) that governs data access for Enterprise JavaBeans (EJB) persistence. You can assign a policy to individual methods on an entity bean's home, remote, or local interfaces during assembly. If you have the IBM WebSphere Application Server Enterprise product installed, you can assign these during development as well. Access intents are settable only within EJB Version 2.x-compliant modules for entity beans with bean-managed persistence or with CMP Version 2.x.

This product supplies a number of access intent policies that specify permutations of read intent and concurrency control; the pessimistic/update policy can be qualified further. The selected policy determines the appropriate isolation level and locking strategy used by the run-time environment.

Access intent policies are specifically designed to supplant the use of isolation level and access intent method-level modifiers found in the extended deployment descriptor for EJB version 1.1 enterprise beans. You cannot specify isolation level and read-only modifiers for EJB version 2.0 enterprise beans.

Access intent policies are named and defined at the module level. A module can have one or many such policies. Policies are assigned, and apply, to individual methods of the declared interfaces of entity beans and their associated home interfaces. A policy is acted upon by either the combination of the EJB container and persistence manager (for entity beans with container-managed persistence) or directly by entity beans with bean-managed persistence.

For entity beans that are backed by tables with nullable columns, use an optimistic policy with caution. Nullable columns are automatically excluded from overqualified updates at deployment time; concurrent changes to a nullable field might result in lost updates. When used with the IBM WebSphere Studio Application Developer product, this product provides support for selecting a subset of the nonnullable columns that are to be reflected in the overqualified update statement that is generated in the deployment code to support optimistic policies.

A method that is configured with a read-only policy that causes a bean to be activated can cause problems if updates are attempted within the same transaction. Those changes will not be committed, and an exception will be thrown because data integrity might be compromised.

Concurrency control: Concurrency control is the management of contention for data resources. A concurrency control scheme is considered *pessimistic* when it locks a given resource early in the data-access transaction and does not release it until the transaction is closed. A concurrency control scheme is considered *optimistic* when locks are acquired and released over a very short period of time at the end of a transaction.

The objective of optimistic concurrency is to minimize the time over which a given resource would be unavailable for use by other transactions. This is especially important with long-running transactions, which under a pessimistic scheme would lock up a resource for unacceptably long periods of time.

Under an optimistic scheme, locks are obtained immediately before a read operation and released immediately afterwards. Update locks are obtained immediately before an update operation and held until the end of the transaction.

To enable optimistic concurrency, this product uses an *overqualified update scheme* to test whether the underlying data source has been updated by another transaction since the beginning of the current transaction. With this scheme, the columns marked for update and their original values are added explicitly through a WHERE clause in the UPDATE statement so that the statement fails if the underlying column values have been changed. As a result, this scheme can provide column-level concurrency control; pessimistic schemes can control concurrency at the row level only.

Optimistic schemes typically perform this type of test only at the end of a transaction. If the underlying columns have not been updated since the beginning of the transaction, pending updates to container-managed persistence fields are committed and the locks are released. If locks cannot be acquired or if some other transaction has updated the columns since the beginning of the current transaction, the transaction is rolled back: All work performed within the transaction is lost.

Pessimistic and optimistic concurrency schemes require different transaction isolation levels. Enterprise beans that participate in the same transaction and require different concurrency control schemes cannot operate on the same underlying data connection.

Whether or not to use optimistic concurrency depends on the type of transaction. Transactions with a high penalty for failure might be better managed with a pessimistic scheme. (A high-penalty transaction is one for which recovery would be risky or resource-intensive.) For low-penalty transactions, it is often worth the risk of failure to gain efficiency through the use of an optimistic scheme. In general, optimistic concurrency is more efficient when update collisions are expected to be infrequent; pessimistic concurrency is more efficient when update collisions are expected to occur often.

Read-ahead hints: Read-ahead schemes enable applications to minimize the number of database roundtrips by retrieving a working set of container-managed persistence (CMP) beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data for their related beans, which ensures that data is present for the beans that are most likely to be needed next by an application. A *read-ahead hint* is a canonical representation of the related beans that are to be read. It is associated with a finder method for the requested bean type, which must be an EJB 2.x-compliant CMP entity bean.

Read-ahead hints can be set only through the Add Access Intent wizard of the IBM WebSphere Studio Application Developer product. In the wizard, the **Read Ahead Hint** check box is enabled only with access intent policies with optimistic concurrency.

Read-ahead is limited to optimistic policies because locking persistent data store for all beans represented in the hint would be more likely to cause lock conflicts, and optimistic policies do not obtain locks until immediately before the database operation.

Currently, only findByPrimaryKey methods can have read-ahead hints. Only beans related to the requested beans by a container-managed relationship (CMR), either directly or indirectly through other beans, can be read ahead.

A read-ahead hint takes the form of a character string. You do not have to provide the string; the wizard generates it for you based on CMRs defined for the bean. The following example is provided as supplemental information only.

Suppose a CMP bean type A has a finder method that returns instances of bean A. A read-ahead hint for this method is specified using the following notation:
`<em>RelB</em>.<em>RelC</em>; <em>RelD</em>`

Interpret the preceding notation as follows:
- Bean type A has a CMR with bean types B and D.
- Bean type B has a CMR with bean type C.

For each bean of type A that is retrieved from the database, its directly-related B and D beans and its indirectly-related C beans are also retrieved. The order of the retrieved bean data columns in each row of the result set is the same as their order in the read-ahead hint: an A bean, a B bean (or null), a C bean (or null), a D bean (or null). For hints in which the same relationship is mentioned more than once (for example, `<em>RelB</em>.<em>RelC</em>;<em>RelB</em>.<em>RelE</em>`), a bean's data columns appear only once, at the position it first appears in the hint.

The tokens shown in the notation (*RelB* and so on) must be CMR field names for the relationships as defined in the deployment descriptor for the bean. In indirect relationships such as `<em>RelB</em>.<em>RelC</em>`, *RelC* is a CMR field name defined in the deployment descriptor for bean type B.

A single read-ahead hint cannot refer to the same bean type in more than one relationship. For example, if a Department bean has a relationship *employees* with the Employee bean and also has a relationship *manager* with the Employee bean, the read-ahead hint cannot specify both *employees* and *manager*.

For more information about how to set read-ahead hints, see the documentation for the Websphere Studio Application Developer product.

Access intent service

Access intent is a IBM WebSphere Application Server run-time service that enables you to more precisely manage an application's persistence. The access intent service defines a set of declarative annotations used by the Enterprise JavaBeans (EJB) container and its agents to make performance optimizations for entity bean access. These annotations are organized into sets called *access intent policies*.

Access intent policies contain a set of annotations considered as hints by the EJB container and its agents. Most access intent policies are hints representing high-level abstractions that can be mapped to a specific backend resource manager. It is the responsibility of the EJB persistence machinery to ensure the necessary concurrency control, connection, and cache management when carrying out the persistence details. The EJB persistence manager can use access intent hints to make better performance decisions when carrying out its assigned task. A smaller number of access intents are hints to the EJB container, influencing the management of EJB collections.

You can apply access intent policies to methods within the scope of an EJB module, in which case the policy becomes the default access intent for all requests upon the configured methods.

You can also apply access intent policies to methods within the scope of application profiles. Consequently, you can configure methods with multiple and opposing access intent policies. The application profiling documentation explains in more detail how to configure an application to apply a particular access intent policy to a method for one request, then apply another access intent policy to the same method for a different request.

Access intent with BMP entity beans: Access intent's declarative functionality provides great power to you as a CMP entity bean developer. You can provide hints on how IBM WebSphere Application Server is to manage the details of persistence without having to explicitly manage any of the persistence logic from within the application.

There are situations, however, in which you might need to develop BMP entity beans. Because the only meaningful difference between BMP and CMP components is who provides the persistence logic, BMP entity beans should be able to leverage access intent hints just as IBM WebSphere Application Server does on behalf of CMP entity beans. BMP entity beans that use the access intent service participate in application profiling; that is, the value of the access intent attributes can differ from request to request, allowing the BMP entity bean to seamlessly modify its persistence strategy.

You can apply access intent policies to BMP entity bean methods as well as CMP entity bean methods. Because access intent hints are not contractual in nature, there is no obligation for a BMP entity bean to exploit them. BMP entity beans are expected to use only those access intent attributes that are important to that particular bean.

The current access intent policy is bound into the java:comp namespace for a particular BMP entity bean. That policy is current only for the duration of the method call during which the access intent policy was retrieved. In a typical scenario, you would cache the access type during invocation of the ejbLoad() method so that appropriate actions can be taken during invocation of the ejbStore() method.

Access intent design considerations

Use the access intent service to solve clear performance problems. Identify usage patterns that lead to poor application performance and apply appropriate access intent policies.

Refrain from over-tuning an application. You can introduce errors by incorrectly using the access intent service. For example, misuse of the wsPessimisticUpdate-NoCollision policy can result in lost updates; inappropriately setting the collection increment value can introduce performance issues; and problem determination is more difficult when an application is confusingly configured with multiple access intent policies. Clarity and simplicity should be your guiding principles when using the access intent service. This is even more important when applying access intent polices within the scope of application profiles (a feature of IBM WebSphere Application Server Enterprise).

Even though access intent policies can be configured on any method of an entity bean, some attributes of a policy can only be leveraged by the run-time environment under certain conditions. For example, concurrency and access intent are only used for CMP entity beans when the ejbLoad() method is driven to open a connection to, and read data from, a given resource; that data is cached and used to drive the proper queries during invocation of the ejbStore() method. Read-ahead hints are only used during the execution of a finder for a bean. Finally, the collection increment and resource manager prefetch increment are only used on multi-object finders. Configuring policies on methods that will not use the policy is not an error (only certain attributes of any policy are used, even when the policy is appropriately applied to a method). However, configuring policies unnecessarily throughout an application obscures the design of the application and complicates the maintenance of the application.

Applying access intent policies to methods

You apply an access intent policy to a method, or set of methods, in an application's entity beans through the Application Assembly Tool (AAT).

Steps for this task

1. Start the AAT.
2. Create or edit the application EAR file.

For example, to change attributes of an existing application, select **File > Open**, then select the EAR file.

3. Select **EJB Modules >** ***moduleName*** **> Access Intent**.
4. To configure a new access intent policy, right-click and select **New**.
5. On the **New Access Intent** panel, specify a name and a description.

   These attributes are provided as a convenience to the developer and are not used at run time.
6. To select the methods to which the access intent policy should apply, click **Add** beside the Methods table.
7. From the **Applied access intent** list, select an access intent policy.
8. **(Optional)** To override an attribute defined in the applied policy, click **Add** beside the Access intent attribute overrides table.
9. Click **OK** to exit the New Access Intent panel.
10. Save your configuration by selecting **File > Save**.

Using the AccessIntent API

This task describes how to programmatically retrieve and call the AccessIntent API during the execution of BMP entity bean methods.

Steps for this task

1. Look up the current access intent in the namespace.

   For example:

   ```
   InitialContext ic = new InitialContext();
   AccessIntent ai = ic.lookup("java:comp/websphere/AppProfile/AccessIntent");
   ```
2. Call the necessary get() methods.

   For example:

   ```
   int concurrency = ai.getConcurrencyControl();
   int accessType = ai.getAccessType();
   if ( (concurrency == AccessIntent.CONCURRENCY_CONTROL_PESSIMISTIC)
      && (accessType == AccessIntent.ACCESS_TYPE_UPDATE) ) {
        boolean exclusive = ai.getPessimisticUpdateHintExclusive();
        // . . .
   }
   // . . .
   ```

Results

Note: The access intent object reference retrieved from the java:comp lookup is current for the duration of the method in which the reference was looked up. Depending on how you configured the application profile, subsequent calls of the same method might not retrieve the same access intent reference. You can only look up the object reference during the call of a BMP entity bean's method; the reference does not exist during a request on a CMP entity bean. Therefore, access intent object references should not be cached beyond, or used outside of, the scope of the execution of any given BMP method.

Access intent assembly settings

Access intent policies contain data-access settings for use by the persistence manager. Specify one or more methods and associate an access intent policy with each method.

These settings are applicable only for EJB 2.x-compliant entity beans that are packaged in EJB 2.x-compliant modules. Connection sharing between beans with bean-managed persistence and those with container-managed persistence is possible if they all use the same access intent policy.

Name: Specifies a name for the mapping between an access intent policy and one or more methods.

Description: Contains text that describes the mapping.

Methods - Name: Specifies the name of an enterprise bean method, or the asterisk character (*). The asterisk is used to denote all of the methods of an enterprise bean's remote and home interfaces.

Methods - Enterprise bean: Specifies which enterprise bean contains the methods indicated in the Name setting.

Methods - Type: Used to distinguish between a method with the same signature that is defined in both the home and remote interface. Use `Unspecified` if an access intent policy applies to all methods of the bean.

Data type
String

Range Valid values are `Home`, `Remote`,`Local`, `LocalHome` or `Unspecified`

Methods - Parameters: Contains a list of fully qualified Java type names of the method parameters. This setting is used to identify a single method among multiple methods with an overloaded method name.

Applied access intent: Specifies how the container must manage data access for persistence.

Data type
String

Default
`wsPessimisticUpdate-WeakestLockAtLoad`. However, this policy cannot be used with Oracle; see the table that follows.

Range Valid settings are `wsPessimisticUpdate`, `wsPessimisticUpdate-NoCollision`, `wsPessimisticUpdate-Exclusive`, `wsPessimisticUpdate-WeakestLockAtLoad`, `wsPessimisticRead`, `wsOptimisticUpdate`, or `wsOptimisticRead`. Only `wsPessimisticRead` and `wsOptimisticRead` are valid when class-level caching is enabled in the EJB container.

This product supports lazy collections. For each segment of a collection, iterating through the collection (*next()*) does not trigger a remote method call to retrieve the next remote reference. Two policies (`wsPessimisticUpdate` and `wsPessimisticUpdate-Exclusive`) are extremely lazy; the collection increment size is set to 1 to avoid overlocking the application. The other policies have a collection increment size of 25.

If a method is not configured with an access intent policy, the run-time environment typically uses `wsPessimisticUpdate-WeakestLockAtLoad` by default. If, however, the Lifetime in cache property is set on the bean, the default value of **Applied access intent** is `wsOptimisticRead`; updates are not permitted. If a method of a **Lifetime in cache**-configured bean is configured with an access intent policy that permits updates, the application will not run until the method or bean is reconfigured.

Additional information about valid settings follows:

Profile name	Concurrency control	Access type	Transaction isolation
wsPessimisticRead (Note 1)	pessimistic	read	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate (Note 2)	pessimistic	update	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate-Exclusive (Note 3)	pessimistic	update	serializable
wsPessimisticUpdate-NoCollision (Note 4)	pessimistic	update	read committed
wsPessimisticUpdate-WeakestLockAtLoad (Note 5)	pessimistic	update	Repeatable read
wsOptimisticRead	optimistic	read	read committed
wsOptimisticUpdate (Note 6)	optimistic	update	read committed

Notes:

1. Read locks are held for the duration of the transaction.
2. The generated SELECT FOR UPDATE query grabs locks at the beginning of the transaction.
3. SELECT FOR UPDATE is generated; locks are held for the duration of the transaction.
4. A plain SELECT query is generated. No locks are held, but updates are permitted. Relative to `wsPessimisticUpdate`, this difference results in generally better transaction throughput.
5. The generated SELECT query does not include `FOR UPDATE`; locks are escalated by the persistent store at storage time if updates were made.

   Do not use this policy with Oracle; doing so results in a NoSuchDataAccessSpec exception. Comparable alternatives are `wsPessimisticUpdate-NoCollision` or `wsOptimisticUpdate`. If you choose `wsOptimisticUpdate`, be sure to review the rules for forming overqualified-update query predicates. Certain column types (for example, BLOB) are ineligible for inclusion in the overqualified-update query predicate and might affect your design.
6. Generated overqualified-update query forces failure if CMP column values have changed since the beginning of the transaction.

Access intent best practices

This topic outlines issues to consider when applying access intent policies to Enterprise JavaBeans (EJB) methods.

- **Start with defaults.** The default access intent policy (`wsPessimisticUpdate-WeakestLockAtLoad`) loads persistent data with the weakest lock that is supported by the persistent store (typically a read lock). Updates are allowed, and the database is permitted to undertake lock escalation when necessary. This option generally works best for most EJB application patterns. After your application is built and running, you can more finely tune certain access paths in your application.
- **Don't mix access types.** Avoid using both pessimistic and optimistic policies in the same transaction. For most databases, pessimistic and optimistic policies use

different isolation levels. This results in multiple database connections, which prevents you from taking advantage of the performance benefits possible through connection sharing.

- **Access intent for the ejbSelect method must be applied indirectly.** Because ejbSelect methods are not exposed through a home, remote, or local interface, you cannot apply a policy to them directly. An ejbSelect method is called by a home or business method, so apply the appropriate policy to the home or business method that governs the behavior of the ejbSelect method.
- **Take care when applying `wsPessimisticUpdate-NoCollision`.** This policy does not ensure data integrity. No database locks are held, so concurrent transactions can overwrite each other's updates. Use this policy only if you can be sure that only one transaction will attempt to update persistent store at any given time.

Configuring a component task policy

Use the Application Assembly Tool to configure a component's own task. A servlet's or application client's own task is associated with distributed requests from the component unless overridden by a container task policy. Components without a configured task of their own run without a task; units of work without a task and application profile load entities using the method-level access intent configuration, without the support of application profiling.

Steps for this task

1. Start the Application Assembly Tool.
2. Create or edit the application EAR file.

   For example, to change attributes of an existing application, select **File > Open** then select the EAR file.
3. Select **Web Modules >** *module_name*.war **> Web Components >** *component_name*
4. Select the **WAS Enterprise** tab.
5. Select the **Own task** checkbox. Provide a name and description for the task.

   Task names do not have to be unique within an application; however, task names should be shared consciously and conservatively; at run time, all tasks with the same name are treated the same way, regardless of where the task was configured.

   The description is provided as a convenience to the developer and is not used by the run-time environment.
6. Click **OK**.
7. **(Optional)** Select **File > Verify** to verify your configuration.
8. Select **File > Save** to save your configuration.

What to do next

"Configuring tasks on application profiles" on page 134.

Configuring a container task policy

Use the Application Assembly Tool to apply a container task policy to a method, or set of methods, for an application's entity beans. A container task policy defines the task under which the method is invoked. A method can run with an imported task, with its own task, or with a specified task.

Note: Applications use application profiling on a *per task* basis. As soon as a path within an application is configured with a task and that task is associated with an application profile, any entity enlisted within units of work in the scope of that task cease to use the method-level access intent configuration and instead use the bean-level access intent configuration of the application profile with which the task is registered.

Steps for this task

1. Start the Application Assembly Tool.
2. Create or edit the application EAR file.

   For example, to change attributes of an existing application, select **File > Open** then select the EAR file.
3. Select **EJB Modules >** *module_name* **> Container Tasks**.
4. To create a new container task policy, select **File > New > Selected Object**.
5. On the **New Container Task** panel, specify a name and a description.

   These attributes are provided as a convenience to the developer and are not used at runtime.
6. To select the methods to which the container task policy should apply, click **Add** beside the **Methods** table.
7. Select one of the these attributes:

   Run as caller
   This is the default attribute applied to EJB method invocations. If the configured methods are invoked with an associated task, the method is executed with the imported task. If a method is invoked by a request that is unassociated with a task, the method continues to execute without a task.

   Run as specified
   The configured methods are never invoked with an imported task. Instead, the method executes as the specified task name. The task name can be selected from the pull-down menu or you can enter a new task name. The description is provided as a convenience to the developer and is not used at runtime.
8. Click **OK**.
9. **(Optional)** Select **File > Verify** to verify your configuration.
10. Select **File > Save** to save your configuration.

What to do next

“Configuring tasks on application profiles” on page 134.

Container assembly settings for tasks

Use this page to configure container task policies.

Tasks identify units of work within a distributed application by the implicit propagation of the task name on remote requests. The task name is configured on application profiles in order to customize the access intent for the units of work associated with that task.

A container task policy instructs the container under which task a request upon an EJB method should operate. Methods can be configured to run as a caller’s task, as the task configured on the bean, or as a specified task.

Name: The name of the policy.

An optional field provided for the convenience of the developer. Although Name is a required field, the name is not used except as a label within the application assembly tool.

Data type
String

Description: A description of the policy.

An optional field provided for the convenience of the developer.

Data type
String

Methods: The methods upon which the container will apply the task policy.

To add a new method to the policy, click **New**. Expand the tree to select the method or methods from the EJB module. Be sure that each method has been configured no more than once with a container task policy. To remove a method, select it and click **Remove**.

Name: The policy that the container should apply when the configured set of methods are invoked.

Default
Run as caller

Range

Run as caller
If the client invokes the bean method with an associated task, the container invokes the bean method with the same task. If the client invokes the bean method without a task, the container invokes the bean method with the task configured as the bean's default task.

Run as specified
The container invokes the bean method with the specified task.

Name The name of the task

Description
The description of the task

Creating an application profile

Use the Application Assembly Tool to create an application profile. An application profile contains a set of access intent policies applied to an application's entity beans. The access intent policies are only applied for requests that are associated with tasks configured on the application profile.

Steps for this task

1. Start the Application Assembly Tool.
2. Create or edit the application EAR file.

   For example, to change attributes of an existing application, select **File > Open**, then select the EAR file.
3. You can create application profiles at the scope of either an application EAR file or an EJB module.

   To create an application profile at the EAR file scope, select **Application Profile**.

To create an application profile at the module scope, expand **EJB Modules >** *module_name* **> Application Profile**.

An application profile logically spans the application EAR file and all contained EJB Jar files. When a profile is created for the EAR file, the profile is automatically created within all EJB Jar files. When a profile is created within an EJB Jar file, the profile is automatically created for all remaining EJB Jar files and the EAR file as well. It makes no difference where an application profile is created, edited, or removed.

4. To create a new application profile, select **File > New > Selected Object**.
5. On the **New Application Profile** panel, specify a name and a description.

   The name of the profile must be unique within the application; there cannot be two distinct application profiles with the same name.

   The description is provided as a convenience to the developer and is not used at runtime.
6. Click **OK**.
7. Be sure that your newly-created application profile is selected. In the navigation pane, expand the new application profile, then select the **Access Intent** node.
8. To apply access intents within the scope of the application profile, follow the steps as described in the topic Applying access intent policies to entity beans. Any custom access intent policies are available within the application profile.
9. Select **File > Verify** to verify your configuration.
10. Select **File > Save** to save your configuration.

Application profile assembly settings

Use this page to configure application profiles.

Application profiles support the definition of alternate access-intent configurations that are mapped to particular requests identified by an association with a task name.

Name

The name of this application profile.

The name must be unique; multiple profiles cannot share the same name.

The creation, configuration, and deletion of profiles is reflected in the configuration of profiles at both the application and module scope

Data type
String

Description

A description of the application profile

An optional field provided for the convenience of the developer and administrator.

Data type
String

Tasks

Tasks that are configured to operate under the application profile.

Requests associated with any of the configured tasks operate under the access-intent policies configured with the profile within the EJB modules.

To add a task that has been declared within the application, click **Add**. To add a task that has not been declared within the application, click **New**.

To remove a task, select the task and click **Remove**. Any given task can be configured on only one application profile.

Name

The name of the task. Select the name of the task from the pulldown menu or specify a new name. The name of the task must be unique among the set of application profiles.

The task name is a required field.

Data type
String

Description

A description of the task.

An optional field provided for the convenience of the developer and administrator.

Data type
String

Configuring tasks on application profiles

Use the Application Assembly tool to associate tasks with application profiles. When a task is configured on an application profile, the access intent policies defined in the profile are applied, as appropriate, to requests associated with that task.

Steps for this task

1. Start the Application Assembly Tool.
2. Create or edit the application EAR file.

   For example, to change attributes of an existing application, select **File > Open**, then select the EAR file.
3. Associate tasks with an application profile at the scope of either an application EAR file or an EJB module.

   To edit the tasks within an application profile at the EAR file scope, select **Application Profile** in the navigation pane, then select the appropriate profile from the list of application profiles.

   To edit the tasks within an application profile at the EJB module scope, expand **EJB Modules** > *module_name* > **Application Profile**, then select the appropriate application profile from the list of application profiles.

   An application profile logically spans the application EAR file and all contained EJB Jar files. Adding a task to, or removing a task from, an application profile within any module or EAR file automatically causes the task to be removed from the same profile in other modules.

4. To add a task defined elsewhere in the application, click **Add** beside the table of tasks and select a task name from the dropdown menu. To add a task that you defined outside of the application, click **New** beside the table of tasks.
5. Select **File** > **Verify** to verify your configuration.
6. Select **File** > **Save** to save your configuration.

Managing application profiles

Manage your application profiles using the administrative console. From the console, you can add tasks to, and remove tasks from, application profiles.

Steps for this task

1. Start the administrative console.
2. Select **Applications** > **Applications** > *application_name* > **Application Profile** > *profile_name* > **Tasks**.
3. On the Tasks collection page, you can add new tasks to the profile, delete tasks, edit current task settings, and so on.

   Note: No task can, within the scope of an application, be configured on more than one application profile. In such a situation, your application cannot be restarted until you correct the configuration.
4. Save your configuration.
5. Restart the application in order for your changes to take affect.

Application profiling exceptions

The following exceptions are thrown in response to various illegal actions related to application profiling:

com.ibm.ws.exception.RuntimeWarning
This exception is thrown when the application is started, if the application is configured incorrectly. The startup is consequently terminated. You can validate an application's configuration by using the **Verify** function in the Application Assembly Tool. Some examples of misconfiguration include:

- A task configured on two different application profiles.
- A method configured with two different task run-as policies .

com.ibm.websphere.appprofile.IllegalTaskNameException
This exception is raised if an application attempts to programmatically set a task when that task has not been configured as a task name reference.

Application profiling service settings

Use this page to enable or disable the application profiling service.

Applications that are configured to use the application profiling service will not start successfully unless the application profiling service is enabled.

To view this administrative console page, click **Servers** > **Application Servers** > *server_name* > **Application Profiling Service**.

Startup

Specifies whether the server will attempt to start the application profiling service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the application profiling service automatically.

Cleared
This option is unavailable. The application profiling service cannot be disabled.

Application profile collection

Use this page to manage application profiles.

An application profile is a set of policies that are to be applied during the execution of an enterprise bean and a set of tasks that are associated with that profile. Mapping tasks to application profiles will control which access intent policies are applied at run time for the units of work that correspond to a particular task.

To view this administrative console page, click **Applications > Applications >** *application_name* **> Application Profile**.

Name

The name of the application profile.

The name must be unique; multiple profiles cannot share the same name.

Data type
String

Description

A description of the application profile.

Data type
String

Application profile settings

Use this page to modify application profile settings.

To view this administrative console page, click **Applications > Applications >** *application_name* **> Application Profile >** *application_profile_name*.

Name: The name of the application profile.

The name must be unique; multiple profiles cannot share the same name.

Data type
String

Description: A description of the application profile.

Data type
String

Using the TaskNameManager interface

You can declaratively configure tasks on a J2EE component and associate tasks with EJB methods using the Application Assembly Tool. On rare occasions, you might find it necessary to *programatically* set the current task name. Application profiling supports this requirement with a simple interface that enables both overriding of the current task associated with the thread of execution, and resetting of the current task with the original task.

Application profiling does not support queries of the task that is in operation at run time. Instead, applications interact with logical task names that are declaratively configured as task references. Logical references enable the actual task name to be changed without having to recompile applications.

While you cannot programmatically set the current access intent policy, you can accomplish this by programmatically setting a task. Consider, for example, an entity bean with a single multi-object finder method, getLargeAccounts(), which is invoked by the single() method of the AccountManager session bean. By default, suppose that the entity bean runs assuming read intent; suppose, also, that an application profile has been configured under which that bean loads assuming update intent. Configure a task, perhaps with the name ″update″, on the profile. Now, depending on logic in the session bean's method, the session bean selectively and programmatically sets the task update before invoking the entity bean's finder method; that finder method correctly functions assuming updates.

Wherever possible, avoid setting tasks programmatically. The declarative method results in more portable function that can be easily adjusted without requiring redevelopment and recompilation.

Steps for this task

1. Configure task references.

   Application profiling requires that a task name reference be declared for any task that is to be set programmatically. Task name references introduce a level of indirection so that the actual task set at run time can be adjusted by reassembly without requiring recoding or recompilation. Any attempt to set a task name that is undeclared as a task reference results in the raising of an exception.

   a. Start the Application Assembly Tool.

   b. Create or edit the application EAR file.

      For example, to change attributes of an existing application, select **File > Open**, then select the EAR file.

   c. Select **Web Modules >** *module_name*.war **> Web Components >** *component_name*

   d. Select the **WAS Enterprise** tab.

   e. Click **Add** beside the **Task references** table.

   f. Provide a name for the task reference.

      The name must be unique for the component. This is the name that is referenced programatically. The name should be short and should be descriptive of the function that is performed when the task is executed.

   g. Provide a name and description for the task, itself.

      The name of the task is the identifier that is propagated on remote requests; it is the task name that is configured on application profiles to dynamically associate access intent hints with entity bean execution.

h. Click **OK**.

i. Select **File > Save** to save your configuration.

2. Perform a JNDI lookup on the TaskNameManager interface:

   ```
   InitialContext ic = new InitialContext();
   TaskNameManager tnManager = ic.lookup
   ("java:comp/websphere/AppProfile/TaskNameManager");
   ```

   The TaskNameManager interface is not bound into the namespace if the application profiling service is disabled.

3. Set the task name:

   ```
   try {
   tnManager.setTaskName("updateAccount");
   }
   catch (IllegalTaskNameException e) {
   // task name reference not configured. Handle error.
   }
   // . . .
   rnManager.resetTaskName();
   ```

   Resetting the task name undoes the effects of any setTaskName() method operations and reestablishes whatever task name was current when the component began execution. If the setTaskName() method has not been called, the resetTaskName() method has no effect.

What to do next

"Configuring tasks on application profiles" on page 134.

TaskNameManager interface

The TaskNameManager interface is available to all J2EE components using the following JNDI lookup:

```
java:comp/websphere/AppProfile/TaskNameManager
```

```
package com.ibm.websphere.appprofile;

/**
* The TaskNameManager is the programmatic interface
* to the application profiling function. Using this interface,
* programmers can set the current task name on the
* thread of execution. The task name must have been
* configured in the deployment descriptors as a task
* reference associated with a task. The set task
* name's scope is the duration of the method
* invocation in the EJB and Web components and for
* the duration of the client process, or until the
* resetTaskName() method is invoked.
*/
public interface TaskNameManager {

/**
* Set the thread's current task name to the specified
* parameter. The task name must have been configured as
* a task reference with a corresponding task or the
* IllegalTaskName exception is thrown.
*/
public void setTaskName(String taskName) throws IllegalTaskNameException;

/**
* Sets the thread's task name to the value that was set
* at, or imported into, the beginning of the method
* invocation (for EJB and Web components) or process
```

```
* (for J2EE clients).
*/
public void resetTaskName();

}
```

Chapter 6. Using Business Rule Beans

Before you begin

This topic provides a brief overview of the steps involved in externalizing Business Rule Beans. To gain an understanding of business rules and Business Rule Beans (BRBeans), review the topic "Overview of Business Rule Beans" on page 142. The following sections provide an overview of externalizing business rules using Business Rule Beans:

Steps for this task

1. "Developing BRBeans" on page 173
2. "Assembling applications for use with BRBeans" on page 178
3. "Managing rules" on page 179

Usage scenario

To help you get started quickly, a sample BRBeans application is provided. Samples are installed by default during a typical WebSphere installation or you can select to install specific samples during a custom installation.

The BRBeans sample is an online movie store application. The application's EAR file is installed and the application is configured to use the IBM Cloudscape database (IBM Cloudscape is provided with IBM WebSphere Application Server). A number of rules are created that you can view using the Rule Management Application (RMA). To do this, change to the `<install_root>/bin` directory and type one of the following commands:

- On a Microsoft® Windows® platform:
  ```
  rulemgmt ..\samples\lib\BRBeans\movieSampleProperties
  ```
- On a Unix platform:
  ```
  rulemgmt.sh ../samples/lib/BRBeans/movieSampleProperties
  ```

By running the sample, you can see how these rules are used. The source code for the sample also is provided in the `<install_root>/samples/src/BRBeans/Movie` directory. To see the use of trigger points, search the code for places where the trigger() method is used.

Advantages of externalizing business rules

Business Rule Beans (BRBeans) provide a framework in which business applications can externalize business rules. You can externalize rules by extending your application analysis and design processes to identify the points of variability (or "trigger points") in application behavior. When the application is implemented, the business logic required at the points of variability is externalized into a business rule. This allows certain aspects of the behavior to be changed without actually changing the application code.

Here are some advantages of externalizing business rules:

Explicit documentation of business practice decisions
Separating business rule values from the application code makes the code easier for others to view and understand while isolating information that relates to business practice decisions.

Clearer understanding of application behavior
Externalization makes it possible to inspect the application to see which business rules are being applied, when they are applied, and under what circumstances.

Reuse of rules across business processes
Separating rules from the business logic of the application makes it easy to reuse a business practice decision in a consistent fashion.

Increased consistency of business practices
Because externalized rules promote reuse and facilitate clear understanding of business practice decisions, they provide a basis for improving business practice consistency across applications.

Decreased maintenance and testing costs
Externalized rules have a clearly defined scope and are not tightly coupled to the application code. This makes them easy to modify, quick to test, and decreases costs and improves cycle time.

Improved manageability of business practice decisions
Externalization, change history, and inspectability all promote clear ownership and consequently a better definition of who can change rules and under what circumstances.

Increased confidence in predicting the business impact of proposed changes
Because rules are available for inspection, have well-defined scope, and are not tightly coupled to application business logic, they make it easy to understand the likely impact of changes and to predict whether contemplated modifications or additions will have unwanted ripple effects.

Ability to identify and correct conflicting business rules in different parts of the business
Externalized rules make it easy to check that rules being used in two different parts of an application or even two different applications dealing with different parts of the business, are consistent.

Overview of Business Rule Beans

Business Rule Beans are used to create and modify rules that keep pace with complex business practices. This enables your application's core behavior and user interface objects to remain intact and untouched, even as business practices change.

The Business Rule Beans (BRBeans) framework enables you to organize rules in folders. Folders provide a structure similar to the file system on your computer's hard drive. For example:

- Rules can be placed in folders based on any criteria you want.
- A rule folder can contain any number of rules and other rule folders.

In the BRBeans framework, each business rule is represented by an entity bean that persistently stores information related to that rule. Each business rule is assigned an appropriate rule name and stored in an appropriate rule folder (See "Rule folders" on page 145 for more information).

When naming rules and folders, adhere to the Java package naming convention. That is, name rules and folders based on the domain name of the organization for which the rules are developed. For example, ACME's `isSeniorCitizen` rule's fully qualified rule name ("full rule name"), might be `com/acme/ageRules/isSeniorCitizen`. In this example, the `com/acme` path is used by all of the rules developed by ACME and the `ageRules` folder is used to separate "age" rules from rules of other kinds. The root folder has no name; therefore, fully qualified path names never start with a forward slash ('/').

A fully qualified rule name consists of the following:
- The full path of the folder followed by a forward slash ('/')
- The name of the rule

This fully qualified rule name is used by a trigger point to identify the rule to trigger. Trigger points are small pieces of code that interface with the Business Rule Beans trigger point framework to run business rules during application execution. See "Placing a trigger point in the application code" on page 175 for more information.

By default, trigger points can only trigger rules that are currently in effect based on the current date and time when the trigger point is called. A business rule has a start date and an end date (see "Rule attributes" on page 145 for more information) that together define the interval during which the rule is in effect (see "Rule states" on page 147 for more information). This behavior can be overridden by specifying a date on the trigger point. This date is referred to as the "As Of Date". If no start date is specified, the rule is not valid and cannot be found by trigger points. Conversely, if no end date is specified, the rule never expires. Dates and times with a precision of one second can be assigned using the "Rule Management Application".

When there is more than one rule with the same fully qualified name, all of the rules with that name that are currently in effect are triggered and the results are combined using the combining strategy specified on the trigger point. See the "CombiningStrategy method" on page 159 for more information.

Externalized business rules

A business rule is a statement that defines or constrains some aspect of a business by asserting control over some behavior of that business.

A business rule officiates over frequently changing business practices and can come from within the company or be mandated from outside, typically by regulatory agencies. Typical uses for business rules include the following:
- Determining the current interest rate for a home loan
- Calculating a discount for a product
- Calculating the tax to apply to a given product
- Determining whether a given person is a senior citizen

The objects used to implement a business rule contain methods and attributes used by the Business Rule Beans (BRBeans) run-time environment, its administrative component, or both. An externalized business rule is implemented as a pair of objects:
- Rule
- RuleImplementor

The **Rule** is an entity enterprise bean that stores all of the persistent data for the business rule. This is the object that the trigger point framework code actually deals with directly. When a trigger point is invoked, the internal framework code performs a query to find the Rule object or objects representing the business rules to be triggered. Once the Rules are found, the framework code determines where the Rule is invoked, either local to the trigger point or remotely on the application server. Then, it invokes the `fire` method on either the Rule enterprise bean itself (for remote triggering) or on a local copy of the enterprise bean (for local triggering) to perform the function of the business rule.

The class name of the business rule's **`RuleImplementor`** is stored persistently in the Rule. The `RuleImplementor` is a transient object (not managed by the application server) that the Rule instantiates and then uses to do the actual work. When the `fire()` method is called on the Rule object, the Rule object combines its persistent set of values with the parameters it received on invocation. This creates the parameter list for the `RuleImplementor` prior to invoking `fire()` on the `RuleImplementor` with this parameter list. The actual execution of the `RuleImplementor` algorithm can take place either remotely (within the application server where the BRBeans enterprise beans are installed) or locally (within the Java virtual machine (JVM) where the trigger point was called).

Types of business rules

Business rules can be divided into the following types:

- Base rules
- Classifier rules

Base rules are the most common type of rule and are triggered by the `TriggerPoint.trigger` method. You can divide Base rules into the following categories:

Derivation rules
These rules use an algorithm to return a value. These rules return any type of value that makes sense in the business context in which they are used. For example, a derivation rule can calculate a discount or compute the total price of an order.

Constraint rules
These rules confirm that an operation has met all of its obligations and that a particular constraint or edit has been met. For instance, a constraint rule can check that a value entered by an external user is within legal bounds. Business Rule Beans (BRBeans) provide a special return type, `com.ibm.websphere.brb.ConstraintReturn`, which can be returned by a constraint-type rule. A `ConstraintReturn` object contains a boolean value so that if it is false, it can contain information that can be used to produce an external message explaining what constraint was not met.

Invariant rules
These rules ensure that multiple changes made by an operation are properly related to one another.

Script rules
These rules implement "micro-workflow" or electronic performance support. They are small, variable pieces of a business process that provide assistance to end-users to get the most from the application.

On the surface, **classifier rules** are much like base rules. However, classifier rules can be used to determine the ways in which variables are classified by a business. Classifier rules are triggered by the `TriggerPoint.triggerClassifier` method.

A classifier rule is used to compute a classification for a particular business situation. The classification returned is required to be of type `string`. For instance, bank customers may be classified into gold, silver, and bronze categories based on their spending history or the amount of money they have in their account. For more information on this type of rule, refer to "Situational trigger point" on page 155.

Rule folders

Rule folders are similar to the directories that divide a computer's hard drive in that they split a large number of files into conceptual units. The rule folder adds its path to the fully qualified rule name. Like the directories on a hard drive, a rule folder can contain any number of rules or rule folders.

Although you can name the folders whatever you deem appropriate, it is recommended that you follow the Java package naming convention. That is, base the names on the domain name of the organization where the rules are developed. So, the fully qualified rule name or full rule name of ACME's `isSeniorCitizen` might be `com/acme/ageRules/isSeniorCitizen`. In this example, the `com/acme` path is used by all of the rules developed by ACME and the `ageRules` folder is used to separate ″age″ rules from rules of other kinds.

Note: The root folder has no name, which means that fully qualified path names never start with a '/'.

When using the Rule Management APIs, a rule folder contains instances of `IRules`, which also are referred to as ″rules″. To begin working with rules, get the root rule folder by using the `getRootFolder` method on `RuleMgmtHelper` class. From the root rule folder you can add, delete, and retrieve folders and rules using methods on this interface.

Rule attributes

Rule name
A name for the rule that is appropriate to its business context.

Rule folder
The folder that contains the rule.

Start date
This is the date and time at which the rule goes into effect. Prior to this time, it will not be found by trigger points. Together with the end date, the start date defines a period of time during which the rule is effective. A rule that does not have a start date specified is not a valid rule and will not be found by trigger points.

End date
This is the date and time at which the rule is no longer effective. After this date and time the rule is no longer in effect and will not be found by trigger points. Together with the start date, the end date defines a period of time during which the rule is effective. A rule that does not have an end date specified is valid and will never expire.

Ready This indicates whether the rule is ready to be used. Rules that are not marked as ready will not be found by trigger points. This is intended to be an easy way to keep a rule from being used until it is completely defined or to temporarily turn a rule off without having to change the basic rule data such as start and end dates.

Java Rule Implementor name
This is the fully package-qualified name of a Java class that implements the BRBeans RuleImplementor interface. The `fire` method of the class performs the function of the rule. Business Rule Beans (BRBeans) provide several predefined rule implementors or you can write your own. See "Rule implementors" on page 151 or "Customized rule implementors" on page 161 for more information.

Initialization parameters
This is an array of parameters that are passed to the rule implementor to initialize it. Each element in the array can be any object. This also can be referred to as the rule data, which is the external data that may change over time. The initialization parameters defined for a rule are passed directly to the `init` method of the rule implementor when it is instantiated. See "Rule Implementors" for more information on how rule implementors can use initialization parameters.

Firing parameters
Normally, firing parameters are simply the parameters passed on the trigger point when a rule is triggered. However, it is allowed to override these parameters by specifying parameters on the rule itself. This is where these overriding parameters are specified.

Firing location
This specifies where the rule implementor for this rule is instantiated and run. The following values are allowed:

Local This option instantiates the rule implementor and runs it local to the trigger point (in the same JVM as the trigger point call). This is run on the client machine if the trigger point call is done there or on the server if the server part of an application makes a trigger point call. Use this option for the best performance since, once a rule is cached on the client, the entire triggering process can be performed locally without going to the server at all. The main disadvantage of this option is that the class files for the rule implementors need to be available on every client that can trigger rules.

Remote
This will instantiate the rule implementor and run it on the application server where the Business Rule Beans enterprise beans are installed. When using this option at least one remote method call always is required to trigger a rule since the trigger takes place on the server. The advantage is that the rule implementor class files only need to be available on the server.

Anywhere
This option tries to instantiate and run the rule implementor locally, and, if the class cannot be found, it tries to trigger it remotely.

Classification
For classified rules, this is the classification to which the rule applies. This is used when you use a situational trigger. Once a classification is computed for the situational trigger point, rules that apply to that

classification are found and triggered. For more information, see "Situational trigger point" on page 155.

Classifier
: This indicates whether this rule computes a classification. Classification is used for a situational trigger. A classifier rule is used to perform the first step of a situational trigger which computes a classification that is used to find rules to deal with the situation. For more information, see "Situational trigger point".

Dependent rules
: In many cases, a rule triggers other rules to complete the overall task. These other rules are referred to as dependent rules and can be specified using the dependent rules attribute. For more information, see "Dependent rules" on page 148.

Business intent
: This is a text description of the intent of this rule from the view point of the business analyst. You can store any text string here.

Description
: This is a text description of the rule at the programmer's level. You can store any text string here.

Original requirement
: This is a text description of the initial business analyst requirement of this rule. You can use this description to keep track of why this rule was originally created (for example, to keep auditing records). You can store any text string here.

User-defined data
: You can store a user-defined text string here. The format and use of this data is completely determined by the user.

Primary key
: Every rule has a primary key to uniquely identify it in the database where the enterprise beans are stored. Normally, a unique primary key is generated automatically when you create a new rule. However, you can use the rule management APIs to specify your own primary key, if desired. See (Rule management APIs) for more information.

Precedence
: This is the relative priority of this rule. The default finding strategy uses this value to order the rules found in the database, from lowest to highest, when more than one rule is found for a particular trigger point. Rules are sorted numerically by precedence with the numerically lowest precedence first and the numerically highest precedence last.

Rule states

Rules can be in any one of the following states at any particular time:

scheduled
: The rule is scheduled to become effective (its start date is in the future) and will not be found by current trigger points.

in effect
: The rule is currently in effect and can be found by trigger points.

expired
: The rule is no longer in effect (the end date is in the past) and will not be found by trigger points.

invalid
: The rule is not correctly defined and will not be found by trigger points.

Typically, only those rules that are ″in effect″ are found by the Business Rule Beans (BRBeans) run-time environment. This behavior can be overridden by setting an `asOfDate` on the TriggerPoint object, which then will execute ″as if″ the current date is the given date. For more information, see ″As Of Date″.

When a Rule is first created, it is marked as ″ready for use″ and is found when firing Rules. If the Rule is not complete and you do not want it to be found by BRBeans, then use either of the following to mark the Rule:

- Use the `setReady(false)` method in the Rule Management APIs
- Use the Rule Management Application to mark the rule as not ready

Rule results

In general, a rule can return any type of result that makes sense for the business purpose of the rule. The return type on the fire() method is `java.lang.Object` so any Java object can be returned, including arrays. You cannot return a Java primitive since the results must be an object. However, you can return the object form of the primitives. For example, you can return a `java.lang.Integer` instead of an `int`. If the rule is fired remotely, the returned value must implement `java.io.Serializable`.

Dependent rules

When a business rule triggers other business rules as part of its implementation, the rules that are triggered are called **dependent rules** of the first rule. An example is the `RuleAND` rule implementor supplied with Business Rule Beans (BRBeans). It uses two or more dependent rules, each of which is assumed to return a true or false value. When a rule with `RuleAND` as its implementor is triggered, it triggers each of its dependent rules and a logical AND operation is performed on all of the returned results. The result of this AND operation is returned as the result of the top-level rule.

Dependent rules are specified in the attributes of the top-level rule where the fully qualified name of each dependent rule is listed. When the top-level rule is triggered, an array of dependent rule names is passed to the rule implementor’s init() method. They are stored here until they are triggered by the fire() method.

Note: The BRBeans framework does not ensure that the dependent rules specified in the enterprise beans are actually triggered. Triggering the dependent rules and interpreting their results is entirely up to the rule implementor of the top-level rule.

Dependent rules can be nested within other dependent rules. In other words, a dependent rule of some particular rule can have its own dependent rules which, in turn, can have their own dependent rule and so on. The BRBeans framework does not place any restriction on the number of levels that dependent rules can be nested. The only practical restriction is the complexity of the rule set that is built up when dependent rules are nested many levels deep.

BRBeans run-time environment

The Business Rule Beans (BRBeans) run-time environment is used to find and trigger rules.

The BRBeans run-time environment is made up of two parts:

- Code that runs on the **client** (″client″ here meaning wherever the trigger point is located). This consists of code that does the following:
  - Finds the specified rules
  - Decides where the rules should be triggered
  - Calls the fire method on all rules
  - Combines the results from the rules
- Code that runs on the **server**. This consists of enterprise beans used to represent rules and rule folders. These enterprise beans do the following:
  - Provides for business rule persistence
  - Provides query functions that the client part of the run time can use to find rules to be triggered

BRBeans run-time behavior

Business Rule Beans (BRBeans) run-time behavior can best be described by giving a simple example of a trigger point selecting, executing, and then responding to the results of a business rule.

The first step in triggering a rule is for the trigger point framework to perform a query on the rule server to determine which rules to trigger. The main item used for the query is the fully qualified rule name. Other items used in the query include the start and end date, whether this is a classifier, the classification of the rule, and whether the rule is marked ″ready″. This query returns zero or more rules. If there is at least one rule, the trigger point assembles the data that is sent as parameters to each rule. The trigger point then loops through the list of rules invoking the fire() method on each and passing the parameters. The results are combined depending on the combining strategy used.

When the trigger point framework invokes fire on a rule, it instantiates the `RuleImplementor` and uses it to do the actual work (to execute the rule algorithm or test). Once it has arrived at a result, the `RuleImplementor` returns that result. For constraint rules (ones that arrive at a boolean true or false answer) the returned value is, by convention, a ConstraintReturn. A ConstraintReturn is a data structure that indicates whether the constraint was satisfied. If not, the ConstraintReturn indicates what went wrong. For derivation rules (ones that calculate a single, generally non-boolean value), the return value can be any type. In the simplest case, the return value from each `RuleImplementor` is returned back to the trigger point where it is analyzed to determine what action to take.

The following is an overview of what happens when the `maxTruckDriverHours` rule is triggered:

A rule exists named `maxTruckDriverHours`. The purpose of this rule is to check that the number of hours entered by a user for a particular truck driver does not exceed the maximum allowed value. This rule contains an initialization parameter list consisting of a single value of 8. This rule is bound to a `RuleImplementor` class called `MaxRuleImpl`. `MaxRuleImpl` tests the parameter it is passed against the initialization list value and returns a `ConstraintReturn`. The `ConstraintReturn` is set to true if the passed parameter is less than or equal to the initialization value. Otherwise, a `ConstraintReturn` is set to false and some information is added that describes which values were compared and why the test failed.

When this rule is triggered, the following details the trigger point process:

1. During the execution of the application, the application reaches a point where it needs to verify that the number of truck driver hours that was entered is valid. The application code invokes a simple trigger point passing the name of the rule to be triggered and a parameter list containing the entered hours for the driver.
2. The trigger point framework performs a query on the rule server to find the rule with the specified name. It receives back a sequence of rule objects. In this case, this sequence contains one rule, `maxTruckDriverHours`.
3. The framework determines whether this rule is to be triggered on a local or remote machine. If local, the framework gets a local copy of the rule object and calls the fire method on the copy. If remote, the framework calls the fire method on the enterprise bean reference. The parameter list containing the weight is passed to the fire method.
4. The `maxTruckDriverHours` rule (either the copy or the enterprise bean itself) creates an instance of the rule implementor class, `maxRuleImpl`, if it does not already have one. When a new rule implementor instance is created, the rule calls its init method passing any initialization parameters defined for the rule. In this case, the initialization parameter list contains the single value 8. If the rule already has a rule implementor instance, it uses that one and does not call the init method.
5. The `maxTruckDriverHours` rule calls the fire method on the rule implementor instance. The firing parameters passed to the trigger point are passed to the rule implementor and are possibly modified by any firing parameters defined in the rule itself. In this case, the firing parameters are passed directly from the trigger point.
6. The `maxRuleImpl` returns a `ConstraintReturn` object to the rule that indicates the result of its comparison. This `ConstraintReturn` is returned to the trigger point framework and ultimately to the application.
7. The application checks the value in the `ConstraintReturn` and takes the appropriate action.

BRBeans run-time exception handling

Business Rule Beans (BRBeans) defines one general exception class for exceptions that might be exposed to the user. All other BRBeans exceptions inherit from this class. The name of this class is `com.ibm.websphere.brb.BusinessRuleBeansException`. A `BusinessRuleBeansException` is generally thrown when an unexpected error occurs within BRBeans. A `BusinessRuleBeansException` might have information in it about the original exception that caused the error. Doing a `printStackTrace` on the `BusinessRuleBeansException` prints out this information and the stack trace for the `BusinessRuleBeansException` itself. Also, there are methods to access the original exception programmatically, if desired.

BRBeans also defines a **`ConstraintViolationException`** , which extends `BusinessRuleBeansException`. A `ConstraintViolationException` is thrown if the `ThrowViolationCombiningStrategy` is specified on the `TriggerPoint` and the rule returns a false value (either a `ConstraintReturn` or a boolean).

Finally, BRBeans defines two exceptions, **`NoRuleFoundException`** and **`MultipleRulesFoundException`** , that are thrown by some of the predefined filtering strategies if an unexpected number of rules is found on a trigger point call. These two exceptions both extend `UnexpectedRulesFoundException` which, in turn, extends `BusinessRuleBeansException`.

Rule implementors

A rule implementor, in terms of Business Rule Beans, is an algorithm written in Java that implements the BRBeans `RuleImplementor` interface.

A Business Rule Beans (BRBeans) Rule is a persistent object that exists on the BRBeans Rule server. One of the rule's persistent attributes, in addition to startDate, endDate, initParams, and so on, is `javaRuleImplementorName`, which is the name of its rule implementor.

BRBeans supplies a number of predefined rule implementor classes that can be used in user-defined BRBeans rules (see the BRBeans Javadoc in the InfoCenter for more information) to implement the BRBeans `RuleImplementor` Interface. The Java source code for these rule implementors is supplied as BRBeans sample code in the `com.ibm.websphere.brb.implementor` package. This sample code, packaged in a JAR file, appears in the CLASSPATH of the BRBeans Rule Server (for "remote" firing) or is co-located in the CLASSPATH of the application or applications using it (for "local" firing). Typically, the `RuleImplementor` is in the application EAR file.

Using standard Java development tools, you can externalize BRBeans by attaching them to either enterprise beans or ordinary Java objects. Programming a new rule implementation in Java is typically a simple process. If you write your own rule implementor, you must create a new Java class that implements the `com.ibm.websphere.brb.RuleImplementor` interface. This class must implement the following methods:

Default constructor
The class must have a default, no-argument constructor so that it can be instantiated when a rule using it is triggered.

init The init method comes from the `RuleImplementor` interface and is called when the rule implementor is first. Its purpose is to perform an initialization needed by the rule implementor instance before it is actually fired. The following parameters are passed to the `init` method:

The initialization parameters defined for the rule being triggered
These can be any parameters needed to properly initialize the rule implementor instance. Often the initialization parameters consist of constants required by the algorithm. For example, when using a rule implementor that checks whether a number is greater than a threshold value, the threshold value normally is passed as an initialization parameter. This parameter is null if there are no initialization parameters for the rule.

An array of names of dependent rules for the rule being triggered
Normally, the rule implementor stores these names to be used when the `fire` method is called. These dependent rules are intended to be triggered as part of the algorithm performed by the rule implementor. See "Dependent rules" on page 148 for more information. This parameter is null if there are no dependent rules defined for the rule.

The user-defined data for the rule being triggered
This data is completely defined by the user of the Business Rule Beans (BRBeans). BRBeans does not interpret this data in any way. This parameter is null if there is no user-defined data defined for the rule.

A reference to the actual rule being triggered
This can be used to extract attribute values from the rule, if needed.

fire The fire method comes from the `RuleImplementor` interface. This method is called to perform the algorithm of the rule implementor. Any desired algorithm can be performed here. Normally, a value is returned by the `fire` method that is ultimately returned as the result of triggering the rule. The following parameters are passed to the fire method:

The TriggerPoint object that is being used to trigger the rule
This parameter is needed if the rule has dependent rules that the fire method needs to trigger.

The target object for this particular trigger call
This parameter can be any object that is thought of as the target of the rule. However, the parameter can be null.

A reference to the actual rule being triggered
This parameter can be used to extract attribute values from the rule, if needed.

The firing parameters for this particular trigger call
Normally, these parameters are the firing parameters passed by the code that invoke the trigger point. However, these can be overridden by specifying firing parameters on the rule itself. Wherever they ultimately come from, these are the parameters that the rule implementor needs at run time to perform its function. Normally, these are run-time variables that are to be processed by the rule implementor. For example, when using a rule implementor that checks whether a number is greater than a threshold value, the number to be checked normally is passed as a firing parameter. This parameter is null if no firing parameters are passed by the caller and none are defined on the rule itself.

getDescription
`getDescription` comes from the `RuleImplementor` interface. The purpose of this method is to return a text string that describes the function of the rule implementor. This information might be displayed on a user interface to help select what implementor to use. This method, however, is currently not used by the BRBeans framework. Users can incorporate this information if they create their own rule implementor. For additional information on the RuleImplementor interface, see the BRBeans Javadoc in the InfoCenter.

Trigger point framework

A trigger point is the location in a method of an object where externalized business rules are invoked.

The proper placement of trigger points can add substantially to the flexibility and speed with which a business application adapts to new business practices.

Wherever a trigger point is placed in user-written code, the Business Rule Beans (BRBeans) trigger point framework needs to do the following:

1. Assemble the parameter list to send to the rules
2. Find the potential rules that apply
3. Filter out any rules which do not apply (optional)

4. Fire the rules in the filtered rule set
5. Combine the results of the rule firings is some meaningful way

The application code that contains the trigger point needs to perform the following functions:

1. Establish a value for the target object. Usually the target object is the object in which the trigger point is encountered. The target object is one of the parameters passed to the fire method of the `RuleImplementor`. However, this is an optional parameter. If the rule implementor does require a target object, null can be passed instead.
2. Build the array of objects containing the run-time parameters needed to satisfy the trigger point's business purpose. This array is normally passed as one of the parameters of the `fire` method of the `RuleImplementor`. If firing parameters are specified on the rule itself, then those firing parameters are passed instead of the ones passed by the caller.
3. Invoke the `trigger()`, `triggerClassifier()`, or `triggerSituational()` method of the `TriggerPoint` class.
4. Catch and handle any exceptions that might occur as a result of firing the rules. Otherwise, take action based upon the rule firing results.

The two simple trigger methods, `trigger` and `triggerClassifier`, perform their function in four steps:

1. Find the rules
2. Filter out those rules which are not desired
3. Fire the remaining rules
4. Combine the results and return to the caller

The complex trigger method, `triggerSituational` does this sequence of steps twice. In the first phase, the method performs the four steps once to find a rule that returns a classification. This classification is fed into the second phase. The second phase triggers rules that have the name specified in the triggerSituational method and have a classification equal to the value returned by the first phase.

How each of these steps is performed can be modified through various methods on the TriggerPoint object. The implementation of each step is defined by a **strategy** object. For more information on strategies, see Administering strategy objects to control triggers.

Trigger points

Examples of how to code a trigger point call are provided in the following topics:

- "Simple trigger point"
- "Classifier trigger point" on page 154
- "Situational trigger point" on page 155

Simple trigger point

A simple trigger point is used to trigger a rule or rules specified by name. This type of trigger point is used by invoking the trigger method on an instance of the `TriggerPoint` class. All rules with the specified name are triggered and the results are combined using the `CombiningStrategy` specified on the `TriggerPoint` object. This type of trigger point only finds rules that are not marked as classifiers.

The following shows an example of using a simple trigger point to trigger a rule named `isSeniorCitizen` (in the `com/acme/ageRules` folder), which determines whether a person is classified as a senior citizen based on the passed in age:

```
 ...
// create an instance of TriggerPoint for triggering the rule and specify that the
// ReturnFirstCombiningStrategy is to be used to return only the first result if
// multiple rules are found.
TriggerPoint tp = new TriggerPoint();
tp.setCombiningStrategy(CombiningStrategy.RETURN_FIRST, TriggerPoint.ALL_RULES);

// define parameter list that's passed to the rule
Object [ ] plist = new Object[1];

// define age of person to be tested
Integer age = new Integer(64);

// define name of rule to be fired
String ruleName = "com/acme/ageRules/isSeniorCitizen";

// define result of rule firing
Object result = null;

// initialize parameter list
plist[0] = age;

try {

 // fire "com/acme/ageRules/isSeniorCitizen" rule passing parameter list containing age.
 // Note: in this case the target object is not used and could be null.
 result = tp.trigger(this, plist, ruleName);

 // put result into usable format.  A single result is returned since we specified to use
 // the ReturnFirstCombiningStrategy.  By default an array of results would be returned.
 boolean seniorCitizen = ((Boolean)result).booleanValue();

 // make use of result
 if( seniorCitizen ) {
  ...
 }

}
catch(BusinessRuleBeansException e ) {

 // handle exception
 ...

}
```

Classifier trigger point

A classifier trigger point is identical to a simple trigger point except that it only finds rules marked as classifiers. Classifiers are rules that determine what sort of business situation is present. These rules then return a classification string that indicates the result.

Usually these rules are used as part of a situational trigger point, but they also can be triggered on their own. This type of trigger point is used by invoking the `triggerClassifier` method on an instance of the `TriggerPoint` class.

The following shows an example of using a classifier trigger point to trigger a rule named `determineCustomerLevel` (in folder `com/acme/customerClassfiers`). This rule classifies customers into levels (gold, silver, and bronze) based on their spending history.

```
 ...
// create an instance of TriggerPoint for triggering the rule and specify that the
// ReturnFirstCombiningStrategy is to be used to return only the first result if
// multiple rules are found.
TriggerPoint tp = new TriggerPoint();
tp.setCombiningStrategy(CombiningStrategy.RETURN_FIRST, TriggerPoint.ALL_RULES);

// define parameter list that's passed to the rule
Object [ ] plist = new Object[1];
```

```
// information about the customer to be checked is stored in this object
Customer cust = ...;

// define name of rule to be fired
String ruleName = "com/acme/customerClassifiers/determineCustomerLevel";

// define result of rule firing
Object result = null;

// initialize parameter list
plist[0] = cust;

try {

 // fire "com/acme/customerClassifiers/determineCustomerLevel" rule passing parameter
 // list containing the customer to be checked.
 // Note: in this case the target object is not used and could be null.
 result = tp.triggerClassifier(this, plist, ruleName);

 // put result into usable format.  A single result is returned since we specified to use
 // the ReturnFirstCombiningStrategy.  By default an array of results would be returned.
 String customerLevel = (String) result;

 // make use of result
 if( customerLevel.equals("Gold") ) {
  ...
 } else if ( customerLevel.equals("Silver") ) {
  ...
 } else if ( customerLevel.equals("Bronze") ) {
  ...
 } else {
  ...
 }
}
catch(BusinessRuleBeansException e ) {

 // handle exception
 ...

}
```

Situational trigger point

A situational trigger point is used when the rule or rules to be triggered depend on the business situation.

This example evaluates a customer's past purchasing history to place them into one of three levels: Gold, Silver, or Bronze. Their classification determines how much of a discount they receive.

To use a situational trigger point to handle this case, it is first necessary to define four rules:

- one **classifier** rule to determine under which of the three levels to classify the customer
- three **classified** rules to determine the actual discount to offer

All of the classified rules have the same name and are marked as applying to one of the three customer levels by specifying the level in its classification attribute. For example, the rule to determine the discount for a Gold level customer will contain the string ″Gold″ in its classification attribute.

The situational trigger point takes two rule names as input: the name of the classifier rule and the name of the classified rule. The situational trigger point then proceeds in two phases:

1. Find the specified classifier rule and trigger it to generate a classification string.

2. Find the rules that have the name specified for the classified rule and have a classification attribute equal to the classification string returned by the first phase.

These rules then are triggered to produce the final result, in this case the discount to offer.

The following shows an example of a situational trigger point used to handle the case described previously:

```
...
// create an instance of TriggerPoint for triggering the rule and specify that the
// ReturnFirstCombiningStrategy is to be used to return only the first result if
// multiple rules are found.
TriggerPoint tp = new TriggerPoint();
tp.setCombiningStrategy(CombiningStrategy.RETURN_FIRST, TriggerPoint.ALL_RULES);

// define parameter list that's passed to the classifier rule
Object [ ] classifierPlist = new Object[1];

// define parameter list that's passed to the classified rule
Object [ ] classifiedPlist = new Object[1];

// information about the customer to be checked is stored in this object
Customer cust = ...;

// define name of classifier rule to be fired
String classifierRuleName = "com/acme/customerClassifiers/determineCustomerLevel";

// define name of classified rule to be fired
String classifiedRuleName = "com/acme/discountRules/determineDiscount";

// define result of rule firing
Object result = null;

// initialize parameter lists
classifierPlist[0] = cust;
classifiedPlist[0] = cust;

try {
// fire the rules to get the discount to offer
// Note: in this case the target object is not used and could be null.
result = tp.triggerSituational(this, classifiedPlist, classifierPlist,
classifiedRuleName, classifierRuleName);

// put result into usable format.  A single result is returned since we specified to use
// the ReturnFirstCombiningStrategy.  By default an array of results would be returned.
Float discountToOffer = (Float) result;

// make use of result
 ...
}
catch(BusinessRuleBeansException e ) {

// handle exception
 ...

}
```

Note: In the preceeding example, the code that following `try {` wrapped onto two lines due to the width of the page. The following line of code is normally typed on one continuous line:

```
result = tp.triggerSituational(this, classifiedPlist, classifierPlist,
classifiedRuleName, classifierRuleName);
```

As Of Date

An ″As Of Date″ can cause rules to be triggered as if the given date is the current date. This is especially useful when you want to test a rule, see what effect a future change in rules or regulations may have on the overall framework, or see what past or future rates, discounts, or both might be.

Normally, a rule only can be triggered if it is "in effect" (see "Rule States") as of the current date and time.

To set an "As Of Date", call the setAsOfDate() method on the `TriggerPoint` object and pass the date that you want to be used. To use the current date again, call `unsetAsOfDate` or `setAsOfDate` and pass null for the date.

Predefined strategy objects

The following is a list of predefined strategy objects that are provided in Business Rule Beans:

FindingStrategy
Accesses the data store and returns those rules that meet the search criteria specified

FilteringStrategy
Takes the list of rules that were found by the FindingStrategy and filters out the rules that should not be fired

FiringStrategy
Takes the rules that were found by the FindingStrategy, (possibly modified by the FilteringStrategy), fires them each in order, and returns an array containing the results of each rule

CombiningStrategy
Takes the results of the rules that are fired by the `FiringStrategy` and combines them to form a reasonable result to the `TriggerPoint` caller.

FindingStrategy method

The job of the `FindingStrategy` is to access the data store and return those rules that meet the search criteria specified. There are two `FindingStrategy` classes provided by Business Rule Beans (BRBeans):

- `DefaultClassifierFindingStrategy`
- `DefaultNonClassifierFindingStrategy`

Both of these strategies perform a case-sensitive search for Rules that are marked "ready" and match the given search criteria. Results are ordered by precedence from highest to lowest (the first rule in the array has the numerically smallest precedence, the next rule has the next smallest precedence, and so on). If no rules are found, then an empty array is returned. The former strategy returns classifier rules (`classifier=true`) only and the latter returns non-classifier rules (`classifier=false`) only.

These default strategies are used automatically by the `TriggerPoint`. There is no need to call `setFindingStrategy` to use these strategies. Instances of these two default finding strategies are stored in static constants defined on the `FindingStrategy` interface.

FilteringStrategy method

The job of the `FilteringStrategy` is to take the list of rules that were found by the `FindingStrategy` and filter out the rules that should not be fired. There are three sets of filtering strategies used in `TriggerPoint:`

- strategy for **zero** rules found
- strategy for **one** rule found
- strategy for **multiple** rules found

A different strategy can be used for each of these scenarios, along with different strategies for classifier and non-classifier rules. The zero rules strategy is invoked if no rules are found by the finding strategy, the one rule strategy is invoked if exactly one rule is found and the multiple rules strategy is invoked if more than one rule is found.

Business Rule Beans (BRBeans) provides the following filtering strategies that can be used:

Accept Any
BRBeans utilizes all of the rules found (this is the default).

Accept One
BRBeans expects one rule only.

Accept First
BRBeans utilizes the first rule found.

Accept Last
BRBeans utilizes the last rule found.

Instances of these filtering strategies are stored in static constants defined in the `FilteringStrategy` interface. You can use these for setting the strategies on a `TriggerPoint`.

As an example, here is one common way to use filtering strategies. You want to ensure that exactly one rule is found on a `TriggerPoint` call. Thus, set all three strategies (zero rules, one rule, and multiple rules) for this `TriggerPoint` to `FilteringStrategy.ACCEPT_ONE`. This strategy throws an exception if the number of rules is not exactly one. The following sequence of method calls accomplishes this for `TriggerPoint tp`:

```
tp.setNoRulesFilteringStrategy(FilteringStrategy.ACCEPT_ONE, TriggerPoint.ALL_RULES);
tp.setOneRuleFilteringStrategy(FilteringStrategy.ACCEPT_ONE, TriggerPoint.ALL_RULES);
tp.setMultipleRulesFilteringStrategy(FilteringStrategy.ACCEPT_ONE, TriggerPoint.ALL_RULES);
```

FiringStrategy method

The FiringStrategy takes the rules that were found by the `FindingStrategy`, (possibly modified by the `FilteringStrategy`), fires them each in order, and returns an array containing the results of each rule.

A single default `FiringStrategy` is provided by Business Rule Beans (BRBeans) as all of the rules are fired in the same way. This implementation takes each rule in order and performs the following steps:

1. Determines what firing parameters to pass to the rule. If there are no firing parameters specified for this rule, the implementation uses the firing parameters passed on the `TriggerPoint` call. Otherwise, it uses the firing parameters specified in the rule in place of the parameters passed on the `TriggerPoint` call.
2. Calls the fire method on the rule and passes the firing parameters from the first step.

Unexpected exceptions result in a `BusinessRuleBeansException` being thrown that contains the original exception.

CombiningStrategy method

The job of the `CombiningStrategy` is to take the results of the rules that are fired by the `FiringStrategy` and combine them to form a reasonable result to the `TriggerPoint` caller. Business Rule Beans (BRBeans) provides several combining strategies to be used in applications:

Return All
: Returns the results from all of the rules fired in an array (this is the default)

Return First
: Returns only the result from the first rule fired

Return Last
: Returns only the result from the last rule fired

Return AND
: Returns the logical AND of the results from all the rules fired. This strategy requires that all of the results returned by the fired rules are either `ConstraintReturn` objects or `java.lang.Boolean` objects. An exception is thrown if this is not the case.

Return OR
: Returns the logical OR of the results from all of the rules fired. This strategy requires that all of the results returned by the fired rules are either `ConstraintReturn` objects or `java.lang.Boolean` objects. An exception is thrown if this is not the case.

Throw Violation
: Throws a `ConstraintViolationException` containing all of the failed `ConstraintReturn` objects if any `ConstraintReturns` contain false. Otherwise, it returns a true `ConstraintReturn`.

Instances of these combining strategies are stored in static constants defined in the `CombiningStrategy` interface. You can use these for setting the strategies on a `TriggerPoint`. For example, the following method call sets the `CombiningStrategy` on `TriggerPoint` `tp` to be the Return_First strategy:

```
tp.setCombiningStrategy(CombiningStrategy.RETURN_FIRST, TriggerPoint.ALL_RULES);
```

Customized strategy objects

The process of triggering a rule or set of rules is controlled by a set of strategy objects. The following four strategies are used each time a rule is triggered:

FindingStrategy
: The `FindingStrategy` accesses the persistent data store to find the set of rules matching the search criteria passed to the trigger call. The search criteria are based on the rule ID information passed on the trigger call. The set of rules found is passed to the `FilteringStrategy`.

FilteringStrategy
: The `FilteringStrategy` can change the set of rules that were found by the `FindingStrategy`. The set of rules returned is the set that are fired by the `FiringStrategy`.

FiringStrategy
: The `FiringStrategy` fires the rules found by the `FindingStrategy`, which may be modified by the `FilteringStrategy`. It gathers the results of the individual rules and passes them to the `CombiningStrategy`.

CombiningStrategy
The `CombiningStrategy` takes the results from firing the rules and combines them to produce the final result of the trigger.

Each `TriggerPoint` object has its own set of strategies that can be changed independent of any other `TriggerPoint` object. There is a set of default strategies that are used by the `TriggerPoint` if none are explicitly set.

For each of the four strategies, you can set different strategies for **classifier** rules and for **non-classifier** rules. The strategies set for classifier rules are used when the Business Rule Beans (BRBeans) framework is triggering a classifier rule. The strategies for non-classifier rules are used in all other cases.

It is also possible to set three different sets of filtering strategies:
- one to be used if no rules are found
- one to be used if exactly one rule is found
- one to be used if more than one rule is found

This capability can be used to set up filtering strategies that throw exceptions if the expected number of rules is not found.

Strategy classes must implement one of the strategy interfaces provided by BRBeans in the `com.ibm.websphere.brb` package:
- `FindingStrategy`
- `FilteringStrategy`
- `FiringStrategy`
- `CombiningStrategy`

Users can write their own strategy implementations to perform special functions not performed by the predefined implementations. Write these strategy implementations with care since part of the functionality of the BRBeans framework is replaced when you write a custom strategy. One simple example of writing a custom strategy is creating a new firing strategy that logs every rule that is fired.

The basic requirement for a strategy implementation is that it implements the appropriate strategy interface.

For the **filtering** and **combining** strategies, create a class that implements either `FilteringStrategy` or `CombiningStrategy` and either the filterRules() method (for `FilteringStrategy`) or the combineResults() method (for `CombiningStrategy`) to perform the required functions. At run time, create an instance of the new class and pass it to the `TriggerPoint` object using the appropriate set method so that the new strategy is used when rules are triggered using that `TriggerPoint`.

The **finding** and **firing** strategies are more complicated to customize since they provide more function than the simple filtering and combining strategies. Default finding and firing strategy implementations are provided that define a general outline of the steps necessary to perform the function. It is suggested that you subclass these when you customize your own strategies and then override the desired methods on the default implementation to provide the new behavior.

The BRBeans Javadoc in the InfoCenter provides more information about the FindingStrategy, FilteringStrategy, FiringStrategy, and the CombiningStragegy interfaces.

Customized rule implementors

To write your own rule implementor, create a new Java class that implements the `com.ibm.websphere.brb.RuleImplementor` interface. This class must implement the following methods:

Default constructor
: The class must have a default, no-argument constructor so that it can be instantiated when a rule using it is triggered.

init
: The init method comes from the `RuleImplementor` interface and is called when the rule implementor is first created. Its purpose is to perform an initialization needed by the rule implementor instance before it is actually fired. The following parameters are passed to the `init` method:

  The initialization parameters defined for the rule being triggered
  : These can be any parameters needed to properly initialize the rule implementor instance. Often the initialization parameters consist of constants required by the algorithm. For example, when using a rule implementor that checks whether a number is greater than a threshold value, the threshold value normally is passed as an initialization parameter. This parameter is null if there are no initialization parameters for the rule.

  An array of names of dependent rules for the rule being triggered
  : Normally, the rule implementor stores these names to be used when the `fire` method is called. These dependent rules are intended to be triggered as part of the algorithm performed by the rule implementor. See "Dependent rules" on page 148 for more information. This parameter is null if there are no dependent rules defined for the rule.

  The user-defined data for the rule being triggered
  : This data is completely defined by the user of the Business Rule Beans (BRBeans). BRBeans does not interpret this data in any way. This parameter is null if there is no user-defined data defined for the rule.

  A reference to the actual rule being triggered
  : This can be used to extract attribute values from the rule, if needed.

fire
: The fire method comes from the `RuleImplementor` interface. This method is called to perform the algorithm of the rule implementor. Any desired algorithm can be performed here. Normally, a value is returned by the `fire` method that is ultimately returned as the result of triggering the rule. The following parameters are passed to the fire method:

  The TriggerPoint object that is being used to trigger the rule
  : This parameter is needed if the rule has dependent rules that the fire method needs to trigger.

  The target object for this particular trigger call
  : This parameter can be any object that is thought of as the target of the rule. However, the parameter can be null.

A reference to the actual rule being triggered
: This parameter can be used to extract attribute values from the rule, if needed.

The firing parameters for this particular trigger call
: Normally, these parameters are the firing parameters passed by the code invoking the trigger point. However, these can be overridden by specifying firing parameters on the rule itself. Wherever they ultimately come from, these are the parameters that the rule implementor needs at run time to perform its function. Normally, these will be run-time variables that are to be processed by the rule implementor. For example, when using a rule implementor that checks whether a number is greater than a threshold value, the number to be checked normally is passed as a firing parameter. This parameter is null if no firing parameters are passed by the caller and none are defined on the rule itself.

getDescription
: `getDescription` comes from the `RuleImplementor` interface. The purpose of this method is to return a text string that describes the function of the rule implementor. This information might be displayed on a user interface to help a user select what implementor to use. This method, however, is currently not used by the BRBeans framework. For additional imformation, see the RuleImplementor interface in the BRBeans Javadoc (which is located in the InfoCenter).

Rule management command

The Rule management command assists the user in performing high-level administration of rules and rule folders.

This includes the capability to create, modify, delete, import, or export rules or rule folders. This command can be used initially by the programmer to define rules interactively and then used by the domain analyst for rule management tasks. You can use the following files:
- On Microsoft Windows platforms, `rulemgmt.bat`
- On Unix platforms, `rulemgmt.sh`

Syntax

```
rulemgmt properties-file [host-address port-number]
```

Parameters

`<properties-file>`
: The fully qualified name of a file containing the JNDI names of the rule EJBs for the rule set that is to be accessed.

: The following must be specified in the file:

```
RuleJndi=<JNDI name of the Rule EJB>
RuleFolderJndi=<JNDI name of the RuleFolder EJB>
RuleHelperJndi=<JNDI name of the RuleHelper EJB>
```

: See `<WAS_HOME>/bin/brbeansDefaultProperties` for an example.

Arguments

host-address
: This is the host name of the name server. The default is the local host.

port-number
This is the port number of the name server. The default is 2809.

Rule importer command

The rule importer command imports rules into a database from one or more XML documents.

The rule importer command can be invoked using the Rule Management Application (RMA). The user interface in RMA provides some assistance in specifying the parameters required by the importer. Alternatively, the rule importer can be invoked from the command line using the following scripts:

- For Microsoft Windows platforms, `ruleimporter.bat`
- For UNIX platforms, `ruleimporter.sh`

Syntax

```
ruleimporter <properties-file> <import-files> [options]
```

Parameters

`<properties-file>`
The fully qualified name of a file containing the JNDI names of the Business Rule Beans (BRBeans) enterprise beans for the rule set that is to be accessed. Refer to “BRBeans properties file” on page 164 for a definition of the contents of this file. This parameter is required.

<import-files>
One or more fully qualified names of the files containing XML rule definitions to be imported. These files must contain XML in the format defined in `<WAS_HOME>\bin\brb.dtd` file. This parameter is required.

Options

-[v]erbose
Shows verbose output while importing. This shows the rule definition of every rule that is imported.

-[t]est
Parses the input files only and does not create rules on the application server. This will ensure that there are no errors in the syntax of the rule definitions provided in the XML document. Combined with the **-verbose** option, it also can be used to see exactly what rules will be imported.

-[u]pdate
Update the existing rule with values from the input file when a rule in an input file has the same primary key as an existing rule. If this option is not specified, then any rule with the same primary key as an existing rule causes an error and that rule is not imported.

-[c]ommiteach
Performs a commit after each rule is created rather than creating all of the rules in a single transaction. If this option is not specified, then all rules are created in a single transaction. This means that if any rule causes an error, the entire transaction is rolled back and none of the rules are imported. If **-commiteach** is specified and a rule causes an error, only that rule is not imported. Other rules are still imported.

-[h]ost <host-name>
Specifies the name of the host for the name server. The default is the local host.

-[p]ort <port_number>
Specifies the port number for the name server. The default is 2809.

Rule exporter command

The rule exporter command exports rules from a database into an XML document.

The rules that are exported are determined by an XML document, which is provided to the command. The rule exporter function can be invoked using the Rule Management Application (RMA). The user interface in RMA provides some assistance in specifying the parameters required by the exporter. Alternatively, the rule exporter can be invoked from the command line using the following scripts:

- For Microsoft Windows platforms, `ruleexporter.bat`
- For UNIX platforms, `ruleexporter.sh`

Syntax

```
ruleexporter <properties-file> <export-list-files>[options]
```

Parameters

<properties-file>
The fully qualified name of a file containing the JNDI names of the BRBeans enterprise beans for the rule set that is to be accessed. Refer to "BRBeans properties file" for a definition of the contents of this file. This parameter is required.

<export-list-files>
One or more fully qualified names of files containing a list of rules, folders, or both to be exported. These files must contain XML in the format defined in the `<WAS_HOME>\AppServer\bin\brb-export-list.dtd` file. This parameter is required.

Options

-[v]erbose
Shows verbose output while exporting.

-[o]utput <file-name>
Specifies the name of the output file where the XML rule definitions are stored. This is a required parameter.

-[h]ost <host-name>
Specifies the name of the host for the name server. The default is the local host.

-[p]ort <port-number>
Specifies the port number for the name server. The default is 2809.

BRBeans properties file

Applications that use the Business Rule Beans (BRBeans) enterprise beans (this includes applications that trigger rules or use the rule management APIs) must specify the JNDI names for these enterprise beans so that the application can find them at run time. If the application is running in a J2EE client container, in a servlet, or on the application server itself (for example, as part of another enterprise bean), then these names probably have been specified by the person who configured the application. If the application is not running in a container, the names must be specified some other way. The BRBeans properties file provides a way to do this.

At run time, the BRBeans code looks for a special Java property that identifies the name of the properties file. This Java property can be specified on the command line as `-DbrbPropertiesFile=<file_name>`. The file specified is expected to contain the JNDI names used to find the BRBeans enterprise beans. The BRBeans framework uses these names when it needs to locate the enterprise beans.

When an application attempts to reference BRBeans enterprise beans, the code first looks for the brbPropertiesFile Java property. If this property is specified, the names listed in that file are used to find the enterprise beans and to override any EJB references that were specified in the container (if the application is running in a container). If the property is not specified, then BRBeans attempts to use the EJB references specified in the container.

The host name and port number used to access the name server also can be set in this file. If these are not specified, the BRBeans framework uses the name server used by the container in which the application is running. If the application is not running in a container, then localhost is used for the host name and 2809 is used for the port number.

The properties file must be in the following format (entries can be specified in any order):

```
host=<host-name-for-server>
port=<port-number-for-server>
RuleJndi=<JNDI-name-for-Rule-EJB>
RuleFolderJndi=<JNDI-name-for-RuleFolder-EJB>
RuleHelperJndi=<JNDI-name-for-RuleHelper-EJB>
```

Location

A default properties file is shipped as `<WAS_HOME>\AppServer\bin\brbeansDefaultProperties`.

There are a set of JAR files that conform to the BRBeans<database-type>.jar naming convention (depending on the type of database that you want to use). If the JAR files are used without changing the JNDI names, then you also can use the default properties file.

Usage note

The file name still must be specified even if you want to use the default file. There is no file that is used automatically if the `brbPropertiesFile` property is not set.

The tools shipped with BRBeans (the Rule Management Application, the rule importer, and the rule exporter) all run outside of any container. Hence, the JNDI names need to be specified when these tools are run. The scripts for these tools all require that you pass a properties file name as a command line parameter. This name then is specified as the value for the brbPropertiesFile property when the tool is run.

Database considerations for BRBeans

The following relational databases are supported by Business Rule Beans (BRBeans):

- IBM DB2
- IBM Cloudscape
- Microsoft SQL Server

- Oracle
- Sybase
- Informix®

This documentation does not provide you with specific instructions on how to use any of these databases. For help with specific commands, consult the documentation that accompanied your database software. The following are general considerations for relational databases that are supported by BRBeans:

Large character data

There are several attributes in the BRBeans Rule enterprise beans that might contain large amounts of data. This includes fields such as: `businessIntent`, `dependentRules`, `description`, `firingParameters`, `initParameters`, `originalReq`, and `userDefinedData`. The value for these attributes is stored in a character type column within a database table. Whenever possible, the values are stored in large character fields like `LONG VARCHAR` (for DB2) and `TEXT` (for Sybase).

There are several cases where the use of large character fields is problematic, mostly in terms of a lack of query support. Refer to each of the supported database sections for details on the column type used for storing the values in these attributes.

Isolation level

All of the enterprise beans accessed in a transaction must specify the same isolation level. If your application contains enterprise beans that are used in the same transaction as the rules, you must do one of the following:

- Change the BRBeans enterprise beans (`Rule`, `RuleFolder`, and `RuleHelper`) to the same isolation level as your beans.
- Change your beans to the same isolation level as the BRBeans enterprise beans.
- Place the BRBeans enterprise beans in a different database than your enterprise beans and configure the application to run using the two-phase commit protocol. This causes the beans to run in different transactions; thereby removing the restriction that they need to have the same isolation level.

Oracle considerations

Large character data

The preferred Oracle data type for storing large character objects is `CLOB`. However, Oracle does not allow a `CLOB` to be queried. Because of this, a data type of `VARCHAR2` is used by Business Rule Beans (BRBeans). A specific length must be specified when specifying `VARCHAR2`. The maximum length for a `VARCHAR2` is 4000 bytes.

To determine the default size of `VARCHAR2`, look in the table.ddl file that was generated when you deployed the code. If the default size is not acceptable for your application, you can do one of the following:

Increase the size of the columns
: Keep in mind that maximum size for a `VARCHAR2` in Oracle is 4000. Increase the column size either by changing the value in the **create table statement** or by changing the schema mapping and deploying the BRBeans JAR file.

Change the schema mapping to specify CLOB
Do this for any of the attributes that you do not wish to query and then deploy the BRBeans JAR file.

Isolation level

The default isolation level is REPEATABLE_READ. Oracle does not support this isolation level. Therefore, the IBM WebSphere runtime environment converts this to the next highest isolation level, which in this case is SERIALIZED. Be aware that this isolation level tends to be overly restrictive as it prevents two clients from reading data at the same time. The BRBeansOracle.jar file specifies an isolation level of READ_COMMITTED.

Sybase considerations

Allowing null values

By default, Sybase does not allow null values in string columns (like VARCHAR, TEXT, and so on). You can change this default value for a database using ″isql″ by issuing the following command:

```
sp_dboption databasename, ″allow nulls by default″, true
```

In this example ″databasename″ is your database name.

Large character data

The large character data fields are stored in a column of type TEXT. Sybase allows TEXT fields to be queried only using the SQL ″LIKE″ operator. Queries against the columns that perform the SQL ″IS NULL″ or ″IS NOT NULL″ operations are not allowed by Sybase. The alternative is to specify a column type of VARCHAR. However, the maximum allowed size for a VARCHAR in Sybase is 255 characters. This is not considered a large enough value for storing firingParameters, initParameters, descriptions, and so on.

If performing ″IS NULL″ and ″IS NOT NULL″ type queries is important and the 255 character limitation is acceptable, change these column types to VARCHAR. To accomplish this, alter the schema mapping for the Rule bean and then deploy the BRBeans JAR file.

The query APIs (in the com.ibm.websphere.brb.query package) allow for ″IS NULL″ and ″IS NOT NULL″ type queries to be performed on several of these fields. In addition, the Rule Management Application allows the firing parameters to be queried in this manner. These queries fail on Sybase with the default column type of TEXT.

Informix considerations

Large character data

The preferred Informix data type for storing large character data is CLOB. However, Informix does not allow a CLOB to be queried. Because of this, a data type of LVARCHAR is used by BRBeans. The maximum length for an Informix LVARCHAR is 2,000 characters. If 2,000 characters is not acceptable and your application does not need to query these data types, you can change the schema mapping to specify CLOB. Then, deploy the BRBeans JAR file.

Custom properties for the data source

When configuring the data source for your application, you must specify the following properties:

- `ifxIFXHOST=Name of the physical machine on which the Informix instance is installed`
- `serverName=Informix instance name`
- `portNumber=Port number for which the Informix instance is configured`
- `informixLockModeWait=500`

  A setting of 500 causes a connection to wait for up to 500 seconds for a lock. If you have a busy system, this wait can appear to be a system hang. This setting has the same effect as running SET LOCK MODE TO WAIT 500 on the connection.

Note: The previous configuration values are subject to change. Consult your Informix documentation for updates.

Rule Management Application

The Rule Management Application (RMA) is a tool that assists the user in performing high-level administration of rules and rule folders.

This includes the capability to create, modify, delete, import, or export rules or rule folders. The RMA tool can be used initially by the programmer to define rules interactively, and then by the domain analyst for rule management tasks. The main window for the RMA is the Rules Browser.

The column on the left side of the Rule Browser window shows a nested hierarchy of all of the existing rule folders. Click one of these folders to display the rules it contains. The names of these rules appear in the right column.

Navigate as you would in a typical file-management browser.

1. Click the ″+″ icon to expand by one level and click the ″-″ icon to collapse it.
2. Click a rule or folder name to highlight it, right-click the rule or folder name to launch a list of actions, or select an option from the main menu.

RMA is designed to be a general purpose tool for interactive management of rules. Alternatively, you can write your own user interface that is tailored more specifically to the domain in which you work. For instance, a domain-specific user interface can provide more help to the user in the task of managing rules than a general purpose tool such as RMA. If you plan to write your own user interface, refer to the RuleImplementor interface in the BRBeans Javadoc for assistance. The BRBeans interface is located in the InfoCenter.

Rule management APIs

Business Rule Beans (BRBeans) provide a set of APIs to perform rule management tasks programmatically. These tasks include creating, deleting, and updating rules and folders. These APIs are provided to simplify the interaction with the BRBeans enterprise beans. Use these APIs to perform rule management tasks instead of coding directly in the EJB interfaces.

The rule management APIs consist of the classes in the `com.ibm.websphere.brb.mgmt` package. You might use the following main classes:

IRule

This is the interface used to access the object representing a business rule in BRBeans. It provides methods to read and update attributes of the rule, to delete the rule, and to make a copy of the rule. The methods to create rules are in the IRuleFolder interface since you must always create a rule and specify a particular folder in which it will reside. In the rule management APIs, any time you get a rule, you have the option to receive a reference to the enterprise bean itself or to receive a local copy of the data contained in the enterprise beans. Regardless of which option you choose, the IRule interface can be used to access the returned object. If a local copy is requested, it is possible to cast the returned object to an IRuleCopy. IRuleCopy extends IRule and adds a couple additional methods to those defined by IRule. See the **IRuleCopy** section for more details.

IRuleCopy

This is the interface used to access a local copy of the enterprise bean that represents a business rule. An object implementing this interface is returned from rule management API methods if you ask for a local copy of the rule. The main reason for requesting a local copy is performance. Calling a method on a local copy is much faster than calling the method on the actual enterprise bean. If you need to access several different rule attributes, this may make a big difference. Similarly, when updating a rule, all updates can be sent to the enterprise bean in one method call instead of many. The individual set methods are called on the copy and then the updatePersistentRule() method is called to actually send the updates to the enterprise bean.

IRuleFolder

This is the interface used to access the object representing a rule folder. It provides methods to create, delete, and find rules and subfolders. It also provides methods to move and rename the folder, and to get the parent folder. The IRuleFolder representing the root folder is generally what you start with when performing rule management tasks. Once you have the root folder you can navigate up and down the folder hierarchy and access rules contained within the folders.

RuleMgmtHelper

This is a helper class intended to contain methods that are of general use for performing rule management tasks. Currently, the only methods available are used to get the IRuleFolder representing the root folder. The root folder is normally the starting point for performing rule management tasks.

IParameter

This is the interface used to represent an initialization or firing parameter stored in a Rule EJB. Every parameter has a user description and a value that are accessible from this interface. The following classes are provided to implement the IParameter interface:

ConstantParameter
: This is the most common type of parameter. It represents a single constant value that is to be passed as an initialization or firing parameter.

MethodCallParameter
: This class represents a parameter whose value is determined by calling a method on the target object. The method to call must be a public method and must take zero parameters. This is only used for firing parameters.

TriggerPointParameter
This class represents a parameter that is retrieved from one of the trigger point firing parameters. This is mainly used for reordering the firing parameters passed on the trigger point. This is only used for firing parameters.

For more details on the rule management interfaces, including a number of coding examples, refer to the **com.ibm.websphere.brb.mgmt** package in the BRBeans Javadoc.

BRBeans performance enhancements

Externalizing business logic using Business Rule Beans (BRBeans) has many benefits, but does not come completely without a cost. Since every business rule is represented by an enterprise bean, then, in general, every rule trigger is performed in three parts:

1. a query is performed to find the enterprise beans that represent the rules to be triggered.
2. a remote method call is performed on the EJB instance to trigger the rule.
3. a remote method call is made to determine whether to fire the rule locally or remotely.

The first two steps both require server processing so processing can become rather slow.

This section documents the following ways to improve performance:
- Performance enhancements through caching
- Performance enhancements using indexes
- Performance enhancements by changing the firing location

Performance enhancements through caching

The Business Rule Beans (BRBeans) framework incorporates a cache on the client side; that is, wherever the trigger() method on the TriggerPoint object is called. This cache is scoped to the Java virtual machine (JVM) in which the client is running so that any trigger calls performed in a particular JVM use the same cache and two triggers performed in different JVMs use two different caches. The BRBeans cache caches the results of all of the queries performed to find a set of rules to be triggered. The next time a trigger is performed in that JVM with the same rules specified, the rules are found in the cache and the query does not require server processing.

Once the rules are found in the cache they are triggered, either locally or remotely, depending on how they were defined. If a rule found in the cache is specified to be triggered locally, then the entire trigger process for that rule is performed on the client without calling the server. Even if the rule is specified to be triggered remotely, finding the rule in the cache eliminates one call to the server since the query is not performed on the server.

The BRBeans cache can improve performance greatly, however it has one disadvantage: changes made to rules are not recognized immediately.

When a change is made to a rule on the server, there is no way to inform all of the potential clients that something has changed and that they may need to refresh their caches. Thus, the client cache must check periodically to see if anything in the persistent rule data has changed. This is implemented by associating a polling frequency with the cache. This polling frequency specifies an interval of time that the cache waits before checking to see if anything has changed. The next time a trigger is performed after a polling interval has passed, the cache checks to see if any changes have been made to the persistent rule data stored on the server. If no changes have been made, then the cache is not refreshed. If any changes have been made, the entire cache is cleared so that the changes are picked up. Thus, changes to the rules are only picked up by the cache after a polling interval has passed.

The default polling frequency is 10 minutes. The user can change this value by changing the single initialization parameter specified for the special rule named `com/ibm/websphere/brb/BRB CacheRule`. The value for this initialization parameter is in the following format: `hh:mm:ss`

hh stands for hours, mm stands for minutes, and ss stands for seconds.

Thus the default of 10 minutes is specified by a value of `00:10:00`. To specify a polling frequency of, for example, 1 hour, 30 minutes, specify `01:30:00`

When this value is changed, it does not take effect until the previous polling interval has passed. Thus, if the previous polling interval is set to 24 hours and the polling frequency is changed to 1 hour, the new frequency does not take effect until the previous 24 hour polling interval passes. The only other ways to get the new frequency to take effect are to either restart the client (since this causes the cache to be re-initialized from scratch) or have the client code call the refreshCache() method on the `TriggerPoint` object. If there is more than one client JVM performing triggers, this must be done for each client since each JVM has its own cache.

Note: There is only one BRB `CacheRule` and this rule applies to all clients. There is no way to set different polling frequencies for different clients.

Caching can be disabled for a particular TriggerPoint object using the disableCaching() method. After the disableCaching() method is called any triggers performed using that TriggerPoint object can not use the cache. Triggers performed using other TriggerPoint objects are not affected.

Performance enhancements using indexes

Creating an index over the database table that is used to store rules is an important way to improve the performance of rule queries. It is recommended that an index be created over the rulename column of the table containing the rules. This greatly improves the performance of rule-triggered queries that are looking for a rule or rules with a specific name. The index saves the query from searching every row in the table. Refer to the documentation for your database for instructions on how to create an index.

Performance enhancements by changing the firing location

The Business Rule Beans (BRBeans) framework allows you to specify where to fire a particular rule. This determines where the rule implementor is instantiated and invoked. The following lists the possible values for the firing location:

Local Fires the Java rule implementor in the same JVM in which the trigger was performed.

Remote
Fires the Java rule implementor on the server where the rules exist.

Anywhere

Tries to fire the Java rule implementor locally. If the Java rule implementor cannot be found, then it fires the the implementor remotely. This is the default value.

For simple rule implementors that do not perform any server-intensive work, specifying **Local** usually results in the best performance. This is true both without and with caching. A complete comparison of local firing versus remote firing must consider four cases: local and remote firing without caching and local and remote firing with caching. A description of these four cases follow:

Remote call without caching

Without caching, the work done to fire a rule remotely involves the following:
1. Finding the rule.
2. Determining whether the rule is to be fired locally or remotely.
3. Calling fire on the remote rule.

Each of these three operations requires a remote call to the server.

Local call without caching

Without caching the work done to fire a rule locally involves the following:
1. Finding the rule.
2. Determining whether the rule is to be fired locally or remotely.
3. Calling fire on a local copy of the rule.

This requires only two remote calls. Firing locally results in a savings of one remote method call.

Local call with caching

With caching, local firing results in even more dramatic improvements. The work done to fire a rule remotely involves the following:
1. Finding the rule. This involves a search of the local cache and does not involve calling the database.
2. Calling fire on the remote copy of the rule found in the server's cache.

This requires only one remote method call.

Remote call with caching

The work done to fire a rule locally with caching involves the following:
1. Finding the rule. This involves a search of the local cache and does not involve calling the database.
2. Calling fire on the local copy of the rule found in the cache.

This does not involve any remote method calls. The entire rule firing process takes place locally without remote method calls. To get the full benefit of the BRBeans cache, use local firing. However, remotely fired rules still benefit from the cache due to the elimination of the query on the server and the elimination of the remote call to determine whether the rule is being fired locally or remotely.

There may be some cases where a rule implementor must perform some work that requires significant interaction with the server. In these cases, it may be beneficial to have rules using this rule implementor defined to be fired remotely. This might make the server interaction performed by the implementor more efficient.

Note: In addition to performance, maintenance also must be considered in relation to specifying a firing location. The rule implementor classes for rules that are defined to be fired locally must be present on any client system that tries to fire those rules. Otherwise, the implementor cannot be instantiated when the rule is fired. This can result in maintenance problems when the rule implementors are changed since they must be updated on many different systems.

Developing BRBeans

Although the development tasks in this article are shown in sequential order, the exact sequence is left to your discretion. In particular, you might choose to create the business rules before or after the trigger point is placed. Also, you can write the rule implementor before or after creating the actual business rules. However, if you do not have the rule implementor at the time that the business rule is created, then you cannot complete the rule implementor field or the initialization parameters in the rule. However, you can leave the business rule marked as not ready until you can complete that information. For this reason, we have chosen to list the task of writing the rule implementor first. Complete the following tasks to develop Business Rule Beans:

Steps for this task

1. "Determining where to place a trigger point"

   Determine where there are points of variability and where business logic must be externalized as part of application analysis and design process.
2. "Placing a trigger point in the application code" on page 175

   Add code to the application to invoke the trigger point framework, to find business rules, and to fire the rules.
3. "Administering strategy objects to control triggers" on page 176

   Control the process of triggering the rules using a set of strategy objects.
4. "Implementing business rules" on page 177

   Use a rule implementor, written in Java, to implement the BRBeans `RuleImplementor` interface. Also, create business rules invoked by the rule implementor.

What to do next

It is possible to develop your own customized strategy objects and rule implementors. See "Customized strategy objects" on page 159Customized strategy objects and "Customized rule implementors" on page 161Customized rule implementors for more information.

Determining where to place a trigger point

To determine where to place your trigger points, you can use either the case analysis or the Object Interaction Diagrams (OIDs) method. The following are examples of methods that you can use to determine where to place a trigger point:

Use case analysis

Trigger points can be found during analysis by inspecting the use cases or user interaction scenarios that are typically developed as statements of requirement as input to the analysis process. A fragment of a use case is shown below:

The vehicle is entered into the system or chosen. The customer service representative attempts to locate the named driver in the system. If the driver is not found, she or he is added to the system and then picked.

Otherwise the found driver is simply picked. If the vehicle is an auto, anyone between the ages of 16 and 75 can be picked as a driver. If the vehicle is a truck, only drivers 16 to 70 years old can be picked. And if the vehicle is a motorcycle, drivers 14 to 65 can be picked.

After the driver has been picked, a rate quote can be performed...

To identify potential trigger locations in use case analyses such as this one, look for **keywords** such as:
- ″if X is in a special category Y″ (For example, ″if the vehicle is a truck″)
- ″except when″
- ″unless″
- ″depends on″

Object Interaction Diagrams (OIDs)

OIDs that are based on use cases can yield a number of observable patterns that can be used to identify trigger points fairly easy. The following are some of the rules to look for and where the trigger point might be placed:
- Validation of edits on create methods.
- Validation of edits on set methods.
- Referential integrity of edits on methods that set references.
- Cardinality checks at a consistency point (a point in time where all of the data is expected to be self consistent).
- Required fields checks at a consistency point.
- Cross field edits at a consistency point.
- Constraints or derivations that have a high potential for reuse (especially if the algorithm is complex) at any appropriate point.
- Constraints or derivations that a business desires to be consistent across applications (at any appropriate point).
- Constraints or derivations where the business wants to decouple the maintenance cycle for a rule from the maintenance cycle for the code (at any appropriate point).

Results

By using either method, you will be able to identify where to locate trigger points to use Business Rule Beans (BRBeans) effectively.

What to do next

Once the trigger points have been identified, place the trigger point or points. See “Placing a trigger point in the application code” on page 175 for more information.

Placing a trigger point in the application code

Before you begin

Before placing a trigger point, review the following topics:
- Trigger point framework
- Determining where to place a trigger point

The `TriggerPoint` class is the primary interface of the Business Rule Beans (BRBeans) Trigger Point framework. The class is used to transfer control to the Trigger Point framework to find and fire the rules specified in the application's trigger point.

Steps for this task

1. Create an instance of the `com.ibm.websphere.brb.TriggerPoint` class. All rule triggers must be performed against an instance of this class. Also, set any desired strategies on the `TriggerPoint` instance.
2. Gather together the parameters to be passed on the trigger.

   For the simple `trigger()` and `triggerClassifier()` methods this includes the following:

   An optional target object
   This can be used to specify an object that is to be the target of the rule's algorithm. Whether this is needed depends completely on the design of the rule implementor being used.

   The firing parameters for this rule trigger
   This is an array of run-time parameters needed by the rule to satisfy its business purpose. The exact set of required firing parameters is determined by the rule implementor that is used by the rule.

   Note: Any firing parameters defined on the rule itself will override whatever is passed here.

   Information identifying the rule or rules to be triggered
   Normally this is either a single String containing the name of the rule to be triggered or an array of Strings each element of which is the name of a rule to be triggered. However if a custom finding strategy is being used, this could be whatever information it needs in order to find the correct rules.

   The `triggerSituational` method differs in that it takes two sets of firing parameters and two sets of rule identification information: one set for the classifier rules and one for the classified rules.
3. Invoke the `trigger()`, `triggerClassifier()`, or `triggerSituational()` method of the TriggerPoint instance. This will actually trigger the rule or rules.
4. Process the results of the triggered rule or rules.

What to do next

Examples of how to code a trigger point call are provided in the following topics:
- Simple trigger point
- Classifier trigger point
- Situational trigger point

For a detailed description of the trigger point programming interfaces refer to the Trigger Point class in the BRBeans Javadoc. The BRBeans Javadoc is located in the InfoCenter.

Administering strategy objects to control triggers

Strategy objects are used to alter `TriggerPoint` functions. The two simple trigger methods, `trigger()` and `triggerClassifier()`, perform their function in the following sequence:

Steps for this task

1. Find the rules.
2. Filter out those rules which are not desired.
3. Fire the remaining rules.
4. Combine the results and return to the caller.

Results

The complex trigger method `triggerSituational()` does this sequence of steps twice; the first sequence finds the classification to feed into the second sequence.

Default strategy objects already are defined for each of the four `TriggerPoint` steps and they are used if none are specified explicitly. For each of these steps, there are at least two strategy objects used, one for triggering classifier rules, and one for triggering non-classifier rules. For the filtering step, there are actually three pairs of strategies that are used, based on the number of rules which the finding strategy returns (zero, one, or multiple).

While the sheer number of strategies that are available can be intimidating (twelve different strategy classes can be set), very few will need updating. Most users will modify the filtering strategies or the combining strategies only.

A number of predefined strategy objects are provided and can be used for the majority of cases. Although the following strategies are described separately, they have a definite dependency on each other. For example, the FilteringStrategy filters rules from the FindingStrategy; the FiringStrategy uses the results of the FindingStrategy to operate; and the CombiningStrategy depends on the results of the FiringStrategy.

FindingStrategy
The FindingStrategy accesses the data store and returns those rules that meet the search criteria specified. See the “FindingStrategy method” on page 157 for more information.

FilteringStrategy
The FilteringStrategy takes the list of rules that were found by the FindingStrategy and filters out the rules that should not be fired. See the “FilteringStrategy method” on page 157 for more information.

FiringStrategy
The FiringStrategy takes the rules that were found by the FindingStrategy, (possibly modified by the FilteringStrategy), fires them each in order, and returns an array containing the results of each rule. See the “FiringStrategy method” on page 158 for more information.

CombiningStrategy
The CombiningStrategy takes the results of the rules that are fired by the

FiringStrategy and combines them to form a reasonable result to the TriggerPoint caller. See the "CombiningStrategy method" on page 159 for more information.

The Java classes for these strategy objects are defined in the `com.ibm.websphere.brb.strategy` package. Static constants also are defined in the interfaces for the various strategies. This allows easy access to instances of the strategy classes to set them on the `TriggerPoint`.

Also, it is possible to write your own strategy class if the supplied ones do not perform the function you need. See "Customized strategy objects" on page 159 for more details.

Implementing business rules

Before you begin

After determining where to place a trigger point, placing the trigger point, and defining your strategy objects, you must provide a method to implement the business rules and then create the rules. In addition, you might choose to organize your rules by creating rule folders.

Note: Although the development tasks in this article are shown in sequential order, the exact sequence is left to your discretion. In particular, you might choose to write the rule implementor before or after creating the actual business rules.

The business rule encapsulates and externalizes the business logic for the rule and any data that parameterizes the rule. Complete the following process to implement business rules:

Steps for this task

1. Provide an implementation, called a rule implementor, for each business rule that you create

   The rule implementor provides the actual business logic for the rule, implemented in Java. The rule implementor's fire() method is called when the business rule is triggered to actually perform the processing for the rule. Several generic rule implementors are shipped with BRBeans, which might be useful in some situations. However, if these rule implementors do not meet your needs, you can write your own rule implementor. Refer to the section on writing your own rule implementor for details.
2. Use the Rule Management Application (RMA) to create the business rule
   a. In the **Rule Browser** window, select the folder where you want the new rule to be created.
   b. From the main menu, click **File > New > Rule**.

      In the **New Rule** properties window, use the following tabs to define the rule:

      General
      Use this tab to enter general information about the rule.

      Implementation
      Use this tab to define the manner in which the rule is implemented.

      Description
      Use this tab to define the purpose and intent of the rule.

Dependent Rules
Use this tab to specify the rules that the newly created rule will depend upon.

Other Use this tab to to establish precedence, and enter information that is relevant to you, but does not fit into any other category.

c. To complete the creation of the rule, click **OK**.

If there are any mandatory fields still undefined, either go back and give them a value, or make the rule unavailable for use (see **Status** in the **General** tab for more information on this).

3. **(Optional)** Create a rule folder using the Rule Management Application to contain the new business rule
   a. In the **Rule Browser** window, select the folder where you want the new folder to be nested.
   b. From the main menu, click **File > New > Folder**.

   A new folder appears in the folder hierarchy in edit mode. Enter a folder name and hit the **Enter** key.

Assembling applications for use with BRBeans

When you are ready to ship your application, include a Business Rule Beans (BRBeans) JAR file in your EAR file. There are several of these JAR files in the `<WAS_HOME>/BRBeans` directory; one for each supported database. Each name reflects the database type that it uses (for example, BRBeans**DB2**.jar). These JAR files contain three enterprise beans with the following JNDI names:

- `brbeans/application/Rule`
- `brbeans/application/RuleFolder`
- `brbeans/application/RuleHelper`

In your EAR files, complete the following steps:

Steps for this task

1. Change the JNDI names of the BRBeans enterprise beans to make them unique for your application.

   For example, if your application is called MyApp, you could change the first one to `brbeans/MyApp/Rule` or `com/MyCompany/MyApp/Rule`.
2. Define EJB references to these three enterprise beans in any module where a `trigger...()` method exists in one or more of its classes.

   You can do this using the WebSphere Studio Application Developer tool or the Application Assembly tool. The Name field should contain the following and correspond to the enterprise beans listed above:
   - `ejb/com/ibm/ws/brb/Rule`
   - `ejb/com/ibm/ws/brb/RuleFolder`
   - `ejb/com/ibm/ws/brb/RuleHelper`

   Note: The JNDI name on the **Bindings** tab should be the same as the JNDI names that you gave earlier to the enterprise beans.

   Since the BRBeans enterprise beans refer to each other, there are also EJB references defined within the BRBeans JAR file itself. Each of the three BRBeans enterprise beans has two EJB references defined. These six references also need to be updated with the JNDI names you assigned earlier.

3. If you are not using the BRBeansCloudscape.jar file, skip this step. In the BRBeansCloudscape.jar file each entity enterprise bean has a resource reference defined for the data source it will use. You must update the JNDI binding for this reference to specify the JNDI name of the actual data source you want to use for the BRBeans entity enterprise bean. The shipped BRBeansCloudscape.jar file contains a dummy JNDI name for each binding. There are two resource references that need to be updated: one for the Rule enterprise bean and one for the RuleFolder enterprise bean. You can use the WebSphere Studio Application Developer tool or the Application Assembly tool to perform this update.

What to do next

After you have an EAR file that includes your application code and the BRBeans enterprise beans, complete the following steps to install and run the application:

1. Deploy the EAR file to generate run-time code for the BRBeans enterprise beans and any other enterprise beans that you may have in your application. Deployment can be accomplished either as a separate step (using the ejbdeploy command) or as part of the process of installing the EAR file onto an application server. Refer to the base WebSphere documentation for more information about deploying an EAR file.
2. The deployment process generates a file named Table.ddl. Table.ddl contains database commands to create the database tables needed by the BRBeans entity enterprise beans. You must use your database software to create a database and then use the commands in the Table.ddl file to create tables in this database. Refer to your database software documentation for more information on the commands needed to set up a database and the database tables.
3. Create a JDBC provider and a data source to access the database created in the previous step. For all of the databases except IBM Cloudscape, the BRBeans enterprise beans are configured to require a Version 4 data source. For Cloudscape, you can use a normal data source.

   Note: The JNDI name you specify for the data source also must be specified when you install the EAR file on the application server so that the server knows which data source to use.

   Refer to the base WebSphere documentation for more information on these topics.
4. Install the EAR file on an application server. To install the EAR file, either use command line tools or the WebSphere Administrative Console.

   Note: The EAR file can be deployed either as a separate step or as part of the installation process on the server.

   Refer to the base WebSphere documentation for more information on installing an EAR file on an application server.
5. Start the application using the WebSphere Administrative Console.

You now have an application installed and running using BRBeans.

Managing rules

Before you begin

In Business Rule Beans (BRBeans), rule management involves making changes to the set of business rules used by applications. This can include any of the following activities:

- "Implementing business rules" on page 177: Creating rules and rule folders
- "Copying or moving rules or rule folders" on page 181
- "Working with Quick Copy" on page 182
- "Finding a rule" on page 182
- "Deleting rules" on page 183
- "Deleting rule folders" on page 183
- "Changing the properties of a rule" on page 183
- "Importing a rule" on page 184
- "Exporting a rule" on page 184
- "Renaming rules" on page 185
- "Renaming rule folders" on page 185
- "Specifying columns" on page 185
- "Changing the date and time format" on page 186

There are two different interfaces that can be used for rule management:

Rule Management Application
An external user interface that allows users to manage rules interactively. It provides a general purpose interface for managing rules where no assumptions are made about the content or implementation of the rules. For information on how to use the Rule Management Application, see "Starting the Rule Management Application" on page 181.

Rule management APIs
A programmatic interface that can be used by programmers to manage rules or to customize an external user interface. For more information on Rule management APIs, see the com.ibm.websphere.brb.mgmt package in the Javadoc. The Javadoc is accessible through the InfoCenter.

Rules can be managed in any way that makes sense for your application, but the BRBeans framework was designed with the following administrative paradigm in mind:

Steps for this task

1. Understand the desired change in business behavior.
2. Inspect the application documentation (in particular information that indicates where trigger points are located) to understand where the changes need to be made in the system.
3. Inspect the corresponding set of existing business rules using the Rule Management Application (or your own custom management application, if you have one) to understand which rules need to change.
4. Use the Rule Management Application, on a test system, to create one or more new rules that implement the required new behavior. Give these rules the correct name so that they are triggered by the appropriate trigger point. Also, make sure that these new rules are currently in effect.
5. Withdraw (by setting the end date of the rule), on the test system, all of the rules that are to be superseded.
6. Test the application to ensure that it behaves as expected.
7. Export the new rules using the Rule Exporter on the test system. Schedule the rules to become effective at the correct point in time.
8. Export the rules to be superseded using the Rule Exporter on the test system. Set them to expire when the new rules come into effect.

9. Import the new rules using the Rule Importer on the production system. This creates the new rules and schedules them to become effective at the date you specified when you exported them.
10. Import the rules to be superseded using the Rule Importer on the production system. This puts the new end date into the existing rules on the production system and sets them to expire on the specified date.

What to do next

For more information on the overall topic of Externalizing business rules, see Chapter 6, "Using Business Rule Beans", on page 141.

Starting the Rule Management Application

Before you begin

Review the overview topic "Managing rules" on page 179

To administer BRBeans, use the Rule Management Application (RMA). To launch the RMA, complete the following steps:

Steps for this task

1. Open a command window and change to the following directory: `<WAS_HOME>/bin`
2. For Microsoft Windows platforms, enter `rulemgmt.bat <properties-file>`. For UNIX® platforms, enter `rulemgmt.sh <properties-file>`

   where `<properties-file>` is a fully qualified name of a file containing port, host, and JNDI names used for the Business Rule Beans (BRBeans) enterprise beans. If you are using localhost port 2809 and you are using the default JNDI names for the BRBeans enterprise beans, you can specify the following in the default properties file: `<WAS_HOME>/bin/brbeansDefaultProperties`. For a full definition of the contents of this file, see "BRBeans properties file" on page 164.

Copying or moving rules or rule folders

Copy or move rules or rule folders either by cutting and pasting, or dragging and dropping.

To copy or move rules or rule folders, proceed as follows:

Cutting and pasting
Use the menu commands (**Edit > Copy, Edit > Cut** and **Edit > Paste**) or keyboard commands (**CTRL-C**, **CTRL-V**, and **CTRL-X**).

Dragging and dropping
Highlight the rule or rule folder you want to copy. Then press and hold the right mouse button, drag the cursor to the target location, and release. Select **Copy** or **Move** from the list.

Note: A rule also can be copied so that the copy replaces the existing rule at a specified date. This is referred to as a Quick Copy.

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Working with Quick Copy

Use **Quick Copy** to make a copy of a rule that will replace the existing one on a specified date.

For example, suppose that we have an ″isSeniorCitizen″ rule. Currently a person is considered a senior citizen if they are 62 years of age or older. Starting on January 1, 2002, we are going to change this to 65. Use Quick Copy to specify the new date, and change the age from 62 to 65. The current rule will be set to expire on the same date that the new rule will take effect with the new senior citizen age defined as 65.

To use **Quick Copy** from the Rule Browser or Search Results window, proceed as follows:

Steps for this task

1. Select the rules you want to **Quick Copy**.
2. From the main menu, click **Edit > Quick Copy**.
3. In the Quick Copy window, specify in the following fields how the copy will differ from the original:

   Start Date For New Rule
   Enter the date that the new rule will replace the existing rule. Use the date and time format that is shown.

   Change parameter values for new rule
   Enter the new parameter values.
4. Click **OK** to finish.

What to do next

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Finding a rule

To search for a specific rule using the Rule Management Application **Find** function, proceed as follows:

Steps for this task

1. To search through all rules in all folders:
   a. From the main menu of the Rule Browser, click **Edit > Find**.
   b. Use the tabs in the Find Rules window to determine your search criteria.
2. To search a specific folder:
   a. Right-click the folder and select **Find** from the list.
   b. Use the tabs in the Find Rules window to determine your search criteria.

What to do next

The results of your search are displayed in a Search Results window.

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Deleting rules

To delete rules from the Rule Browser or Search Results window, proceed as follows:

Steps for this task

1. Select the rules you want to delete.
2. From the main menu, click **File > Delete.**
3. Click **Delete** and then confirm the delete request.

What to do next

Note: You cannot delete `com/ibm/websphere/brb/BRB CacheRule` as this rule is needed by the Business Rule Beans run-time environment.

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Deleting rule folders

To delete rule folders from the Rule Browser window, proceed as follows:

Steps for this task

1. Select the folder you want to delete.
2. From the main menu, click **File > Delete.**
3. Click **Delete** and then confirm the delete request.

What to do next

Note: You cannot delete the root folder or any of the folders in the path `com/ibm/websphere/brb`

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Changing the properties of a rule

To change the properties of a rule, perform the following steps in either the Rule Browser or Search Result window:

Steps for this task

1. Highlight the rule you wish to edit.
2. From the main menu, click **File > Properties**.

   In the Rule Properties properties window, use the following tabs to edit the rule's definition:

   General
   Use this tab to edit general information about the rule.

   Implementation
   Use this tab to edit the manner in which the rule is implemented.

   Description
   Use this tab to edit the purpose and intent of the rule.

Dependent Rules
Use this tab to edit the list of rules that the newly created rule will depend upon.

Other Use this tab to establish precedence, and enter information that is relevant to you, but does not fit into any other category.

3. To complete the editing of the rule, click **OK**.

   If there are any mandatory fields still undefined, either go back and give them a value, or make the rule unavailable for use (see **Status** in the **General** tab for more information on this).

What to do next

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Importing a rule

To import rules from an XML format, use the Rule Browser window and proceed as follows:

Steps for this task

1. In the main menu, click **File > Import.**
2. In the Import Rules window, specify the file you want to import.
3. Click **OK**.

   Rules and rule folders are created as specified within the XML.

What to do next

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Exporting a rule

To export rules, use the Rule Browser or Search Results window and proceed as follows:

Steps for this task

1. In the main menu click **File > Export.**
2. In the **Export Rules Wizard**, proceed as follows:
   a. In the Specify Rules to Export window, select the rule or rules that you want to export and click **Next**.
   b. In the Change Effective Dates On Exported Rules window, alter the start and end dates of the rule (if desired) and click **Next**.
   c. In the Select File For Rule Export window, choose a name and location for the exported rule.
3. Click **Export** to finish.

What to do next

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Renaming rules

To rename rules, use the Rule Browser or Search Results window and proceed as follows:

Steps for this task

1. Highlight the rule you want to rename.
2. From the main menu, click **File > Rename.**
3. Type a new name and press the **Enter** key.

   To cancel the name change while it is still in progress, press the **Esc** key.

What to do next

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Renaming rule folders

To rename rule folders, use the Rule Browser or Search Results window and proceed as follows:

Steps for this task

1. Place the folder name in edit mode by performing one of the following tasks:
   a. Right-click the folder and select **Rename** from the list.
   b. Highlight the folder and click **File >Rename** in the main menu.
   c. Click the folder name twice with a slight pause between each click.
2. Type a new name and press the **Enter** key.

   To cancel the name change while it is still in progress, press the **Esc** key.

Results

Note: You cannot change the name of the root folder.

What to do next

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Specifying columns

To choose which columns you want shown in your Rule Browser window, perform the following steps in either the Rule Browser or Search Results window:

Steps for this task

1. From the main menu, click **View > Specify Columns**.
2. In the Specify Column window, proceed as follows:

   To add a new column

   Select one or more entries in the **Available columns** list and click the **Add** button. The selected entries are added to the end of the Columns displayed list.

To remove a column
: Select one or more entries in the Columns displayed list and click the **Remove** button. The selected entries are added to the end of the Available columns list.

To reorder columns
: Select one or more entries in the Columns displayed list. To move the entries towards the top of the list, click the Up arrow. To move them towards the bottom, click the Down arrow.

What to do next

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Changing the date and time format

To change the date and time format, use either the **Rule Browser** or **Search Results** window and proceed as follows:

Steps for this task

1. In the main menu, click **View > Specify Date/Time Format.**
2. In the **Specify Date/Time Format** window, choose one of the following radio button options:

   Use default format for this locale
   : Use this option to adjust the date and time format to match the default setting of your current locale.

   Select a predefined format for the date and time
   : Use this option to choose from one of several existing date and time formats.

   Specify a custom format for the date and time
   : Use this option to determine your own format for your date and time display.
3. When the example in the lower left of the window meets your needs, click **OK** to finish.

What to do next

See "Managing rules" on page 179 for other tasks related to the management of your rules.

Rule Browser

The Rule Browser is the main window of the Rule Management Application (RMA), which is the tool used to administer business rules for Business Rules Beans. The RMA is a simple graphic user interface that assists the user in the high-level administration of rules and rule folders. The column on the left side of the Rule Browser window shows a nested hierarchy of all of the existing rule folders. Open one of these rule folders to display the rules. The names of these folders appear in the right column.

To navigate through the information, perform the following actions:

- Click the **+** icon to expand the folder by one level; click the **-** icon to collapse it.

- Click a file name to highlight it, right-click it to launch a list of actions, or select an option from the main menu.

File menu

This article describes the options available in the File menu window.

New

Use the **New** option to create one of the following objects:

Folder The **Folder** selection creates a new folder within the folder currently selected in the browser. To create a new folder, complete the following steps:

1. Select the folder, in the Rule Browser window in which to nest the new folder.
2. Click **File > New > Folder** from the main menu. A new folder appears in the folder hierarchy in edit mode.
3. Enter a folder name and press **Enter**.

Rule The **Rule** selection creates a new rule within the folder that is currently selected in the browser. To create a new rule, complete the following steps:

1. Select the folder, in the Rule Browser window, where you want to create the new rule.
2. Click **File > New > Rule** from the main menu.
3. Use the following options, in the New Rule Properties window, to define the rule. For more information on each of these tabs, see the associated help file.

   General
   Use the **General** tab to enter general information about the rule. For more information, click the **New Rule properties window: General tab** link under Related reference.

   Implementation
   Use the **Implementation** tab to define the manner in which the rule is implemented. For more information, see “Rule properties window: Implementation tab” on page 197.

   Description
   Use the **Description** tab to define the purpose and intent of the rule. For more information, see “Rule properties window: Description tab” on page 200.

   Dependent Rules
   Use the **Dependent Rules** tab to specify the rules that the newly created rule will depend upon. For more information, see “Rule properties window: Dependent Rules tab” on page 201.

   Other Use the **Other** tab to establish precedence and enter information that is relevant to you, but does not fit into any other category. For more information, see “Rule properties window: Other tab” on page 201.

   Note: When you change the properties of a rule and there are undefined mandatory fields, either give them a value or make the rule unavailable for use. See ″status″ on the General tab for more information.

4. Click **OK** to complete the creation of the rule.

Rule Browser Window
: This selection opens a new Rule Browser window on your desktop and shows the content of the currently selected folder.

Import

Use the **Import** option to import rules that are defined in a file and written in XML. The rules are imported into folders as specified within the XML.

Importing rules

You can use the Rule Browser window to import from an XML format. To import rules, complete the following steps:

1. Click **File > Import** in the main menu.
2. Specify the file, in the Import Rules window, that you want to import.
3. Click **OK**. The rules are imported as specified within the XML.

For more information, see "Import Rules window" on page 195.

Export

Use the Export option to export a file in an XML format.

Exporting rules

You can export rules from the Rule Browser or Search Results windows. To export rules, complete the following steps:

1. Click **File > Export**, in the main menu.
2. Proceed as follows using the Export Rules Wizard:
   - In the Select Rules to Export window, select the rule or rules that you want to export and click **Next**.
   - In the Change Effective Dates On Exported Rules window, alter the start and end dates of the rule, if desired, and click **Next**.
   - In the Select File For Rule Export window, choose a name and location for the exported rule.
3. Click **Export** to finish.

For more information on this Export Rules Wizard, see "Select Rules To Export window" on page 196.

Delete

Use the **Delete** option to delete the selected rule or rule folder. If a rule folder is selected, all of the rules and subfolders the folder contains also are deleted.

Deleting rules

You can delete rules from the Rule Browser or Search Results windows. To delete rules, complete the following steps:

1. Select the rules that you want to delete using the Rule Browser window.
2. Click **File > Delete** from the main menu.
3. Click **Delete** and then confirm the delete request.

Note: You cannot delete `com/ibm/websphere/brb/BRB CacheRule` as this rule is needed by the Business Rule Beans run-time environment.

Deleting a folder

To delete a folder, complete the following steps:
1. Select the folder you want to delete using the Rule Browser window.
2. Click **File > Delete** in the main menu.
3. Click **Delete** and then confirm the delete request.

Note: You cannot delete the root folder.

Rename

Use the **Rename** option to rename the selected rule or rule folder.

Renaming a rule

You can rename rules from the Rule Browser or Search Results windows. To rename a folder, complete the following steps:
1. Highlight the rule you want to rename.
2. Click **File >Rename** from the main menu.
3. Type a new name and press **Enter**. To cancel the name change while it is still in progress, press **Esc**.

Renaming a folder
1. Place the folder name in edit mode by doing any of the following:
   - Right-click the folder and select **Rename** from the list.
   - Highlight the folder and click **File >Rename** in the main menu.
   - Click the folder name twice with a slight pause between each click.
2. Type a new name and press **Enter**. To cancel the name change while it is still in progress, press **Esc**.

Note: You cannot rename the root folder.

Properties

Use the **Properties** option to modify the properties of the selected rule or rule folder and then click **OK**.

Use the following tabs, in the **Rule Properties** window, to define the rule. For more information on each of these tabs, see the associated help file.

General
: Use the **General** tab to enter general information about the rule. For more information, see "Rule properties window: General tab" on page 197.

Implementation
: Use the **Implementation** tab to define the manner in which the rule is implemented. For more information, see "Rule properties window: Implementation tab" on page 197.

Description
: Use the **Description** tab to define the purpose and intent of the rule. For more information, click "Rule properties window: Description tab" on page 200.

Dependent Rules
: Use the **Dependent Rules** tab to specify the rules that the newly created rule will depend upon. For more information, see "Rule properties window: Dependent Rules tab" on page 201.

Other Use the **Other** tab to establish precedence and enter information that is relevant to you, but does not fit into any other category. For more information, see “Rule properties window: Other tab” on page 201.

Note: When you change to the properties of a rule and there are undefined mandatory fields, either give the fields a value or make the rule unavailable for use. See ″status″ in the “Rule properties window: General tab” on page 197 for more information.

Close

Use the **Close** option to terminate the application and close the window.

New Rule properties window: General tab

Use the **New Rule properties window: General** tab to enter general information about the rule. The following fields and options are available on this tab:

Name and location

Folder name
(Required) Use the **Folder name** field to identify the folder in which to create the rule. To browse the existing folders, click the **ellipses** button to the right of the text field.

Name (Required) Use the **Name** field to give the rule a name. The name cannot include the forward slash ′/′ and must contain at least one non-blank character. The name cannot exceed the maximum length of the rule name column in the database table.

Period when rule is in effect

Start date
(Required) Use the **Start date** field to specify the date and time that the rule will go into effect. If you do not specify the time, a value of midnight is used.

End date
Use the **End date** field to determine the date when this rule expires. If you do not specify a value, the rule never expires.

Classification

Use this section to specify your rule’s classification status. Choose one of the following options:

- Rule is not classified and does not perform a classification (default)
- Rule performs a classification
- Rule is classified with the following classification

Status

Rule is available for use
Select **Rule is available for use** when the rule is available for use by the Business Rule Beans run-time environment. This feature is useful when you have not finished creating the rule, but need to save your changes.

New Rule properties window: Implementation tab

Use the **New Rule properties window: Implementation** tab to define the manner in which the rule is implemented.

Note: The rule contains the data, but it does not perform the implementation. Rather, the Java rule implementor implements the rule.

For example, suppose you want to create a rule that determines whether a given person is a senior citizen, 62 years old or older. To implement this rule, give the ″com.ibm.websphere.brb.implementor.RuleGreaterThanEqual″ Java rule implementor the value 62 and specify it as an initialization parameter. When the application fires the rule, the person's age is passed to the Java rule implementor as a firing parameter and 62 is passed as an initialization parameter. The person's age is compared against the initialization parameter of 62 and a value of true or false is returned from the Java rule implementor to the application. To change the age at which a person is considered a senior citizen, change the value of the initialization parameter.

The following fields and options are available on this tab:

Java rule implementor

(Required) Use the **Java rule implementor** field to specify a class to implement this rule. The initialization and firing parameters that are required are determined by looking at the documentation for the specified Java rule implementor.

Firing location

Use the **Firing location** field to determine where the rule is fired. You can fire the Java rule implementor on the server where the rules exist or fire it locally on the client machine. The client can be a servlet running on the server. Specify one of the following values for the firing location:

Local Use the **Local** option to fire the Java rule implementor local to the application that fired the rule.

Remote
Use the **Remote** option to fire the Java rule implementor on the server where the rules exist.

Anywhere (default)
Use the **Anywhere** option to attempt to fire the Java rule implementor locally first. If the Java rule implementor cannot be found, then it is fired remotely.

To choose the value of the firing location, you must take both performance and maintenance into consideration. Most rules perform better if they are run on the same Java virtual machine (JVM) as the application (locally). However, there might be cases where a Java rule implementor performs server-intensive tasks, in which case the rules might run better when they run on the server. To run locally, you must have all of the Java rule implementors installed locally. They must be accessible by the application that fires the rules.

Initialization parameters

The initialization parameters contain constant values passed to the rule implementor when it is initialized. Typically, the initialization parameters contain values that might change as your business practices evolve, such as the age at which a person is considered a senior citizen or the current interest rate for a loan.

To add a new initialization parameter to the list, click **Add** and fill in the fields in the Add Initialization Parameter window.

To edit an existing initialization parameter, highlight it, click **Change**, and fill in the fields in the Change Initialization Parameter window.

To delete an initialization parameter, highlight it and click **Delete**.

To change the order of the initialization parameters, highlight one and click the up or down arrows to move it to a new location.

Firing parameters

The firing parameters contain values passed from the trigger point in the application to the Java rule implementor at run time. You can alter the parameters coming from the application before passing them to the Java rule implementor. Typically, these parameters are left unaltered.

For example, when implementing the ″isSeniorCitizen″ rule that determines whether a person is a senior citizen, you might want to pass a person from the application to the rule as the target object. However, the ″isSeniorCitizen″ rule uses the RuleGreaterThanEqual Java rule implementor, which requires that you pass an integer value. You can alter the firing parameters to specify that the method ″getAge″ is called on the person object and pass that result to the Java rule implementor.

You must choose one of the following options:

Pass firing parameters from trigger point unchanged
: The parameters specified in the trigger point of the application are passed to the Java rule implementor without being altered. This is the default value.

Specify firing parameters
: The values specified in the table are passed to the Java rule implementor.

To add a new firing parameter to the list, click **Add** and fill in the fields in the Add Firing Parameter window.

To edit an existing firing parameter, highlight the firing parameter, click **Change**, and fill in the fields in the Change Firing Parameter window.

To delete a parameter, highlight the firing parameter and click **Delete**.

To change the order of the firing parameters, highlight one and click the up or down arrows to move it to a new location.

Add Initialization Parameter window

Use the Add Initialization Parameter window to add initialization parameters to a rule. The initialization parameters contain constant values passed to the Java rule implementor when it is initialized. Typically, the initialization parameters contain values that will change as your business practices evolve. These values might be the age at which a person is considered a senior citizen or the current interest rate for a loan.

To add an initialization parameter, proceed as follows:

1. Fill in the fields as needed.

2. Click **Add**. The initialization parameter is added and the window remains open to specify additional parameters.
3. When you are finished specifying initialization parameters, click **Close**.

The following fields and options are available in this window:

Description
Use the **Description** field to specify a description of the initialization parameter. This field can contain any information necessary to describe the purpose of the initialization parameter.

Type Use the **Type** field to specify the type of data that is contained within this initialization parameter. The data itself is stored in the **Value** field (see the following field description). For example, if this initialization parameter specifies the age at which a person is considered a senior citizen, then the Type likely is an ″Integer″. If the initialization parameter specifies a company name, such as ″IBM″, then the Type is a ″String″.

The following values are available:

- String
- Character
- Byte
- Short
- Integer
- Long
- Float
- Double
- Boolean
- java.math.BigDecimal
- java.math.BigInteger
- Null Value

Value Use the **Value** field to add a value for the parameter. For example, if the initialization parameter is intended to specify the age at which a person is considered a senior citizen, then this field might be set to 62.

Change Initialization Parameter window

Use the Change Initialization Parameter window to edit an existing initialization parameter.

The initialization parameters contain constant values passed to the Java rule implementor when it is initialized. Typically, the initialization parameters contain values that change as your business practices evolve. These values might be the age at which a person is considered a senior citizen or the current interest rate for a loan.

To edit an existing initialization parameter, modify the fields and click **OK**.

The following fields are available in this window:

Description
Use the **Description** field to specify a description of the initialization parameter. This field can contain any information necessary to describe the purpose of the initialization parameter.

Type Use the **Type** field to specify the type of data that is contained within this initialization parameter. The data itself is stored in the **Value** field (see the following field description). For example, if this initialization parameter specifies the age at which a person is considered a senior citizen, then the Type likely is an ″Integer″. If the initialization parameter specifies a company name, such as ″IBM″, then the Type is a ″String″.

The following values are available:

- String
- Character
- Byte
- Short
- Integer
- Long
- Float
- Double
- Boolean
- java.math.BigDecimal
- java.math.BigInteger
- Null Value

Value Use the **Value** field to add a value for the parameter. For example, if the initialization parameter is intended to specify the age at which a person is considered a senior citizen, then this field might be set to 62.

New Rule properties window: Description tab

Use the **New Rule properties window: Description** tab to define the purpose and intent of the rule. All of the fields in this panel are optional and none of them are used by the Business Rule Beans run time. The following fields are available on this tab:

Business Intent: Use the **Business Intent** field to describe the business intent of this rule.

Description: Use the **Description** field to define a general description of the rule and its purpose.

Original requirement: Use the **Original requirement** field to compose a description of the original requirement that created this rule.

New Rule properties window: Dependent Rules tab

Use the **New Rule properties window: Dependent Rules** tab to specify the rules that the newly created rule will depend upon.

To add names to the **Dependent rule names** field, proceed as follows:

1. Locate the dependent rule. You can do this in one of two ways:

   Browse for a rule
   If you are familiar with the location of the dependent rule, then click the **Browse** button. Navigate to the rule’s location and highlight it.

   Find a rule
   If you are unfamiliar with the location of the rule, then click the **Find** button. This launches a Find Rules window in which you can specify options and then initiate a search. When you have located the rule, highlight it.

2. Click the **Add** button.

To delete a rule from the **Dependent rule names** field, highlight the rule and click **Delete**.

To change the order of the rules, highlight a rule and click the Up or Down arrows to move it to a new location.

New Rule properties window: Other tab

Use the **New Rule properties window: Other tab** to establish precedence and enter information that is relevant to you, but does not fit into any other category. The following fields are available on this tab:

Precedence: Use the **Precedence** field to specify the relative priority when firing the rule. This value is used to order the rules from lowest to highest.

User defined data: Use the **User defined data** field to enter any additional text that you want to store. The Business Rule Beans run time does not use this field.

Import Rules window

Use the Import Rules window to select and import a rule expressed in an XML format. The following fields and options are available in this window:

File Name: Use the **File Name** field to specify the name of a file that contains the rules that you want to import. To search for a file, click the **ellipses** icon to the right of the text entry field.

Show output from rule importer: Select **Show output from rule importer** to display detailed information about the rules that you want to import.

Show rules to be created but do not create them: Select **Show rules to be created but do not create them** to validate the XML prior to committing to the rule's creation. The rule XML runs through the importer but is not created.

Update existing rules with the same primary key: Select **Update existing rules with the same primary key** to update the rules with the same primary key. If this check box is clear, the rule is not imported if one is found with this same primary key. An error message is shown and the transaction in which this rule was created is not committed. The primary key is an optional tag within the XML and there is a possibility that a rule already exists on the system with this same primary key.

Transaction Option: The following is a list of transaction options:

One transaction per rule
Use the **One transaction per rule** option to start a transaction for each rule. If one rule fails to import, it does not prevent other rules in the specified file from being imported.

One transaction for all rules
Use the **One transaction for all rules** option to stop all of the rules from being imported if any rule fails to import successfully. Use this feature to ensure that all of your rules are in a consistent state. Typically, it is undesirable to have only a portion of the rules imported successfully since rules might have dependencies on other rules.

Select Rules To Export window

This is the first of three windows in the Export Rules Wizard. Use the Select Rules To Export window to select the rules to export. After entering the appropriate information in the following fields, click **Next**.

Note: If you specify a folder, the tool exports the entire contents of the folder including its subfolders.

The following options are available in this window:

Add: Use the **Add** option to open a window in which you can type the fully qualified name of a rule or a rule folder. If there are multiple rules with the specified name, they are all exported.

Find

Use the **Find** option to open a Find Rules window in which you can specify your search criteria. If there are multiple rules with the same name, only the selected rules are exported.

Browse button

Use the **Browse** button to open a window and browse for rules to add to the list. If there are multiple rules with the same name, only the selected rules are exported.

Remove button

Use the **Remove** button to remove the selected rules from the list.

Show output from rule exporter

Select **Show output from rule exporter** to open a window that contains details about the export operation of the select rules. This window is shown when the export operation begins.

Change Effective Dates On Exported Rules window

This is the second of three windows in the Export Rules Wizard. Use the Change Effective Dates on Exported Rules window to alter the rule’s start and end times. This procedure is useful when the application and the rules are tested on a development system prior to being deployed on a production system. You can change the dates of the rules and test on the development system using the current date, even if you plan to use the rules on the production system at a future date. The rules on the development system are not changed; only the exported version of the rules are changed. After determining whether to select the check boxes in the following descriptions, click **Next**.

The following fields and options are available in this window:

Change start date and time on exported rules
: Select **Change start date and time on exported rules** to alter the start date of the rules to export. You must specify a valid date and time using the format shown. For example, the format might be ″m/d/yy h:mm a″ resulting in 10/23/01 1:45 PM.

Change end date and time on exported rules
: Select **Change end date and time on exported rules** to alter the end date

of the rules to export. You must specify a valid date and time using the format shown. For example, the format might be ″m/d/yy h:mm a″ resulting in 10/23/01 1:45 PM.

Select File For Rule Export window

This is the third of three windows in the Export Rules Wizard. Use the Select File For Rule Export window to chose a name and location for the exported rule.

1. Browse to an existing directory or create a new one.
2. Type the name of the file. Typically, the file will end with an ″.xml″ extension.
3. Click **Export**.

Rule properties window: General tab

Use the **Rule properties window: General** tab to enter general information about the rule. The following fields and options are available in this window:

Name and location

Folder name
(Required) Use the **Folder name** field to identify the folder in which to create the rule. To browse the existing folders, click the **ellipses** button to the right of the text field.

Name (Required) Use the **Name** field to give the rule a name. The name cannot include the forward slash ′/′ and must contain at least one non-blank character. The name cannot exceed the maximum length of the rule name column in the database table.

Period when rule is in effect

Start date
(Required) Use the **Start date** field to specify the date and time that the rule goes into effect. If you do not specify the time, a value of midnight is used.

End date
Use the **End date** field to determine the date when this rule expires. If you do not specify a value, the rule never expires.

Classification

Use the **Classification** section to specify your rule’s classification status. Choose one of the following options:

- Rule is not classified and does not perform a classification (default)
- Rule performs a classification
- Rule is classified with the following classification

Status

Rule is available for use
Select **Rule is available for use** when the rule is available for use by the Business Rule Beans run-time environment. This feature is useful when you have not finished creating the rule, but you want to save your changes.

Rule properties window: Implementation tab

Use the **Rule properties window: Implementation** tab to define the manner in which the rule is implemented.

Note: The rule contains the data and it does not perform the implementation. Rather, the Java rule implementor implements the rule.

For example, suppose you want to create a rule that determines whether a given person is a senior (62 years old or older). To implement this rule, give the ″com.ibm.websphere.brb.implementor.RuleGreaterThanEqual″ Java rule implementor the value 62 and specify it as an initialization parameter. When the application fires the rule, the person's age is passed to the Java rule implementor as a firing parameter and 62 is passed as an initialization parameter. The person's age is compared against the initialization parameter of 62 and a value of true or false is returned from the Java rule implementor to the application. To change the age at which a person is considered a senior citizen, change the value of the initialization parameter.

The following fields and options are available on this tab:

Java rule implementor

(Required) Use the **Java rule implementor** field to specify a class to implement this rule. The initialization and firing parameters that are required are determined by looking at the documentation for the specified Java rule implementor.

Firing location

Use the **Firing location** field to determine where the rule is fired. You can fire the Java rule implementor on the server where the rules exist or fire it locally on the client machine. The client can be a servlet running on the server. Specify one of the following values for the firing location:

Local Use the **Local** option to fire the Java rule implementor local to the application that fired the rule.

Remote
Use the **Remote** option to fire the Java rule implementor on the server where the rules exist.

Anywhere (default)
Use the **Anywhere** option to attempt to fire the Java rule implementor locally first. If the Java rule implementor cannot be found, then it is fired remotely.

To choose the value of the firing location, you must take both performance and maintenance into consideration. Most rules perform better if they are run on the same Java virtual machine (JVM) as the application (locally). However, there might be cases where a Java rule implementor performs server-intensive tasks, in which case the rules might run better when they run on the server. To run locally, you must have all of the Java rule implementors installed locally. They must be accessible by the application that fires the rules.

Initialization parameters

The initialization parameters contain constant values passed to the rule implementor when it is initialized. Typically, the initialization parameters contain values that might change as your business practices evolve, such as the age at which a person is considered a senior citizen or the current interest rate for a loan.

To add a new initialization parameter to the list, click **Add** and fill in the fields in the Add Initialization Parameter window.

To edit an existing initialization parameter, highlight it, click **Change**, and fill in the fields in the Change Initialization Parameter window.

To delete an initialization parameter, highlight it and click **Delete**.

To change the order of the initialization parameters, highlight one and click the up or down arrows to move it to a new location.

Firing parameters

The firing parameters contain values passed from the trigger point in the application to the Java rule implementor at run time. You can alter the parameters coming from the application before passing them to the Java rule implementor. Typically these parameters are left unaltered.

For example, when implementing the ″isSeniorCitizen″ rule that determines whether a person is a senior citizen, you might want to pass a person from the application to the rule as the target object. However, the ″isSeniorCitizen″ rule uses the RuleGreaterThanEqual Java rule implementor, which requires that you pass an integer value. You can alter the firing parameters to specify that the method ″getAge″ is called on the person object and pass that result to the Java rule implementor.

You must choose one of the following:

Pass firing parameters from trigger point unchanged
The parameters specified in the trigger point of the application are passed to the Java rule implementor without being altered. This is the default value.

Specify firing parameters
The values specified in the table are passed to the Java rule implementor.

To add a new firing parameter to the list, click **Add** and fill in the fields in the Add Firing Parameter window.

To edit an existing firing parameter, highlight the firing parameter, click **Change**, and fill in the fields in the Change Firing Parameter window.

To delete a parameter, highlight the firing parameter and click **Delete**.

To change the order of the firing parameters, highlight one and click the up or down arrows to move it to a new location.

Add Firing Parameter window

Use the Add Firing Parameter window to add a firing parameter to a rule. The firing parameters contain values passed from the trigger point in the application to the Java rule implementor at run time. You can alter the parameters coming from the application before passing them to the Java rule implementor. Typically, these parameters are left unaltered.

To add a firing parameter, enter a field description, select an appropriate option, and click **Add**. The parameter is added and the window remains open to specify additional parameters. When you finish specifying initialization parameters, click **Close**. The following fields and options are available in this window:

Description
: Use the **Description** field to type a description of the firing parameter.

Specify a type and value
: Use the **Specify a type and value** option to specify a constant value to pass to the Java rule implementor.

Get value from method call
: Use the **Get value from method call** option to call the specified method on the target object.

Get value from trigger point firing parameters
: Use the **Get value from trigger point firing parameters** option to get a specific value from the firing parameters that were specified in the trigger method in the application. An index to the original firing parameter must be specified. This index starts with 0; thus, specify the value 0 to pass the first firing parameter. Specify the trigger point parameter number.

Change Firing Parameter window

Use the Change Firing Parameter window to edit an existing firing parameter. The firing parameters contain values passed from the trigger point in the application to the Java rule implementor at run time. You can alter the parameters coming from the application before passing them to the Java rule implementor. Typically, these parameters are left unaltered.

To change a firing parameter, enter a field description, select an appropriate option, and click **Change**. The parameter is changed and the window remains open to specify additional parameters. When you finish specifying initialization parameters, click **Close**.

The following fields and options are available in this window:

Description
: Use the **Description** field to type a description of the firing parameter. There are three types of firing parameters that can be specified.

Specify a type and value
: Use the **Specify a type and value** option to specify a constant value to pass to the Java rule implementor.

Get value from method call
: Use the **Get value from method call** option to call the specified method on the target object.

Get value from trigger point firing parameters
: Use the **Get value from trigger point firing parameters** option to pass a specific value from the firing parameters that were specified on the trigger method in the application. An index to the original firing parameter must be specified. This index starts with 0; thus, specify the value 0 to pass the first firing parameter.

Rule properties window: Description tab

Use the **Rule properties window: Description** tab to define the purpose and intent of the rule. All of the fields in this panel are optional and none are used by the Business Rule Beans run-time environment. The following fields are available on this tab:

Business Intent: Use the **Business Intent** field to describe the business intent of this rule.

Description: Use the **Description** field to define a general description of the rule and its purpose.

Original requirement: Use the **Original requirement** field to compose a description of the original requirement that created this rule.

Rule properties window: Dependent Rules tab

Use the **Rule properties window: Dependent Rules** tab to specify the rules that the newly created rule will depend upon. To add names to the **Dependent rule names** field, proceed as follows:

1. Locate the dependent rule. You can do this in one of two ways:

   Browse for a rule

   If you are familiar with the location of the dependent rule, then click the **Browse** button. Navigate to the rule's location and highlight it.

   Find a rule

   If you are unfamiliar with the location of the rule, then click the **Find** button. This will launch a Find Rules window in which you can specify options and then initiate a search. When you have located the rule, highlight it.
2. Click the **Add** button.

To delete a rule from the **Dependent rule names** field, highlight the rule and click **Delete**.

To change the order of the rules, highlight a rule and click the Up or Down arrows to move it to a new location.

Rule properties window: Other tab

Use the **Rule properties window: Other** tab to establish precedence and enter information that is relevant to you, but does not fit into any other category. The following fields are available in this window:

Precedence: Use the **Precedence** field to specify the relative priority when firing the rule. This value is used to order the rules from lowest to highest.

User defined data: Use the **User defined data** field to enter any additional text that you want to store. The Business Rule Beans run-time environment does not use this field.

Edit menu

This menu describes the options available on the Edit menu window. The following options are available in this window:

Cut

Use the **Cut** option to move rules and rule folders.

Copy

Use the **Copy** option to copy rules and rule folders. The following tasks can be accomplished using the **Copy** option:

- **Copying rules**

  Copy or move a rule from one folder to another by either cutting and pasting it or dragging and dropping it.

Cutting and pasting
Use menu commands (**Edit > Copy**, **Edit > Cut** and **Edit > Paste**) or keyboard commands (**CTRL-C**, **CTRL-V** and **CTRL-X**) to copy or move a rule.

Dragging and dropping
Highlight the rule you want to copy. Then, press and hold the right mouse button, drag the cursor to the target location, and release. Select **Copy** or **Move** from the list.

Note: A rule also can be copied so that the copy replaces the existing rule at a specified date. This is referred to as a Quick Copy.

- **Copying rule folders**

  Copy or move a rule folder and all its contents by either cutting and pasting it or dragging and dropping it.

  Cutting and pasting
  Use menu commands (**Edit > Copy**, **Edit > Cut** and **Edit > Paste**) or keyboard commands (**CTRL-C**, **CTRL-V** and **CTRL-X**) to copy or move a rule.

  Dragging and dropping
  Press and hold the right mouse button on the folder to be copied. Drag the cursor to the target location and release the mouse button. Select **Copy** or **Move** from the list.

Paste
Use the **Paste** option to add cut or copied rules and rule folders.

Find
Use the **File** option to search for a rule. A window opens in which you can specify your search criteria. If you would like to search a specific folder, then right-click it and select **Find** from the list. For more information, see “Find Rules window” on page 204.

- **Search the whole directory**

  To search the whole directory, complete the following steps:
  1. Click **Edit > Find** from the main menu of the Rule Browser.
  2. Determine your search criteria in the Find Rules window.
- **Search a specific folder**

  To search a specific folder, complete the following steps:
  1. Right-click the folder and select **Find** from the list.
  2. Determine your search criteria in the Find Rules window.

The results of your search are displayed in a Search Results window.

Quick Copy
Use the **Quick Copy** option to make a copy of a rule that will replace the existing one on a specified date. You can modify the copy slightly so that a new value goes into effect on the desired date and time while the old rule expires. For more information, see the **Quick Copy window** link in Related reference.

Select All
Use the **Select All** option to facilitate rule selection.

Deselect All
Use the **Deselect All** option to deselect rules.

Quick Copy window

Use the Quick Copy window to make a copy of a rule that replaces the existing one on a specified date.

For example, suppose that you have an ″isSeniorCitizen″ rule. Currently, a person is considered a senior citizen if they are 62 years of age or older. Starting on January 1, 2002, you must change this to 65. Use Quick Copy to specify the new date and to change the age from 62 to 65. The current rule is set to expire on the same date that the new rule takes effect. The new senior citizen age is defined as 65.

Note: Use the **Quick Copy** function for simple changes only.

In the following fields, specify how the copy differs from the original:

Start Date For New Rule: Use the **Start Date For New Rule** field to enter the date that the new rule replaces the existing rule. Use the date and time format that is shown. For example, the format might be the following:

Usage scenario

M/d/yy h:mm a.

Change parameter values for new rule: Use the **Change parameter values for new rule** field to add new parameter values.

View menu

This article describes the options available in the View menu window. The following options are available in this window:

Status Bar

Use the **Status Bar** option to toggle the status bar on or off. The status bar is shown along the bottom of the Rule Browser window.

Specify Columns

Use the **Specify Columns** option to adjust the type and order of the columns that display in your window.

In the window that opens, the following tasks can be accomplished:

Add a new column
: Select one or more entries in the **Available columns** list and click the **Add** button. The selected entries are added to the end of the **Columns displayed** list.

Remove a column
: Select one or more entries in the **Columns displayed** and click the **Remove** button. The selected entries are added to the end of the **Available columns** list.

Reorder columns
: Select one or more entries in the **Columns displayed** list. To move the entries towards the top of the list, click the Up arrow; to move the entries towards the bottom, click the Down arrow.

Specify Date/Time Format

Use the **Specify Date/Time Format** option to adjust the format used when displaying dates and times. For more information, click the **Specify Date/Time Format window** link under Related reference.

Refresh

Use the **Refresh** option to update the contents of the folder hierarchy and the rule table.

Specify Date/Time Format window

Use the Specify Date/Time Format window to change the date and time format. Choose one of the following radio button options and then click **OK**:

Use default format for this locale: Use the **Use default format for this locale** option to adjust the date and time format to match the default setting of your current locale.

Select a predefined format for the date and time: Use the **Select a predefined format for the date and time** option to select one of several existing date and time formats.

Specify a custom format for the date and time: Use the **Specify a custom format for the date and time** option to determine your own format for your date and time display. Choose one of the date and time formats from the two menus.

Find Rules window

Use the Find Rules window to specify search criteria to locate rules. The search combines your queries using a logical ″AND″ operation. For example, if you specify both a folder name and a rule name, the search finds rules that match both the folder name and rule name. It displays the results in a Search Results window.

Note: All of the fields in this window are optional.

Main menu

The main menu has the following options:

- **File**

  Save As
  Use the **Save as** option to open a Save Search window and store the current search criteria.

  Open Use the **Open** option to open the Open Saved Search window and load a previously saved set of search criteria into the Find Rules window.

  Close Use the **Close** option to close the Find Rules window.

- **View**

  Show Search
  Use the **Show Search** option to display a text description of your search criteria on one screen.

  Specify Date/Time Format
  Use the **Specify Date/Time Format** option to adjust the format used when displaying dates and times. For more information, see “Specify Date/Time Format window”.

Tabs

The following tabs are displayed in this window:

Name Use the **Name** tab to specify the basic search criteria. For more information, “Find Rules window: Name tab”.

Date Use the **Date** tab to specify the date-related search criteria. For more information, see “Find Rules window: Date tab” on page 206.

Classification
Use the **Classification** tab to specify the search criteria related to a rule’s classification. For more information, see “Find Rules window: Classification tab” on page 206.

Implementation
Use the **Implementation** tab to specify the search criteria that is based on the manner in which the rule is implemented. For more information, see “Find Rules window: Implementation tab” on page 207.

Description
Use the **Description** tab to specify the text-based search criteria related to a rule’s description. For more information, see “Find Rules window: Description tab” on page 207.

Other Use the **Other** tab to specify the search criteria based on precedence and user-defined data. For more information, see “Find Rules window: Other tab” on page 208.

Find Rules window: Name tab

Use the **Find Rules window: Name** tab to specify the basic search criteria. The following fields and options are available on this tab:

Name: Use the **Name** field to search for a specific rule name.

Note: This is case-sensitive.

Drop-down search option list: Use the one of the following options in the **Drop-down search option list** to narrow your search:

equal Use the **equal** selection to look for an *exact* match.

starting with
Use the **starting with** selection to find rules whose name *starts* with the specified value.

ending with
Use the **ending with** selection to find rules whose name *ends* with the specified value.

containing
Use the **containing** selection to find rules that *contain* the specified value.

Location: Use the **Location** field to specify the folder that you want to search. Click the **ellipses** button to the right of the field if you want to browse for the folder.

Note: The folder names are case-sensitive.

Include subfolders: Select **Include subfolders** to include the folder’s subfolders in the search.

Status: Use the **Status** menu to specify a search criteria that is based on a rule’s availability.

Find Rules window: Date tab

Use the **Find Rules window: Date** tab to specify date-related search criteria. A rule always has a start date and a range of time in which it is in effect. The end date is optional and if it is not specified, the rule never expires. The following options are available on this tab:

Find Rules for any date: Use the **Find Rules for any date** option to remove the date from consideration in the search criteria.

Find Rules that are: Use the **Find Rules that are** option to search for rules in one of the following states:

currently in effect
The **currently in effect** selection finds rules that are active at this point in time.

scheduled
The **scheduled** selection finds rules that go into effect at a future date.

expired
The **expired** selection finds previously active rules that are beyond the rule's end date.

Find Rules: Use the **Find Rules** option to specify the dates you want to search. Modify the following criteria to narrow your search:

1. Select one of the following rule states from the menu:
   - in effect
   - starting
   - ending
2. Select **Query on date only** if you would like your search to ignore time-specific information.
3. Select one of the following three methods by which to search:

   on *x* Select the **on *x*** option to find rules that are in the desired state (as chosen in Step 1) on the specified date. If the **Query on date only** check box is clear, then midnight is used for the time.

   anytime between *x* and *y*
   Select the **anytime between *x* and *y*** option to find rules that are in the desired state (as chosen in Step 1) *anytime* between the given dates. Specify a start date (represented by *x*) and an end date (represented by *y*). If the **Query on date only** check box is clear, then midnight of each day is used for the time.

   anytime during the next *x* days
   Select the **anytime during the next *x* days** option to find rules that are in the desired state (as chosen in Step 1) within this period of time and specified in days.

Find Rules window: Classification tab

Use the **Find Rules window: Classification** tab to specify search criteria related to a rule's classification. The following options are available on this tab:

Show all: Use the **Show all** option if you do not want to include the classification information in the search criteria.

Show rules that are not classified and do not perform classification: Use the **Show rules that are not classified and do not perform classification** option to find rules that you do not need to classify.

Show rules that perform classification: Use the **Show rules that perform classification** option to find rules that return a classification such as ″Gold″, ″Silver″, or ″Bronze″.

Show rules that are classified: Use the **Show rules that are classified** option to find rules that are classified with the specified classification. The specified classification is case-sensitive. Enter a specific classification into the field that is provided.

Find Rules window: Implementation tab

Use the **Find Rules window: Implementation** tab to specify search criteria that is based on the manner in which the rule is implemented. The following options are available on this tab:

Java rule implementor: Use the **Java rule implementor** option to search for rules that use the specified Java rule implementor. You can use one of the values in the list or type in your own.

Firing location: Use the **Firing location** option to search for the location in which the rule implementor is run. Choose one of the following values from the check box:

Local
Use the **Local** option to search locally for the location in which the rule implementor is run.

Remote
Use the **Remote** option to search the server for the location in which the rule implementor is run.

Anywhere
Use the **Anywhere** option to search both locally and on the server for the location in which the rule implementor is run.

Firing parameters: Use the **Firing parameters** option to search for rules that alter the firing parameters passed from the trigger point to the Java rule implementor. For more information, see the Add Firing Parameter window..

The following selections are available for the firing parameter option:
- show rules that alter firing parameters
- show rules that do not alter firing parameters

Find Rules window: Description tab

Use the **Find Rules window: Description** tab to specify text-based search criteria related to a rule’s description. The following fields and options are available on this tab:

Business intent: Use the **Business intent** field to search for rules with a given business intent.

Description: Use the **Description** field to search for rules with a given description.

Original requirement: Use the **Original requirement** field to search for rules with a given original requirement.

Drop-down search option list: Use the following options in the **Drop-down search option list** to narrow your search in the **Business intent**, **Description**, and **Original requirements** fields:

equal Use the **equal** selection to look for an *exact* match.

starting with
Use the **starting with** selection to find rules whose name *starts* with the specified value.

ending with
Use the **ending with** selection to find rules whose name *ends* with the specified value.

containing
Use the **containing** selection to find rules that *contain* the specified value.

Find Rules window: Other tab

Use the **Find Rules window: Other** tab to specify search criteria based on precedence and user-defined data. The following options are available on this tab:

Precedence: Use the **Precedence** option to search for rules with given precedence. The precedence is an integer value that specifies the relative priority of this rule when it is fired.

Drop-down search option list: Use the **Drop-down search option list** to narrow your search by using one of the following options:

- equal
- less than
- less than or equal
- greater than
- greater than or equal
- not equal

User defined data: Use the **User defined data** option to search for rules with given user defined data.

Drop-down search option list: Use one of the following options from the **Drop-down search option list**:

equal Use the **equal** selection to look for an *exact* match.

starting with
Use the **starting with** selection to find rules whose name *starts* with the specified value.

ending with
Use the **ending with** selection to find rules whose name *ends* with the specified value.

containing
Use the **containing** selection to find rules that *contain* the specified value.

Search Results window

This window contains the results of a search from a Find Rules window. It is virtually identical to the Rule Browser in terms of form and function. Use the options in the main menu to perform many of the same administrative actions, with a few additions:

The following options are available in this window:

File menu:

Save As
Use the **Save As** option to open a Save Search window and store the current search criteria.

Open Containing Folder
Use the **Open Containing Folder** option to open a Rule Browser window and display the contents of the folder in which the selected rule resides.

Close Use the **Close** option to close the Find Rules window.

View menu:

Show Search
Use the **Show Search** option to display a text description of your search criteria on one screen.

Specify Date/Time Format
Use the **Specify Date/Time Format** option to adjust the format used when displaying dates and times. For more information, see the **Specify Date/Time Format window** link in the Related reference.

Refresh
Use the **Refresh** option to reissue the search and repopulate the table with the updated search results.

Save Search window

Use the Save Search window to store the current search criteria for later retrieval. Follow the following steps to save your search:

1. Type in a name for your search or replace a previously saved search from the list.
2. Click **OK**.

The saved search criteria is loaded into the Find Rules window.

Open Saved Search window

Use the Open Saved Search window to open a previously saved search. Follow these steps to open your saved search:

1. In the Select a Search window, highlight the name of the search you want to open.
2. Click **OK**.

The saved search criteria is loaded into the Find Rules window.

Business rule beans: Resources for learning

Use the following links to find relevant supplemental information about business rule beans. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- "Planning, business scenarios, and IT architecture"
- "Programming instructions and examples"
- "Administration"

Planning, business scenarios, and IT architecture
- Delivering new business value to the enterprise on a J2EE and Web services base (Update)

  http://www7b.software.ibm.com/wsdd/library/summaries/200462.html

  This paper, in PDF form, describes the strategy behind the IBM extensions to J2EE and Web services functionality in the IBM WebSphere Application Server Enterprise, Version 4.0. It explains Enterprise Services, business rule beans, message beans and JMS listener, internationalization, shared work areas, bidirectional CORBA connectivity, the ActiveX bridge, and the C++ CORBA SDK.
- WebSphere Application Server V5.0 Architecture and Overview

  http://developerworks.cybercentral.com/ibm0502/amt/ibmpresentations/683_1.pdf

  This is an IBM developerWorks presentation that provides an overview of the functionality available with IBM WebSphere Application Server, Version 5

Programming instructions and examples
- Message-Driven Beans and Encapsulated Business Rules

  http://www2.theserverside.com/resources/article.jsp?l=Message-Driven-Beans-And-Encapsulated-Business-Rules

  This article describes how to use business rules with Message-driven Beans.
- WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide

  http://www.redbooks.ibm.com/redbooks/SG246504.html

  Chapter three of this programmer's guide provides information about implementation, modification, and deployment of business rules.
- WebSphere Application Server Enterprise Edition Technology Sample

  http://www7b.software.ibm.com/wsdd/downloads/ee41_landing.html

  This technology sample enables developers to gain experience with the Business Rule Beans technology.

Administration
- IBM WebSphere Administration

  http://books.mcgraw-hill.com/cgi-bin/pbg/0072223154.html
- Listing of all IBM WebSphere Application Server Redbooks

  http://publib-b.boulder.ibm.com/Redbooks.nsf/Portals/WebSphere

  This is a listing of the Redbook publications about the WebSphere software platform.
- WebSphere Application Server Version 4.0 Enterprise Edition -- Presentations and Labs

  http://www7b.boulder.ibm.com/wsdd/library/presents/WAS_EE_Training.html

Chapter 7. Using asynchronous beans

The asynchronous beans feature adds a new set of APIs that enable J2EE applications to run work asynchronously inside a IBM WebSphere Application Server Enterprise. This topic provides a brief overview of the tasks involved in using asynchronous beans. For a more detailed description of the asynchronous beans model, review the conceptual topic “Asynchronous beans”.

Steps for this task

1. “Configuring work managers” on page 214
2. “Assembling applications that use work managers” on page 219
3. “Developing work objects to run code in parallel” on page 220
4. “Developing event listeners” on page 222
5. “Developing Asynchronous scopes” on page 225

Asynchronous beans

An asynchronous bean is a Java object or enterprise bean that can be executed asynchronously by a J2EE application, using the J2EE context of the bean’s creator.

Asynchronous beans can improve performance by enabling a J2EE program to decompose operations into parallel tasks. Asynchronous beans enable the construction of stateful, ″active″ J2EE applications. These applications address a segment of the application space that J2EE has not previously addressed (that is, advanced applications that require application threading, active agents within a server application, or distributed monitoring capabilities).

Asynchronous beans can run using the J2EE security context of the creator J2EE component. These beans also can run with copies of other J2EE contexts. For example:

- Internationalization context
- Application profiles
- Work areas
- Access intent policies

Asynchronous bean interfaces

There are three types of asynchronous bean:

Work object
A work object implements the com.ibm.websphere.asynchbeans.Work interface. A work object runs parallel to its caller using the WorkManager.startWork() method. Applications implement work objects in order to run code blocks asynchronously. For more information on the Work interface, see the Related reference section at the end of this article.

Alarm listener
An alarm listener is an object that implements the com.ibm.websphere.asynchbeans.AlarmListener interface. Alarm listeners

are called when a high-speed transient alarm expires. For more information on the AlarmListener interface, see the Related reference section at the end of this article.

Event listener
An event listener can implement any interface. An event listener is a lightweight, asynchronous notification mechanism for asynchronous events within a single JVM. An event listener would typically be used to enable J2EE components within a single application to notify each other about various asynchronous events.

Supporting interfaces

Work manager
A work manager is a thread pool that administrators create for J2EE applications. The administrator specifies the properties of the thread pool and a policy that determines which J2EE contexts the asynchronous bean will inherit.

Event source
An event source implements the com.ibm.websphere.asynchbeans.EventSource interface. An event source is a system-provided object that supports a generic, type-safe asynchronous notification server within a single JVM. The event source enables event listener objects, which implement any interface, to be registered. For more information on the EventSource interface, see the Related reference section at the end of this article.

Event source events
Every event source can generate events of its own. Event sources also can generate their own events such as 'listener count changed'. An application can register an event listener object that implements com.ibm.websphere.asynchbeans.EventSourceEvents. This enables the application to catch events such as listeners being added or removed, or a listener throwing an unexpected exception. For more information on EventSourceEvents, see the Related reference section at the end of this article.

Additional interfaces, including alarms and subsystem monitors, are introduced in the topic "Developing Asynchronous scopes" on page 225, which discusses some of the advanced applications of asynchronous beans.

Transactions

Every asynchronous bean method is called using its own transaction, much like container-managed transactions in a typical enterprise bean. It is very similar to the situation when an EJB method is called with TX_NOT_SUPPORTED. The run-time environment starts a local transaction before invoking the method. The asynchronous bean method is free to start its own global transaction if this is possible for the calling J2EE component. For example, if an enterprise bean creates the component, the method that creates the asynchronous bean must be TX_BEAN_MANAGED.

If the asynchronous bean method throws an exception, any local transactions are rolled back. If the method returns normally, any incomplete local transactions are completed according to the unresolved action policy configured for the bean. EJB methods can configure this policy using their deployment descriptor. If the asynchronous bean method starts its own global transaction and does not commit this global transaction, the transaction is rolled back when the method returns.

Access to J2EE component meta-data

If an asynchronous bean is a J2EE component, such as a session bean, its own meta-data is active when a method is called. If an asynchronous bean is a simple Java object, the J2EE component metadata of the creating component is available to the bean. Like its creator, the asynchronous bean can look up the java:comp namespace. This enables the bean to access connection factories and enterprise beans, just as it would if it were any other J2EE component. The environment properties of the creating component also are available to the asynchronous bean.

The java:comp namespace is identical to the one available to the creating component and the same restrictions apply. For example, the java:comp/UserTransaction object is only available if the creating enterprise bean was TX_BEAN_MANAGED. All of the connection factories use the same resource-sharing scope as the creating component.

Connection management

An asynchronous bean method can use the connections that its creating J2EE component obtained using java:comp resource references. (For more information on resource references, see References). However, the bean method must access those connections using a get, use, close pattern. There is no connection caching between method calls on an asynchronous bean. The connection factories or DataSources themselves can be cached, but the connections must be retrieved on every method call, used, and then closed. While the asynchronous bean method can look up connection factories using a global JNDI name, this is not recommended for the following reasons:

- The JNDI name is hard-coded in the application (for example, as a property or string literal).
- The connection factories are unshared because there is no way to specify a sharing scope.

For code examples that demonstrate both the correct and the incorrect ways to access connections from asynchronous bean methods, see the topic “Example: Asynchronous bean connection management”.

Example: Asynchronous bean connection management

An asynchronous bean method can use the connections that its creating J2EE component obtained using java:comp resource references. (For more information on resource references, see the topic References.) The following is an example of an asynchronous bean that uses connections correctly:

```
class GoodAsynchBean
{
 DataSource ds;
 public GoodAsynchBean()
  throws NamingException
 {
  // ok to cache a connection factory or datasource
  // as class instance data.
  InitialContext ic = new InitialContext();
  // we are assuming that the creating J2EE component has this
  // resource reference defined in its deployment descriptor.
  ds = (DataSource)ic.lookup("java:comp/env/jdbc/myDataSource");
 }
 // When the asynchronous bean method is called, get a connection,
 //  use it, then close it.
 void anEventListener()
 {
  Connection c = null;
  try
  {
   c = ds.getConnection();
   // use the connection now...
  }
  finally
  {
   if(c != null) c.close();
  }
 }
}
```

The following is an example of an asynchronous bean that uses connections illegally:

```
class BadAsynchBean
{
 DataSource ds;
 // Do not do this. You cannot cache connections across asynch method calls.
 Connection c;

 public BadAsynchBean()
  throws NamingException
 {
  // ok to cache a connection factory or datasource as
  // class instance data.
  InitialContext ic = new InitialContext();
  ds = (DataSource)ic.lookup("java:comp/env/jdbc/myDataSource");
  // here, you broke the rules...
  c = ds.getConnection();
 }
 // Now when the asynch method is called, illegally use the cached connection
 // and you'll likely see a bunch of J2C related exceptions at runtime.
 // close it.
 void someAsynchMethod()
 {
  // use the connection now...
 }
}
```

Configuring work managers

Before you begin

If you are unfamiliar with work managers, review the conceptual topic "Work managers" on page 215.

A work manager acts as a thread pool for application components that use asynchronous beans. Use the administrative console to configure work managers. You can define multiple work managers for each cell. Each work manager is bound to a unique place in JNDI.

Note: The work manager service is only supported from within the EJB Container or Web Container (EJBs or Servlets). Looking-up and using a configured WorkManager from a J2EE Application Client container is not supported.

Steps for this task

1. Start the administrative console.

2. Select **Resources > Work Managers**.
3. Click **New**.
4. Specify the following required properties:

   Name The display name for the work manager.

   JNDI Name
   The JNDI name for the work manager. This name is used by asynchronous beans that need to look up the work manager. Each work manager must have a unique JNDI name within the cell.

   Number of Alarm Threads
   The maximum number of threads to be used for processing alarms. There is a single thread that is used to monitor pending alarms and dispatch them. Also there is an additional pool of threads that is used for dispatching the threads. All alarm managers on the asynchronous beans associated with this work manager share this set of threads. That is, there is a single alarm thread pool for each work manager and all of the asynchronous beans associated with the work manager share this pool of threads.

   Minimum Number Of Threads
   The initial number of threads to be created in the thread pool.

   Maximum Number Of Threads
   The maximum number of threads to be created in the thread pool. The maximum number of threads can be exceeded temporarily if the **Growable** checkbox is selected. These additional threads are discarded when the work on the thread completes.

   Thread Priority
   The order of priority for threads available in the thread pool.
5. [Optional] Specify a **Description** and a **Category** for the work manager.
6. [Optional] Select the **Service Names** (J2EE contexts) on which you want this work manager to be made available. Any asynchronous beans that use this work manager then will inherit the selected J2EE contexts from the component that creates the bean. The list of selected services also is known as the ″sticky″ context policy for the work manager.

   Note: Selecting more services than are actually required might impede performance.
7. Save your configuration.

Results

The work manager is now configured and ready to be accessed by application components that need to manage asynchronous code execution.

Work managers

A work manager is a thread pool created for J2EE applications that use asynchronous beans.

Using the administrative console, an administrator can configure any number of work managers. The administrator specifies the properties of the work manager, including the ″sticky″ context (inheritance) policy for any asynchronous beans that use the work manager. The administrator binds each work manager to a unique place in JNDI.

When writing a Web or EJB component that uses asynchronous beans, the developer should include a resource reference in each component that needs access to a work manager. (For more information on resource references, see the topic References.) The component looks up a work manager using a logical name in the component's java:comp namespace, just as it would look up a datasource, enterprise bean, or connection factory.

The deployer binds physical work managers to logical work managers when the application is deployed.

For example, if a developer needs three thread pools to partition work between bronze, silver, and gold levels, the developer writes the component to pick a logical pool based on an attribute in the client application's profile. The deployer has the flexibility to decide how to map this request for three thread pools. The deployer might decide to use a single thread pool on a small machine. In this case, the deployer binds all three resource references to the same work manager instance (that is, the same JNDI name). A larger machine might allow for three thread pools, so the deployer binds each resource reference to a different work manager. Work managers can be shared between multiple J2EE applications installed on the same server.

An application developer can use as many logical work managers as necessary; the deployer chooses whether to map one physical work manager or several to the logical work manager defined in the application.

Note: All J2EE components that need to share asynchronous scope objects must use the same work manager. These scope objects have an affinity with a single work manager so an application that uses AsynchScopes should verify that all of the components using scope objects use the same work manager.

When multiple work managers are defined, the underlying thread pools are created in a JVM only if an application within that JVM looks up the work manager. For example, there might be ten thread pools (work managers) defined, but none are actually created until an application looks them up.

How to look up a work manager

An application can look up a work manager as follows. Here, the component contains a resource reference named wm/myWorkManager, which was bound to a physical work manager when the component was deployed:

```
InitialContext ic = new InitialContext();
WorkManager wm = (WorkManager)ic.lookup("java:comp/env/wm/myWorkManager");
```

″Sticky″ J2EE contexts

Asynchronous beans can inherit the following J2EE contexts. In other words, these contexts can be made ″sticky″:

Internationalization context

Work area

Application profile

Security
The asynchronous bean can be run as anonymous or as the client authenticated on the thread that created it. This is useful because the

asynchronous bean can do only what the caller can do. This is more useful than a RUN_AS mechanism, for example, which prevents this kind of behavior.

Component meta-data
: Component meta-data is relevant only when the asynchronous bean is a simple Java object. If the bean is a J2EE component, such as an enterprise bean, the component's meta-data is active.

Which contexts are sticky depends on the work manager used by the application that creates the asynchronous bean. Using the administrative console, the administrator defines the sticky context policy of a work manager by selecting the services on which the work manager is to be made available.

Work manager collection

Use this page to view the configuration properties of work managers.

A work manager contains a pool of threads bound into JNDI.

To view this administrative console page, click **Resources > Work Managers** .

Name

The name by which the work manager is known for administrative purposes.

Data type
: String

JNDI Name

The JNDI name used to look up the work manager in the namespace.

Data type
: String

Description

A description of this work manager for administrative purposes.

Data type
: String

Category

A string that can be used to classify or group this work manager.

Data type
: String

Number of Alarm Threads

The number of threads used to execute concurrent alarms.

Data type
: Integer

Minimum Number of Threads

The minimum number of threads available in this work manager for running works.

Data type
: Integer

Maximum Number of Threads

The maximum number of threads available in this work manager for running works.

Data type
Integer

Thread Priority
The priority of the threads available in this work manager

Data type
Integer

Growable
Specifies whether the number of threads in this work manager can be increased.

Service Names
A list of service names on which this work manager is made available.

The context information of each selected service is propagated to each work or alarm that is created using this work manager. Selecting services that are not needed can negatively impact performance.

Work manager settings
Use this page to modify work manager settings.

A work manager contains a pool of threads bound into JNDI.

To view this administrative console page, click **Resources > Work Managers >** *workmanager_name*.

Name: The name by which the work manager is known for administrative purposes.

Data type
String

JNDI Name: The JNDI name used to look up the work manager in the namespace.

Data type
String

Description: A description of this work manager for administrative purposes.

Data type
String

Category: A string that can be used to classify or group this work manager.

Data type
String

Number of Alarm Threads: The number of threads used to execute concurrent alarms.

Data type
Integer

Minimum Number of Threads: The minimum number of threads available in this work manager for running works.

Data type
Integer

Maximum Number of Threads: The maximum number of threads available in this work manager for running works.

Data type
: Integer

Thread Priority: The priority of the threads available in this work manager

Data type
: Integer

Growable: Specifies whether the number of threads in this work manager can be increased.

Service Names: A list of service names on which this work manager is made available.

The context information of each selected service is propagated to each work or alarm that is created using this work manager. Selecting services that are not needed can negatively impact performance.

Work manager service settings

Use this page to enable or disable the work manager service, which manages work manager resources used by the server.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Work Manager Service** .

Startup

Specifies whether the server will attempt to start the work manager service.

Default
: Selected

Range

Selected
: When the application server starts, it attempts to start the work manager service automatically.

Cleared
: The server does not try to start the work manager service. If work manager resources are to be used on this server, the system administrator must start the work manager service manually or select this property then restart the server.

Assembling applications that use work managers

Before you begin

Your administrator needs to configure at least one work manager using the administrative console.

If your application references one or more logical work managers, the logical work managers must be bound to one or more physical work managers using the Application Assembly Tool.

Steps for this task

1. Declare a resource reference for each work manager (required action by the application developer). This forms an EAR file. (For more information on resource references, see the topic References.)
2. Using the Application Assembly Tool (AAT), bind each resource reference to a physical work manager.
3. Add a resource reference with the type ’com.ibm.websphere.asynchbeans.WorkManager’ to the application’s descriptor.

   The application then can look up this work manager using its resource reference name in java:comp. The AAT or WebSphere Studio Application Developer Integration Edition (WSAD-IE) then can specify which resource references are bound to a physical work manager.

   Note: The previous process is the same as the process used for DataSources.

Developing work objects to run code in parallel

Before you begin

Your administrator must have configured at least one work manager using the administrative console.

To run code in parallel, or in a different J2EE context, wrap the code in a work object.

Steps for this task

1. Create a work object.

   A work object implements the com.ibm.websphere.asynchbeans.Work interface. For example:

   ```
   class SampleWork implements Work
   ```
2. Determine the number of work managers needed by this application component.
3. Look up the work manager or managers using the work manager’s resource reference (or logical name) in the java:comp namespace. (For more information on resource references, see the topic References.)

   ```
   InitialContext ic = new InitialContext();
   WorkManager wm = (WorkManager)ic.lookup("java:comp/env/wm/myWorkManager");
   ```

   The resource reference for the work manager (in this case, wm/myWorkManager) must be declared as a resource reference in the application’s deployment descriptor.
4. Call the WorkManager.startWork() method using the work object as a parameter.

   For example:

   ```
   Work w = new MyWork(...);
   WorkItem wi = wm.startWork(w);
   ```

   The startWork() method can take a startTimeout parameter. This specifies a hard time limit for the Work object to be started.

   The startWork() method returns a work item object. This object is a handle that provides a link from the component to the now running work object.
5. [Optional] If your application component needs to wait for one or more of its running work objects to complete, call the WorkManager.join() method.

For example:

```
WorkItem wiA = wm.start(workA);
WorkItem wiB = wm.start(workB);
ArrayList l = new ArrayList();
l.add(wiA);
l.add(wiB);
if(wm.join(l, wm.JOIN_AND, 5000)) // block for up to 5 seconds
{

// both wiA and wiB finished
}
else
{

// timeout

// we can check wiA.getStatus or wiB.getStatus to see which, if any, finished.
}
```

This method takes an array list of work items that your component wants to wait on and a flag that indicates whether the component will wait for one or all of the work objects to complete. You also can specify a timeout value.

6. Using the release() method, set a variable in a synchronized block.

   For example:

```
public synchronized void release()
{
 released = true;
}
```

The Work.run() method should periodically examine this variable to check whether the loop should exit or not.

Work objects

A work object is a type of asynchronous bean used by application components to run code in parallel or in a different J2EE context.

A work object implements the com.ibm.websphere.asynchBeans.Work interface. A work object is essentially a java.lang.Runnable object that is serializable and provides additional methods. For details, see the Interface Work in the Javadoc, which is located in the InfoCenter.

A component wanting to run work in parallel, or in a different J2EE context, locates a work manager in JNDI, then calls the WorkManager.startWork() method using the work object as a parameter.

The startWork() method returns a work item object. This object is a handle that provides a link from the component to the now running work object. The work item object is typically used when the component needs to wait for one or more of its running work objects to complete. The WorkManager.join() method takes an array list of work items that the component wants to wait on, and a flag indicating whether the component will wait for all or one of the work objects to complete. A timeout can be specified, which prevents the component from waiting indefinitely.

Why not have the application simply create Java 2 SDK threads? The threads created by the Java 2 SDK are not managed threads and hence know nothing about the J2EE environment and are unusable inside an application server. In addition, these threads have no J2EE context (for example, a java:comp) and are not authenticated when they fire. Work object threads, on the other hand, are fully supported by the application server and have the same properties as any other asynchronous bean.

Example: Work object

The following is an example of a work object that dynamically subscribes to a topic:

```
class SampleWork implements Work
{
 boolean released;
 Topic targetTopic;
 EventSource es;
 TopicConnectionFactory tcf;
 public SampleWork(TopicConnectionFactory tcf, EventSource es, Topic targetTopic)
 {
  released = false;
  this.targetTopic = targetTopic;
  this.es = es;
  this.tcf = tcf;
 }
 synchronized boolean getReleased()
 {
  return released;
 }
 public void run()
 {
  // setup our JMS stuff.
  TopicConnection tc = tcf.createConnection();
  TopicSession sess = tc.createSession(false, Session.AUTOACK);
  tc.start();

  MessageListener proxy = es.getEventTrigger(MessageListener.class, false);
  while(!getReleased())
  {
   // block for up to 5 seconds.
   Message msg = sess.receiveMessage(5000);
   if(msg != null)
  }
  tc.close();
 }
 // called when we want to stop the Work object.
 public synchronized void release()
 {
  released = true;
 }
 }
```

As a result, any component that has access to the event source can add an event on demand, which allows components to subscribe to a topic in a more scalable way than by simply giving each client subscriber its own thread. The previous example is fully explored in the WebSphere Trader sample. See your Samples Gallery for details.

Developing event listeners

Application components that listen for events can use the EventSource.addListener() method to register an event listener object (a type of asynchronous bean) with the event source to which the events will be published. An event source also can fire events in a type-safe manner using any interface.

Notifications between components within a single EAR file are handled by a special event source. See the topic, “Using the application notification service”.

Steps for this task

1. Create an event listener object, which can be any type. For example, see the following interface code:
   ```
   interface SampleEventGroup
   {

   void finished(String message);
   }

   class myListener implements SampleEventGroup
   {

   public void finished(String message)

   {

   // This will be called when we 'finish'.

   }
   }
   ```
2. Register the event listener object with the event source

   For example, see the following code:
   ```
   InitialContext ic = ...;
   EventSource es = (EventSource)ic.lookup("java:comp/websphere/ApplicationNotificationService");
   myListener l = new myListener();
   es.addListener(l);
   ```

   This enables the myListener.finished() method to be called whenever the event is fired. The following code example shows how this event might be fired:
   ```
   InitialContext ic = ...;
   EventSource es = (EventSource)ic.lookup("java:comp/websphere/ApplicationNotificationService");
   myListener proxy = es.getEventTrigger(myListener.class);
   // fire the 'event' by calling the method
   // representing the event on the proxy
   proxy.finished("done");
   ```

Using the application notification service

During an application’s lifetime, individual J2EE components (servlets or enterprise beans) within a single EAR file might need to signal each other. There is an event source in the java:comp namespace that is bound into all components within an EAR file. The JNDI name for this event source is:

```
java:comp/websphere/ApplicationNotificationService
```

Components within the same application can fire asynchronous events and register event listeners using this application notification service. Startup beans can be used to register these event listeners at application startup or they can be registered dynamically at run time.

To have your enterprise bean or servlet use the application notification service, write code similar to what is shown in the following example:

```
InitialContext ic = new InitialContext();
EventSource appES = (EventSource)
  ic.lookup("java:comp/websphere/ApplicationNotificationService");
// now, the application can add a listener using the EventSource.addListener method.
// MyEventType is an interface.
MyEventType myListener = ...;
AppES.addListener(myListener);
```

```
// later another component can fire events as follows
InitialContext ic = new InitialContext();
EventSource appES = (EventSource)
ic.lookup("java:comp/websphere/ApplicationNotificationService");

// This highlights a constant string on the EventSource interface which
// specifies the 'java:comp/websphere/ApplicationNotificationService' string.
ic.lookup(appES.APPLICATION_NOTIFICATION_EVENT_SOURCE)
// now, the application can add a listener using the EventSource.addListener method.
MyEventType proxy = appES.getEventTrigger(MyEventType.class, false);
proxy.someEvent(someArguments);
```

Example: Event listener

The following code example demonstrates how to fire a listenerCountChanged event:

```
// imagine this snippet inside an EJB or servlet method.
// Make an inner class implementing the required event interfaces.
EventSourceEvents listener = new Object() implements EventSourceEvents.class
{
 void listenerCountChanged(EventSource es, int old, int newCount)
 {
  try
  {
     InitialContext ic = new InitialContext();
   // here, the asynch bean can access an environment variable of
   // the component which created it.
    int i = (Integer)ic.lookup("java:comp/env/countValue").intValue());
   if(newCount == i)
   {
    // do something interesting
   }
   // this should be called when the code below executes.
  }
  catch(NamingException e)
  {
  }
 }
 void listenerExceptionThrown( EventSource es, Object listener,
       String methodName, Throwable exception)
 {
 }
 void unexpectedException(EventSource es, Object runnable, Throwable exception)
 {
 }
}
// register it.
es.addListener(listener);

...

// now fire an event which the above listener should receive.
EventSourceEvents proxy = (EventSourceEvents)
   es.getEventTrigger(EventSourceEvents.class, false);

proxy.listenerCountChanged(es, 0, 1);

// now, fire another event, we could call any of the methods.
proxy.listenerCountChanged(es, 4, 5);
```

In this example, we get a proxy for the interface that we want to fire a method on. Then, call the method corresponding to the event on the proxy. This causes the same method, with the same parameters, to be called on any event listeners that implement the EventSourceEvents interface and that were previously registered with the EventSource ″es″. The same proxy can be used to send multiple events simultaneously.

The boolean parameter on the getEventTrigger() is named ″sameTransaction″. When the sameTransaction parameter is false, a new transaction is started for each event listener invoked and these event listeners can be called in parallel to the caller. However, the event() method always is blocked until all of the event listeners have been notified. If the sameTransaction parameter is true, the current transaction, if any, on the thread is used for all of the event listeners; that is, the event listeners share the transaction of the method that fired the event. For that reason, all event listeners must run serially in an undetermined order. That is, the order in which the listeners are called is undefined and the order in which listeners were registered should not be a guide for the order used at run time. The method on the proxy does not return until all of the event listeners have been called; that is, it is a synchronous operation.

The parameters are passed by reference and listeners should not interfere with these references unless the method that fired the event has purposefully designed such interaction. For example, event listeners can be used as collaborators and add data to a map, which was a parameter. Each event listener runs on its own transaction, independent of any transaction that is active on the thread. Extreme care must be taken when the sameTransaction parameter is false because the parameters can potentially be accessed by multiple threads.

Developing Asynchronous scopes

Asynchronous scopes are units of scoping that comprise a set of alarms, subsystem monitors, and child asynchronous scopes. Using asynchronous scopes can involve some or all of the following steps:

Steps for this task

1. Create asynchronous scopes

   Create a parent asynchronous scope object by calling the AsynchScopeManager.createAsynchScope() method using a unique name as the parameter.

   You can store properties in an asynchronous scope object. This provides J2EE applications with a way to store a non-serializable state that otherwise cannot be stored in a session bean.

   You also can create child asynchronous scopes, which is useful for scoping data beneath the parent.
2. Listen for alarm notifications
   a. Create a listener object by implementing the AlarmListener interface. For more information, see the AlarmListener interface in the Javadoc, which is located in the InfoCenter.
   b. Supply this object to the AlarmManager.create() method, as the target for the alarm.

      The create() method takes the following parameters:

      Target for the alarm
         The target on which the fired() method is called when the alarm is fired.

Context
The context object for the alarm. This is useful for supplying alarm-specific data to the listener and allows a single listener to be used for multiple alarms.

Interval
The number of milliseconds before the alarm fires.

After the specified interval, the alarm fires and the fired() method of the listener is called with the firing alarm as a parameter. The alarm object, itself, is returned. By calling methods on this object, you can cancel or reschedule the alarm.

3. Monitoring remote systems
   a. Implement a mechanism for detecting messages sent from the remote system. For example, publish-subscribe messaging.
   b. Create a subsystem manager object by calling the SubsystemMonitorManager.create() method with the following parameters:

      Name Each subsystem monitor must have a unique name.

      Heartbeat interval
      The expected interval, in milliseconds, between heartbeats.

      Missed heart beats until stale or suspect
      The number of heartbeats that can be missed before the subsystem is marked as stale.

      Missed heart beats until dead
      The number of heartbeats that can be missed before the system is marked as dead.
   c. Create an object that implements the SubsystemMonitorEvents interface. For more information, see the SubsystemMonitorEvents in the Javadoc, which is located in the InfoCenter
   d. Add an instance of this object to the subsystem monitor using the SubsystemMonitor.addListener() method.
   e. Whenever a heartbeat message arrives from the remote system, call the SubsystemMonitor's ping() method.

   The subsystem monitor configures alarms to track the heartbeat status of the remote system. Whenever the ping() method is called, the alarms are reset. If an alarm fires, the ping() method has not been called; that is, the application did not receive a heartbeat from the subsystem being monitored.

Usage scenario

Asynchronous scopes are useful in stateful server applications. An application can have a startup bean that creates an asynchronous scope on a named work manager. The application also might create subsystem monitors to monitor the health of any remote systems on which the application is dependent.

When a client attaches to the server, the application creates a child asynchronous scope that is owned by the application asynchronous scope for the client and named using the client ID. A subsystem monitor for monitoring the client itself might be created on the client asynchronous scope. If the client times out, a callback can clean up the client state on the server. Callbacks can be attached to the application subsystem monitors, on behalf of the client. When a remote system becomes unavailable, the client code in the server is notified and an event is sent to the client to warn it that a critical remote system has failed. For example, the failure might be a data feed in an electronic trading application.

Asynchronous scopes

An asynchronous scope (AsynchScope object) is a unit of scoping provided for use with asynchronous beans.

Asynchronous scopes are collections of alarms, subsystem monitors, and child asynchronous scopes that enable a relationship to be formed. Each asynchronous scope uses a single work manager.

Each AsynchScope object owns and controls the life cycle of the following objects:

Child asynchronous scopes
: Each AsynchScope object extends the AsynchScopeManager interface, which is a factory for AsynchScope objects. (For more information on the AsynchScopeManager interface, see the Javadoc, which is located in the InfoCenter.). Any asynchronous scope can therefore create named asynchronous scopes (children). Child asynchronous scopes can be useful for scoping data underneath the parent. All of the child asynchronous scopes must be uniquely named. These children are destroyed if the parent asynchronous scope is destroyed.

Alarms
: Each asynchronous scope has an associated alarm manager. All of the alarms created by the alarm manager are automatically cancelled if the associated asynchronous scope is destroyed.

Subsystem monitors
: Each asynchronous scope has a subsystem monitor manager, which manages a set of subsystem monitors associated with the asynchronous scope. When the asynchronous scope is destroyed, all of the associated subsystem monitors also are destroyed.

To summarize, asynchronous scopes can be organized into an acyclic tree. The life cycle of each asynchronous scope is directly coupled to that of its parent asynchronous scope. Each asynchronous scope is associated with a set of alarms and subsystem monitors, and an optional set of child asynchronous scopes. These objects are cancelled and destroyed when the asynchronous scope is destroyed.

Asynchronous scope state

Each asynchronous scope has an associated map, in which applications can store state in the form of name and value pairs.

Asynchronous scope events

Each asynchronous scope is also an event source. Applications can therefore register event listeners against the asynchronous scope. The event listeners can receive notification if, for example, the AsynchScope object is about to be destroyed.

Applications also can use this event source to fire events only to listeners of this asynchronous scope. For example, an AsynchScope object created for a client session might be used to fire asynchronous events to parties interested in that client.

Alarms

An alarm executes J2EE context-aware code at a given time interval. Alarm objects are fine-grained, non-persistent, transient, and can fire at millisecond intervals.

Alarms, themselves, are executed using a thread pool associated with the work manager that owns the associated asynchronous scope.

The AlarmManager.createAlarm() method takes an application-written object that implements the AlarmListener interface. (For more information on the AlarmListener interface, see the Javadoc, which is located in the InfoCenter.)The fired method is called when the alarm expires. The createAlarm() method returns a non-serializable handle, which can be used to cancel or reset the alarm. All of the pending alarms are cancelled when its associated AsynchScope object is destroyed.

The Java 2 SDK already has a timer mechanism, so why create a new one? The Java 2 SDK is a J2SE feature that knows nothing about the J2EE environment. Timers fired by the J2SE feature do not run on a managed thread and are therefore unusable inside an application server. These timers also lack a J2EE context (that is, a java:comp value) and are not authenticated when they fire. The asynchronous scope alarms are fully supported by IBM WebSphere Application Server Enterprise and have the same properties as any other asynchronous bean.

Alarm performance

The alarm subsystem is designed to handle a large number of alarms. However, do not have alarms undertake heavy processing when they are firing as this slows the processing of later alarms. If an alarm needs to process a heavy load, design a work object that is activated by a work manager. This procedure moves the heavy processing to a different thread and enables the alarm threads to process alarms unhampered. All of the alarms owned by asynchronous scopes that, in turn, are owned by a single work manager, share a common thread pool. The properties of this thread pool can be tuned at the work manager level using the administrative console.

Subsystem monitors

A subsystem monitor is an object that monitors the health of a remote system. It uses an event source to inform all registered listeners of the health of the system.

Advanced J2EE applications often rely on remote, non-managed, non-J2EE systems. These remote systems can periodically send clients a message to indicate that they are working. A subsystem monitor is essentially a set of alarms that track indicators messages or "heartbeats" from a remote system.

An application creates a subsystem monitor by calling the SubsystemMonitorManager.create() method with the following parameters:

Name Each subsystem monitor must be uniquely named.

Heart beat interval
The time period, in milliseconds, between arriving heartbeat messages.

Missed heart beats until stale or suspect
The number of heartbeats that can be missed before the subsystem is marked as stale. This designation indicates that the subsystem might be having problems.

Missed heart beats until dead

The number of heartbeats that can be missed before the system is considered to be down. The system then is marked as dead.

The subsystem monitor configures alarms to track the heartbeat status. Whenever the ping() method is called, the alarms are reset. If an alarm fires, the ping() method has not been called; that is, the application did not receive a heartbeat from the subsystem being monitored. When the number of **Missed heart beats until stale** has elapsed without a ping, a stale event is fired. Later, if the number of **Missed heart beats until dead** elapses without a ping, a dead event is fired. If a ping is received after a stale or dead notification, a fresh event is sent, which indicates that the subsystem is alive again.

Make **Missed heart beats until dead** greater or equal to the **Missed heart beats until stale**. If **Missed heart beats until stale** equals **Missed heart beats until dead**, a stale event is not published; only a dead event is published.

Applications that want to be informed of these events can register a listener that implements the SubsystemMonitorEvents interface. For more information on the SubsystemMonitorEvents interface, see the Javadoc, which is located in the InfoCenter..

Heart beat messages can be transmitted using a variety of mechanisms. The application must call the SubsystemMonitor's ping() method whenever a heartbeat message arrives from a remote system, but the method used to detect these messages is up to the application. For example, you might use a Java Message Service (JMS) publish or subscribe implementation or even a third-party Java messaging product that does not implement JMS.

Asynchronous scopes: Dynamic message bean scenario

J2EE now supports message-driven beans, but the beans are static. All of the message sources must be known in advance and bound at deployment time. This is not always viable, especially in fluid messaging environments such as those found in brokerages. Some environments have publish-subscribe topic spaces that are continually changing and clients need servers that can subscribe on demand to an arbitrary topic.

An asynchronous bean application can create a work object that performs a blocking receive on a JMS topic and then publishes the message as an event on an application-defined event source. Clients requiring a subscription to that message can add an event listener to the event source. The event source can inform the work object when there are no listeners. Then, the event source can shut down and make the JMS and thread resources available. The work object registers a listener with its own event source. When the count is one again, the work object knows that it is the only listener and its time to shut down the work object. The WebSphere Trader sample (see your installed Samples Gallery) uses this pattern to dynamically subscribe to JMS topics at run time to gather stock prices. For more information, see an overview of the samples.

How does the server catch clients that disconnect or crash? It creates a subsystem monitor to watch the client and adds an event listener to catch dead events. When a dead event occurs, the server application can clean up the client's server state. For example, the server application can remove the client's event listener from the dynamic message bean; thereby allowing the server to subscribe to a dynamic topic only when it is needed.

Chapter 8. Using object pools

An object pool enables an application to avoid creating new Java objects repeatedly. Most objects can be created once, used, and then reused at a later point. An object pool allows an object to be pooled while waiting for the point when it can be reused. These object pools are not meant to be used for pooling JDBC connections or JMS connections and sessions. WebSphere provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic JDK types.

To use an object pool, the WebSphere administrator must define an *object pool manager* using the administrative console. Multiple object pool managers can be created in a Websphere cell.

Note: The Object pool manager service is only supported from within the EJB Container or Web Container (EJBs or Servlets). Looking-up and using a configured object pool manager from a J2EE application client container is not supported.

Steps for this task

1. Start the administrative console.
2. Select **Resources > Object Pools**.
3. Define the name of the object pool manager. This name can be up to 30 ASCII characters long.
4. Assign the object pool manager a JNDI name.
5. Provide a description of this object pool manager.
6. Categorize the object pool manager.

Results

After completing this steps, applications can find the object pool manager by doing a JNDI lookup using the specified JNDI name.

Usage scenario

The following code illustrates how an *application* can find an Object pool manager object:

```
InitialContext ic = new InitialContext();
ObjectPoolManager opm = (ObjectPoolManager)ic.lookup("java:comp/env/pool");
```

Once the application has an ObjectPoolManager, it can cache an object pool for classes of the types it wants to use. The following is an example:

```
ObjectPool arrayListPool = nulll;
ObjectPool vectorPool = null;
try
{
 arrayListPool = opm.getPool(ArrayList.class);
 vectorPool = opm.getPool(Vector.class);
}
catch(InstantiationException e)
{
 // problem creating pool
}
```

```
catch(IllegalAccessException e)
{
 // problem creating pool
}
```

Once the application has the pools, it can use them as in the following example:

```
ArrayList list = null;
try
{
 list = (ArrayList)arrayListPool.getObject();
 list.clear(); // just in case
 for(int i = 0; i < 10; ++i)
 {
  list.add("" + I);
 }
 // do what ever we need with the ArrayList
}
finally
{
 if(list != null) arrayListPool.returnObject(list);
}
```

This is the basic pattern for using object pooling. If the application "forgets" to return the object, the only adverse effect is that the object cannot be reused.

Object pool managers

Object pool managers control the reuse of application objects and JDK objects such as Vectors and HashMaps.

Multiple object pool managers can be created in a WebSphere cell. Each object pool manager has a unique cell-wide JNDI name. Applications can find a specific object pool manager by doing a JNDI lookup using the specific JNDI name.

The Object pool manager and its associated objects implement the following interfaces:

```
public interface ObjectPoolManager
{
 ObjectPool getPool(Class aClass)
  throws InstantiationException, IllegalAccessException;
 ObjectPool createFastPool(Class aClass)
  throws InstantiationException, IllegalAccessException;

}

public interface ObjectPool
{
 Object getObject();
 void returnObject(Object o);

}
```

Each object pool manager can be used to pool any Java object with the following characteristics:

- The object must be a public class with a public default constructor.
- Each object class to be pooled must have its own object pool.
- An application gets an object pool for a specific object using either the ObjectPoolManager.getPool() or ObjectPoolManager.createFastPool() method. The difference between these methods is that the getPool() method returns a pool

that can be shared across multiple threads. The createFastPool() method returns a pool that can only be used by a single thread.

If, in a JVM, the getPool() method is called multiple times for a single class, the same pool is returned. A new pool is returned for each call when the createFastPool() method is called. Basically, the getPool() method returns a pool that is thread-synchronized.

The pool for use by multiple threads is slightly slower than a fast pool due to the need to handle thread synchronization. However, extreme care must be taken when using a fast pool. Consider the following interface:

```
public interface PoolableObject
{
 void init();
 void returned();
}
```

If the objects placed in the pool implement this interface and theObjectPool.getObject() is called, the object returned has the init() method called on it. When the ObjectPool.returnObject() method is called, the returned method is called on the object before it is returned to the object pool. This allows objects to be pre-initialized or cleaned up.

It is not always possible for an object to implement PoolableObject. For example, an application might want to pool ArrayList objects. The ArrayList would need to be cleared each time the application reuses it. The application might extend ArrayList and have that implement Poolable. For example, consider the following:

```
public class PooledArrayList extends ArrayList implements PoolableObject
{
 public PooledArrayList()
 {
 }

 public void init() {
 }

 public void returned()
 {
  clear();
 }
}
```

If the application uses this, in place of a true ArrayList, the ArrayList is cleared automatically when it is returned to the pool.

Note: Clearing an ArrayList simply marks it as empty and the array backing the ArrayList is not freed.

Therefore, as the application reuses the ArrayList, the backing array expands until it is big enough for all of the application requirements. Once this point is reached, it stops allocating and copying new backing arrays and achieves the best performance.

It might not be possible or desirable to use the previous procedure. An alternative is to implement a custom object pool and register this with the object pool manager as the pool to use for classes of that type. The class is registered by the WebSphere administrator when the object pool manager is defined in the cell. Take care that these classes are packaged in JAR files available on all of the nodes in the cell where they might be used.

Object pool manager collection

Use this page to manage object pool managers.

To view this administrative console page, click **Resources > Object Pools** .

Name

The name by which the object pool manager is known for administrative purposes.

Data type
String

Range 1 through 30 ASCII characters

JNDI Name

The JNDI name for the object pool manager.

Data type
String

Description

A description of the object pool manager.

Data type
String

Category

A category string used to classify or group this object pool manager.

Data type
String

Object pool manager settings

Use this page to modify object pool manager settings.

To view this administrative console page, click **Resources > Object Pools >** *objectpoolmanager_name*

Name

The name by which the object pool manager is known for administrative purposes.

Data type
String

Range 1 through 30 ASCII characters

JNDI Name

The JNDI name for the object pool manager.

Data type
String

Description

A description of the object pool manager.

Data type
String

Category

A category string used to classify or group this object pool manager.

Data type
String

Custom object pool collection

Use this page to manage object pools.

To view this administrative console page, click **Resources > Object Pools >** *objectpoolmanager_name* **> Object Pools**.

Pool Class Name

The fully-qualified class name of the objects that are stored in the object pool.

Data type
String

Pool Impl Class Name

The fully-qualified class name of the CustomObjectPool implementation class for this object pool.

Data type
String

Custom object pool settings

Use this page to modify custom object pool settings.

An object pool manages a pool of arbitrary objects.

To view this administrative console page, click **Resources > Object Pools >** *objectpoolmanager_name* **> Object Pools >** *objectpool_name*.

Configuration tab

Pool Class Name
The fully-qualified class name of the objects that are stored in the object pool.

Data type
String

Pool Impl Class Name
The fully-qualified class name of the CustomObjectPool implementation class for this object pool.

Data type
String

Object pool service settings

Use this page to enable or disable the object pool service, which manages object pool resources used by the server.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Object Pool Service** .

Startup

Specifies whether the server will attempt to start the object pool service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the object pool service automatically.

Cleared
The server does not try to start the object pool service service. If object pool resources are to be used on this server, the system administrator must start the object pool service manually or select this property then restart the server.

Object pools: Resources for learning

Use the following links to find relevant supplemental information about object pools. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Programming model and decisions

- Java theory and practice: Thread pools and work queues
  http://www-106.ibm.com/developerworks/library/j-jtp0730.html
- Java performance programming, Part 1: Smart object-management saves the day
  http://www-106.ibm.com/developerworks/library/jw-performance.html
- Build your own ObjectPool in Java to boost app speed
  http://www.javaworld.com/jw-06-1998/jw-06-object-pool.html
- Improve the robustness and performance of your ObjectPool
  http://www.javaworld.com/jw-08-1998/jw-08-object-pool.html
- Java Tip 78: Recycle broken objects in resource pools
  http://www.javaworld.com/javaworld/javatips/jw-javatip78.html

Chapter 9. Using startup beans

A startup bean is a stateful session bean that is loaded when an application starts. Startup beans enable J2EE applications to execute business logic automatically, whenever an application starts or stops normally.

Startup beans are especially useful when used in combination with asynchronous bean features. For example, a startup bean might create an alarm object that uses JMS to periodically publish heartbeat messages on a well-known topic. This enables clients or other server applications to determine whether the application is available.

Steps for this task

1. Use the home interface, com.ibm.websphere.startupservice.AppStartUpHome, to designate a bean as a startup bean
2. Use the remote interface, com.ibm.websphere.startupservice.AppStartUp, to define a start() and stop() method on the bean.

   The bean's start() method is called when the application starts. It implements any business logic that needs to run at application start time.

   The start() method returns a boolean. True indicates normal application startup and false indicates that the application start process should be aborted. The start() and stop() methods should not use a TX_MANDATORY attribute because there never is a transaction on the thread when the start() or stop() methods are invoked. Any other TX_* attribute can be used. If TX_MANDATORY is used, an exception is logged (need a transaction for mandatory) and the application does not start.

   The bean's stop() method is called when the application stops and implements any business logic that needs to run at this time. Any exception thrown by a stop() method is ignored, but logged to trace.

   The start() and stop() methods on the remote interface use **Run-As** mode. **Run-As** mode specifies the credential information to be used by the security service to determine the permissions that a principal has on various resources. If security is on, the **Run-As** mode needs to be defined on all of the methods called. The identity of the bean without this setting is undefined. For more information about the **Run-As** mode, see the topic Method extension assembly settings.

   There are no restrictions on what code the start() and stop() methods can run, since the full Enterprise Application Server programming model is available to these methods.
3. Use an *optional* environment property called `wasStartupPriority`, which is an integer, to specify the start order of multiple startup beans in the same JAR file.

   If the environment property is found and is the wrong type, application startup is aborted. If no priority value is specified, a default priority of 0 is used. It is recommended that you specify the priority property. Beans that have specified a priority are sorted using this property. Beans with numerically lower priorities are executed first. Beans that have the same priority are executed in an undefined order. All priorities must be positive integers. The priority is used to order beans within an EJB JAR file. The order in which this process is applied to different EJB JAR files in a single EAR file is undefined. Beans are stopped in the opposite order to their start priority.

Startup beans must specify a timeout value of 0. Failure to do so causes the bean to be passivated and results in errors when attempting to call the stop() method when the application is stopped.

Chapter 10. Using the scheduler service

The scheduler service enables tasks to be executed at a requested time. The following tasks can be scheduled:

- Invoke a session bean method.
- Send a JMS message on a Queue or Topic.

The scheduler service performs the task, repeating as necessary, according to the task's metadata.

Steps for this task

1. "Developing and scheduling tasks" on page 249.

   Includes instructions for developing various types of tasks, receiving notifications from a scheduler, submitting tasks to a scheduler, and managing tasks.

   Note: Creating and manipulating scheduled tasks through the Scheduler interface is only supported from within the EJB Container or Web Container (enterprise beans or servlets). Looking-up and using a configured scheduler from a J2EE application client container is not supported.
2. "Managing the scheduler service".

   Includes instructions for creating and configuring a database for scheduler, configuring a scheduler instance, and enabling or disabling the scheduler service (the service is enabled by default).

Managing the scheduler service

Schedulers are configured using the administrative console. Schedulers are available to all servers on which the scheduler service is enabled.

Steps for this task

1. "Creating the database for scheduler"
2. "Configuring a scheduler" on page 244
3. "Enabling the scheduler service" on page 249

   The scheduler service is enabled by default.

Creating the database for scheduler

Before you begin

Your database system must be installed and available.

It is important to realize that the scheduler uses this database for storing tasks and then executing them. The performance of the scheduler is ultimately limited by the performance of the database. If you need more tasks per second, you can run the scheduler daemons on larger systems or you can use clusters for the session beans used by the tasks. Eventually, however, the task database becomes saturated and you then need a larger or better-tuned database system.

Multiple applications can share a scheduler database. This can lower the cost of administering the scheduler database.

Scheduler requires a database, JDBC provider, and data source.

Steps for this task

1. Create the database according to the description for your database system:
   - "Creating a Cloudscape database for scheduler"
   - "Creating a DB2 database for scheduler" on page 241
   - "Creating an Informix database for scheduler" on page 242
   - "Creating a Microsoft SQL Server database for scheduler" on page 242
   - "Creating an Oracle database for scheduler" on page 243
   - "Creating a Sybase database for scheduler" on page 244
2. If the database is not on the same machine as your IBM WebSphere Application Server, verify that you can access the database from your application server machine.
3. Configure your JDBC provider and data source.

   For details, see "Creating and configuring a JDBC provider and data source" in the InfoCenter..

Creating a Cloudscape database for scheduler

Cloudscape is a database system implemented in Java. It is delivered with IBM WebSphere Application Server as three JAR files. The Cloudscape license that comes with WebSphere is only for development and test, not for production purposes.

Steps for this task

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. If you want to use an existing database, skip to [ ].

   Note: Make sure that the database supports Unicode (UTF-8) . Otherwise, it cannot store all characters that can be handled in Java, and you could run into codepage conversion problems when a client uses an incompatible codepage.
4. Use the Cloudview utility supplied with Cloudscape to create a database named `scheddb`.

   Note: Cloudscape allows only one local connection. If IBM WebSphere Application Server is running and accessing a Cloudscape database, attempts to open a second connection to the database from the command line are rejected.
5. Create the schema.
   a. Using a text editor, edit the script `%WAS_HOME%\Scheduler\createSchemaCloudscape.ddl` according to the instruction at the top of the file.
   b. Enter one of the following commands:

      On Windows:

      ```
      %WAS_HOME%\java\jre\bin\java -Djava.ext.dirs=%WAS_HOME%/lib
      -Dij.protocol=jdbc:db2j: -Dij.database=scheddb com.ibm.db2j.tools.ij
      %WAS_HOME%\Scheduler\createSchemaCloudscape.ddl
      ```

On UNIX:

```
%WAS_HOME%/java/jre/bin/java -Djava.ext.dirs=%WAS_HOME%/lib
-Dij.protocol=jdbc:db2j: -Dij.database=scheddb com.ibm.db2j.tools.ij
%WAS_HOME%/Scheduler/createSchemaCloudscape.ddl
```

Note: The previous two commands were split into three lines due to the maximum width of the page. However, type the appropriate command on one continuous line.

Results

The Cloudscape database for Scheduler exists.

Creating a DB2 database for scheduler

Steps for this task

1. Open a DB2 command-line window.
2. Make sure that you have administrator rights for the database system.
3. If you want to use an existing database, skip to [ ].

   Note: Make sure that the database supports Unicode (UTF-8) . Otherwise, it cannot store all characters that can be handled in Java, and you could run into codepage conversion problems when a client uses an incompatible codepage.

   To avoid deadlocks, be sure that the DB2 isolation level is set to "read stability". If necessary, enter the command `db2set DB2_RR_TO_RS=YES` then restart the DB2 instance to activate the change.
4. In the DB2 command line processor, enter this command to create the database:
   ```
   db2 CREATE DATABASE scheddb USING CODESET UTF-8 TERRITORY en-us
   ```
   A DB2 database named `scheddb` has been created.
5. Create the tablespace and schema.
   a. **(Optional)** Analyze the results of your experiences during development and system testing.

      The size of your database depends on many factors. If possible, distribute tablespace containers across different logical disks, and implement an appropriate security policy. Consider the performance implications of your choices for bufferpools and log file settings.

   b. Using a text editor, edit the following scripts according to the instruction at the top of each file: `%WAS_HOME%\Scheduler\createTablespaceDB2.ddl`, `%WAS_HOME%\Scheduler\createSchemaDB2.ddl`, `%WAS_HOME%\Scheduler\dropSchemaDB2.ddl`, and `%WAS_HOME%\Scheduler\dropTablespaceDB2.ddl`.

   c. Make sure that you are attached to the correct instance.

      Check the environment variable `DB2INSTANCE`.

   d. To connect to a database named `scheddb`, enter the command:
      ```
      db2 connect to scheddb
      ```
   e. To create the tablespace, enter the command:
      ```
      db2 -tf createTablespaceDB2.ddl
      ```
      Make sure that the script's output contains no errors. If there were any errors, you can drop the tablespace using the script `dropTablespaceDB2.ddl`.

f. To create the schema (tables and indices), in the DB2 command line processor, enter the command:

```
db2 -tf createSchemaDB2.ddl
```

Make sure that the script's output contains no errors. If there were any errors, you can use `dropSchemaDB2.ddl` to drop the schema.

Results

The DB2 database for scheduler exists.

Creating an Informix database for scheduler

Steps for this task

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. If you want to use an existing database, skip to [ ].

   Note: Make sure that the database supports Unicode (UTF-8) . Otherwise, it cannot store all characters that can be handled in Java, and you could run into codepage conversion problems when a client uses an incompatible codepage.
4. If you want to create a new database named `scheddb`, enter the command:
   ```
   dbaccess CREATE DATABASE scheddb
   ```
5. Create the schema.
   a. Using a text editor, edit the script `%WAS_HOME%\Scheduler\createSchemaInformix.sql` according to the instruction at the top of the file.
   b. Enter the command:
   ```
   dbaccess scheddb createSchemaInformix.sql
   ```

Results

The Informix database for scheduler exists.

Creating a Microsoft SQL Server database for scheduler

Steps for this task

1. Open a command-line window.
2. Change to the directory where the configuration scripts for scheduler are located. This is the `Scheduler` subdirectory of the IBM WebSphere Application Server installation directory.

   On Windows, enter:
   ```
   cd %WAS_HOME%\Scheduler
   ```
   On UNIX, enter:
   ```
   cd $WAS_HOME/Scheduler
   ```
3. Using a text editor, edit the schema creation script `createSchemaMSSQL.sql` according to the instruction at the top of the file.
4. If you want to use an existing database, skip to [ ].

Note: Make sure that the database supports Unicode (UTF-8) . Otherwise, it cannot store all characters that can be handled in Java, and you could run into codepage conversion problems when a client uses an incompatible codepage.

5. If you want to create a new database named `scheddb`:
   a. Make sure that you are using a user ID that has administrator rights for the database system.
   b. In the **Enterprise Manager**, expand a server group, then expand a server.
   c. Right-click **Databases**, then click **New Database**.
   d. Type the name `scheddb`.
   e. Modify any default values, as desired, then save.

   An Microsoft SQL Server database named `scheddb` has been created.
6. To create the schema:
   a. Make sure that you have administrator rights for the database system.

      The user ID you use to create the schema must be the one that you tell WebSphere to use when accessing the database.
   b. Run the script to create the schema (tables and views):

      ```
      isql -S <serverName> -U<userid> -P<password> -D<databaseName> -i createSchemaMSSQL.sql
      ```

Results

The Microsoft SQL Server database for scheduler exists.

Creating an Oracle database for scheduler

Steps for this task

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. If you want to use an existing database, skip to [ ].

   Note: Make sure that the database supports Unicode (UTF-8) . Otherwise, it cannot store all characters that can be handled in Java, and you could run into codepage conversion problems when a client uses an incompatible codepage.
4. Use the Database Configuration Assistant to create a database named `scheddb`.

   Make sure that you select the JServer option for the database. It is recommended to use a Unicode codepage when creating the database. The text data you pass to the APIs must be compatible with the selected codepage.
5. Create the tablespace and schema.
   a. Using a text editor, edit the scripts `%WAS_HOME%\Scheduler\createTablespaceOracle.ddl` and `%WAS_HOME%\Scheduler\createSchemaOracle.ddl` according to the instruction at the top of the files.
   b. If you do not want the schema to be created in the default instance, set the environment variable `ORACLE_SID`
   c. To create the tablespace, run the script `createTablespaceOracle.ddl`.

      For test purposes you can use the same location for all tablespaces and pass the path as a command line argument to the script, for example, on Windows, user ID `scheduser`, password `schedpwd`, database name `scheddb`, and tablespace path `d:\mydb\ts`, enter the command:

      ```
      sqlplus scheduser/schedpwd@scheddb @createTablespaceOracle.ddl d:\mydb\ts
      ```

If you get any errors creating the tablespace, you can use `dropTablespaceOracle.ddl` to drop the tablespace.

d. To create the schema, run the script `createSchemaOracle.ddl`.

   For example, on Windows, enter:

```
sqlplus scheduser/schedpwd@scheddb @createSchemaOracle.ddl
```

   If you get any errors creating the schema (tables and views), you can use `dropSchemaOracle.ddl` to drop the schema.

Results

The Oracle database for scheduler exists.

Creating a Sybase database for scheduler

Steps for this task

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. Make sure that you have the DTM option for Sybase ASE installed.
4. If you want to use an existing database, skip to [ ].

   Note: Make sure that the database supports Unicode (UTF-8) . Otherwise, it cannot store all characters that can be handled in Java, and you could run into codepage conversion problems when a client uses an incompatible codepage.
5. Use the Sybase isql utility to create a database named `scheddb`. See your Sybase product documentation for details.
6. Create the schema:
   a. Using a text editor, edit the script `%WAS_HOME%\Scheduler\createSchemaSybase12.ddl` according to the instruction at the top of the file.
   b. Enter the command:

```
isql -S <serverName> -U<userid> -P<password> -D scheddb -i createSchemaSybase12.ddl
```

Results

The Sybase database for scheduler exists.

Configuring a scheduler

Before your application can make use of the scheduler service, you need to configure a scheduler instance using the administrative console. Conceptually, a scheduler is similar to a datasource: you specify various configuration attributes, including a JNDI name where the instance will be bound. Once defined, an application using the scheduler API can look up the scheduler object and call various methods to manage tasks.

Steps for this task

1. Start the administrative console.
2. Select **Resources > Scheduler Configurations**.
3. Click **New**.
4. Specify configuration settings.

   Fields marked with an asterisk (*) are required. The settings are described in detail in the topic (configuration settings).

Scheduler configuration collection

Use this page to manage scheduler configurations.

To view this administrative console page, click **Resources > Scheduler Configurations** .

Name

The name by which this scheduler is known for administrative purposes.

Data type
String

JNDI Name

The JNDI name for the scheduler.

The JNDI name specifies where this scheduler instance is bound in the namespace. Clients can look this name up directly, although the use of resource references is recommended.

Data type
String

Description

A description of this scheduler for administrative purposes.

Data type
String

Category

A string that can be used to classify or group this scheduler.

Data type
String

Datasource JNDI Name

Datasource where persistent tasks will be stored.

Any datasource available in the name space can be used with a scheduler. Multiple schedulers can share a single datasource while using different tables by specifying a table prefix.

Data type
String

Datasource Alias

Alias to a user name and password used to access the datasource.

Data type
String

Table Prefix

String prepended to the table name TASK.

Multiple independent schedulers can share the same database if each instance specifies a different prefix string.

Data type
String

Poll Interval

The interval at which the scheduler daemon polls the database. Each scheduled, repeating task's minimum repeat interval will be equal to this value regardless of what is specified on the task.

Each poll operation can be expensive. If the interval is extremely small and there are many scheduled tasks, polling can consume a large portion of system resources.

Data type
Integer

Units Seconds

Default
30

Range Any positive long integer

Work Manager

Specifies the work manager used by this scheduler.

The Work Manager is a server object that serves as a logical thread pool for the scheduler. Each repeating task that is created using this scheduler will use the ″Number Of Alarm Threads″ specified in the Work Manager which will affect the number tasks that can run concurrently. Use the Work Manager's ″Service Names″ property to limit the amount of context information that is propagated to the task when it executes.

When a task fires, the task is run in the Work Manager associated with the scheduler instance. Configuring a scheduler with a specific Work Manager enables you to control how many tasks are actively running at a given time.

Scheduler configuration settings

Use this page to modify scheduler settings.

To view this administrative console page, click **Resources > Scheduler Configurations >** *scheduler_name*.

Configuration tab

Name The name by which this scheduler is known for administrative purposes.

Data type
String

JNDI Name
The JNDI name for the scheduler.

The JNDI name specifies where this scheduler instance is bound in the namespace. Clients can look this name up directly, although the use of resource references is recommended.

Data type
String

Description
A description of this scheduler for administrative purposes.

Data type
String

Category
A string that can be used to classify or group this scheduler.

Data type
String

Datasource JNDI Name
Datasource where persistent tasks will be stored.

Any datasource available in the name space can be used with a scheduler. Multiple schedulers can share a single datasource while using different tables by specifying a table prefix.

Data type
String

Datasource Alias
Alias to a user name and password used to access the datasource.

Data type
String

Table Prefix
String prepended to the table name TASK.

Multiple independent schedulers can share the same database if each instance specifies a different prefix string.

Data type
String

Poll Interval
The interval at which the scheduler daemon polls the database. Each scheduled, repeating task's minimum repeat interval will be equal to this value regardless of what is specified on the task

Each poll operation can be expensive. If the interval is extremely small and there are many scheduled tasks, polling can consume a large portion of system resources.

Data type
Integer

Units Seconds

Default
30

Range Any positive long integer

Work Manager
Specifies the work manager used by this scheduler.

The Work Manager is a server object that serves as a logical thread pool for the scheduler. Each repeating task that is created using this scheduler will use the ″Number Of Alarm Threads″ specified in the Work Manager which will affect the number tasks that can run concurrently. Use the Work

Manager's ″Service Names″ property to limit the amount of context information that is propagated to the task when it executes.

When a task fires, the task is run in the Work Manager associated with the scheduler instance. Configuring a scheduler with a specific Work Manager enables you to control how many tasks are actively running at a given time.

Creating a scheduler resource reference

When a scheduler has been defined in the server configuration, the object instance is bound into the global name space under the configured JNDI name. A resource reference can be used to avoid hardcoding this JNDI name into your application.

You can alternatively create a scheduler resource reference by editing the XML directly. A scheduler resource reference is a J2EE compliant resource that uses the class com.ibm.websphere.scheduler.Scheduler as the object type. For information regarding the XML file format, see the J2EE Specification.

Steps for this task

1. Start the Application Assembly Tool.
2. Select your application.
3. In the left-hand panel, right-click on **Resource References** and select **New**.
4. On the **General** tab, complete the following fields:

   Name Specify the name suffix. For example, if the scheduler name is *MyScheduler*, the reference JNDI name is **java:comp/env/*MyScheduler***

   Type From the drop-down list select **com.ibm.websphere.scheduler.Scheduler**
5. (Optional) At this time you can also specify the global JNDI name to which this resource reference is bound by entering the JNDI name on the **Bindings** tab.

Scheduler daemon

A scheduler daemon is a background thread that searches for events in the persistent store.

A scheduler daemon is started for each scheduler defined on each server. If ″Scheduler 1″ is configured on server1, then there will only be one scheduler daemon running on server1 unless it is cloned. If ″Scheduler 1″ is defined at the node scope level, then the scheduler will run on each server within that node.

The poll interval determines the frequency at which the persistent store is queried. By default, this value is set to 30 seconds. When a task is found that is scheduled to fire within the current poll interval, an alarm is set. The task then runs as close to this time as possible using an alarm thread from the scheduler's associated work manager. Thus, the number of alarm threads configured on the work manager determines how many concurrent tasks are executed. No tasks are lost. If we reach this limit, then new tasks are simply queued to be executed when an alarm thread becomes available. The actual firing time is dictated by server load and availability of free threads in the alarm thread pool of the associated work manager.

Scheduler daemons in a cluster

When multiple scheduler daemons are configured to the same table (as is the case in a clustered environment), any of the daemons can find a task and set the timer in its Java Virtual Machine (JVM). The task is executed in the virtual machine where the timer first fires.

Enabling the scheduler service

Before you begin

Before an application can make use of the scheduler service, you need to "Configuring a scheduler" on page 244.

The scheduler service manages all schedulers used by a given server. The scheduler service can be enabled and disabled on a server-by-server basis using the administrative console. The service is enabled by default. If you disable the service on a server, all schedulers configured on that server are no longer available. All lookups fail and all scheduler daemons are inactive.

Steps for this task

1. Start the administrative console.
2. Select **Servers >** *server_name* **> Scheduler Service**.
3. Select or clear the **Startup** checkbox in order to enable or disable the service.
4. Click **Save** on the menu bar to save your configuration.

Results

The change takes effect on the next server restart.

Scheduler service settings

Use this page to enable or disable the scheduler service, which manages scheduler resources used by the server.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Scheduler Service** .

Startup: Specifies whether the server will attempt to start the scheduler service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the scheduler service automatically.

Cleared
The server does not try to start the scheduler service. If scheduler resources are to be used on this server, the system administrator must start the scheduler service manually or select this property then restart the server.

Developing and scheduling tasks

Steps for this task

1. Developing a task.

   The scheduler API supports different implementations of the TaskInfo interface, each of which can be used to schedule a particular type of work. Refer to one of the following topics for details:

   - "Developing a task that calls a session bean" on page 250.
   - "Developing a task that sends a JMS message" on page 251. This task object can send a JMS message to either a queue or a topic.

Note: Creating and manipulating scheduled tasks through the Scheduler interface is only supported from within the EJB Container or Web Container (enterprise beans or servlets). Looking-up and using a configured scheduler from a J2EE application client container is not supported.

2. "Receiving scheduler notifications" on page 252.

   A notification sink is set on a task in order to receive the notification events that are generated by a scheduler when it performs an operation on the task.
3. "Submitting a task to a scheduler" on page 253.

   After a TaskInfo object has been created, it can be submitted to the scheduler for task creation by calling the Scheduler.create() method.
4. "Managing tasks with a scheduler" on page 253.

Developing a task that calls a session bean

The scheduler API supports different implementations of the TaskInfo interface, each of which can be used to schedule a particular type of work. This topic describes how to call a method on a task handler session bean using the BeanTaskInfo implementation.

Steps for this task

1. Create a stateless session bean that implements the process() method in the com.ibm.websphere.scheduler.TaskHandler remote interface. The process() method is called when the task fires.

   The Home and Remote interfaces must be set as follows in the bean's deployment descriptor:
   - com.ibm.websphere.scheduler.TaskHandlerHome
   - com.ibm.websphere.scheduler.TaskHandler
2. Create an instance of the BeanTaskInfo class.

   To create a BeanTaskInfo instance, call the constructor on the class. For example:
   ```
   BeanTaskInfo taskInfo = new BeanTaskInfo();
   ```
   Several other constructors are available that can simplify this programming model. For more information, refer to the Javadoc for the BeanTaskInfo class, which is located in the InfoCenter.

   Note: Creating a BeanTaskInfo object does not add the task to the persistent store. Rather, it creates a placeholder for the necessary data. The task is not added to the persistent store until the create() method is called on a Scheduler instance, as described in the topic "Submitting a task to a scheduler" on page 253.
3. Set parameters on the BeanTaskInfo object. These parameters define which task is to run, which session bean is called, and so on.

   The TaskInfo interface contains various set() methods that you can use to control execution of the task, including when the task will fire and what work the task will do when it fires. For example:
   ```
   //create a date object which represents 30 seconds from now
   java.util.Date startDate = new java.util.Date(System.currentTimeMillis()+30000);

   //find the session bean to be called when the task executes
   Object o = new InitialContext().lookup("java:comp/env/ejb/MyTaskHandlerHome");
   TaskHandlerHome home = (TaskHandlerHome)javax.rmi.PortableRemoteObject.narrow
   (o,TaskHandlerHome.class);
   ```

```
//now set the start time and task handler to be called in the task info
taskInfo.setTaskHandler(home);
taskInfo.setStartTime(startDate);
```

Note: The sixth line of the preceeding example wrapped onto a second line due to the width of the page.

The TaskInfo.html interface specifies additional control points, as documented in Javadoc.The Javadoc is located in the InfoCenter.

Results

A TaskInfo object has been created that contains all of the relevant data for a task.

What to do next

Submit the task to a scheduler instance for creation, as described in the topic "Submitting a task to a scheduler" on page 253.

Developing a task that sends a JMS message

The scheduler API supports different implementations of the TaskInfo interface, each of which can be used to schedule a particular type of work. This topic describes how to use the MessageTaskInfo implementation, which sends a JMS message to either a queue or a topic.

Steps for this task

1. Create an instance of the MessageTaskInfo class.

   To create a MessageTaskInfo instance, call the constructor on the class. For example:

   ```
   MessageTaskInfo taskInfo = new MessageTaskInfo();
   ```

   Several other constructors are available that can simplify this programming model. For more information, refer to the MessageTaskInfo.html class in the Javadoc.The Javadoc is located in the InfoCenter.

   Note: Creating a MessageTaskInfo object does not add the task to the persistent store. Rather, it creates a placeholder for the necessary data. The task is not added to the persistent store until the create() method is called on a Scheduler instance, as described in the topic "Submitting a task to a scheduler" on page 253.
2. Set parameters on the MessageTaskInfo object.

   The TaskInfo interface contains various set() methods that can be used to control execution of the task, including when the task will fire and what work the task will do when it fires. For example:

   ```
   //create a date object which represents 30 seconds from now
   java.util.Date startDate = new java.util.Date(System.currentTimeMillis()+30000);


   //now set the start time and the JNDI names for the queue connection factory and the queue
   taskInfo.setConnectionFactoryJndiName("jms/MyQueueConnectionFactory");
   taskInfo.setDestination("jms/MyQueue");
   taskInfo.setStartTime(startDate);
   ```

   The TaskInfo interface specifies additional control points, as documented in Javadoc. The Javadoc is located in the InfoCenter.

Results

A TaskInfo object has been created that contains all of the relevant data for a task.

What to do next

Submit the task to a scheduler instance for creation, as described in the topic "Submitting a task to a scheduler" on page 253.

Receiving scheduler notifications

Various notification events are generated by a scheduler when it performs an operation on a task. These events include:

Scheduled
A task has been scheduled.

Purged
A task has been permanently deleted from the persistent store.

Suspended
A task was suspended.

Resumed
A task was resumed.

Complete
A task has run completely. If it was a repeating task, all repeats have been performed.

Cancelled
A task has been cancelled. It will not run again.

Fired A task fired successfully.

Fire Failed
A task could not fire successfully.

To receive notification events, call the setNotificationSink() method on the TaskInfo interface before creating the event. The setNotificationSink() method enables you to specify the session bean that is to act as the callback, and a mask that restricts which events are generated.

Steps for this task

1. Create a notification sink session bean.

   Create a stateless session bean that implements the handleEvent() method in the com.ibm.websphere.scheduler.NotificationSink remote interface. The handleEvent() method is called when the notification is fired. The Home and Remote interfaces can be set as follows in the bean's deployment descriptor:

   ```
   com.ibm.websphere.scheduler.NotificationSinkHome
   com.ibm.websphere.scheduler.NotificationSink
   ```

   The notification sink bean must exist in the same application (EAR file) that is used to create the task.

   The NotificationSink interface defines the following method:

   ```
   public void handleEvent(TaskNotificationInfo task) throws java.rmi.RemoteException;
   ```

   The transactional context used by the session bean is defined by the assembler.

2. Specify the notification sink session bean to be used as the callback.

   The following code illustrates how to set this option:

```
TaskInfo taskInfo = ...
Object o = new InitialContext().lookup("java:comp/env/ejb/NotificationSink");
NotificationSinkHome home = (NotificationSinkHome )
javax.rmi.PortableRemoteObject.narrow(o,NotificationSinkHome.class);
taskInfo.setNotificationSink(home,TaskNotificationInfo.ALL_EVENTS);
```

Note: The third line of the preceeding code example beginning with ″NotificationsSinkHome″ was split into two lines due to the width of the page.

3. Specify the event mask.

   The event mask is specified as an integer mask. You can either use an individual mask such as TaskNotificationInfo.CREATED to receive specific events, TaskNotificationInfo.ALL_EVENTS to receive all events or a combination of specific events.. For example:

   ```
   int eventMask = TaskNotificationInfo.CREATED+TaskNotificationInfo.PURGED;
   ```

Submitting a task to a scheduler

Before you begin

This task assumes that you have already “Configuring a scheduler” on page 244 using the administrative console.

Once you have developed a TaskInfo object that contains all relevant data for a task, submit the task to a scheduler instance for creation. For example:

```
//lookup the scheduler instance to be used
Scheduler scheduler = (Scheduler)new InitialContext.lookup("java:comp/env/Scheduler");

TaskStatus status = scheduler.create(taskInfo);
```

When you configure a scheduler, it is bound to a global JNDI name. Although the desired scheduler instance can be found by performing a lookup on that JNDI name, it is better to create a resource reference, which allows for more flexibility in configuring the scheduler.

Note: The scheduler interface is a local interface. It can only be used by server-side code; that is, J2EE applications.

Once the call to the create() method is executed, the task exists in the persistent store and is run at the time specified in the TaskInfo object. This call is transactional. If a transactional context is present on the thread when the create() method rolls back or is aborted, the task does not run.

The status object, which has been returned by the call to the create() method, contains information about the state of the task, as well as the task ID. The task ID is the unique identifier for this task, and is required if the task is to be suspended, resumed, cancelled, and so on, at a later time.

Note: The status object is only a snapshot of the current state of the task. Use the Scheduler.getStatus() method to receive the current state when needed.

Managing tasks with a scheduler

When a task is created by calling the create() method on a scheduler instance, a TaskStatus object is returned to the caller. The status object contains the task ID, which is a unique identifier. The scheduler API defines several additional methods that pertain to the management of tasks, each of which accepts the task ID as a parameter. The following task management methods are defined:

suspend()
Suspends a task. The task does not run until it has been resumed.

resume()
Resumes a previously suspended task.

cancel()
Cancels a task. The task is not run.

purge()
Permanently deletes the task from the persistent store.

getStatus()
Returns the current status of the task.

For example, the following code creates and cancels a task:

```
//Create the task.
TaskInfo taskInfo = ...
TaskStatus status = scheduler.create(taskInfo);

//Get the task ID
String taskId = status.getTaskId();

//Cancel the task. Specify the purgeAlso flag so that the task does not remain
in the persistent store
scheduler.cancel(taskId,true);
```

Transactionality. All methods of the scheduler API are transactional. If a global transactional context is present, it is used to perform the operation. If an unexpected exception is thrown, the transaction is marked to roll back. If an expected or declared exception is thrown, the transaction remains intact and the caller must choose to roll back or to commit the transaction. If the transaction is rolled back at some point, all scheduler operations performed within the transaction ware also rolled-back.

If a local transactional context is present, it is suspended and a new global transactional context begins. Likewise, if no transactional context is active, a global transactional context begins. In both cases, if an unexpected exception is thrown, the transaction rolls back. If a declared exception is thrown, the transaction is committed.

If another thread is concurrently modifying the task in question, a TaskPending exception is thrown. This is because schedulers lock the database optimistically. The calling application can then retry the operation.

All methods defined by the scheduler API are described in Javadoc. The Javadoc is located in the InfoCenter.

Transactions and the scheduler service

Transactions and the scheduler daemon

Scheduled BeanTaskInfo and MessageTaskInfo objects are guaranteed to execute only once. This is accomplished by grouping all of the work done in the task as a single unit of work. When each task fires, the following events occur in a single global transactional context:

1. The context of the application that created the task is applied to the thread.
2. A global transactional context is started.
3. The next fire time and start-by time are calculated using the UserCalendar bean or the DefaultUserCalendar.

4. The task database task record is updated in the database with the state of the next task or deleted if the task is complete and the task's auto-purge setting is true.
5. The BeanTaskInfo or MessageTaskInfo object is executed.
6. If the task fails and the NotificationSink bean is set, a FIRE_FAILED notification is fired on a separate transaction.
7. If the task's NotificationSink bean is set, then the various notifications are fired as required.
8. The global transaction is committed.

Because all of a task's events are executed in a single global transactional context, you need to consider the following points in order to avoid transaction-related errors:

- Each resource participating in the task's transaction must be 2-phase XA capable.

  This includes the JDBC datasource configured for the scheduler, any JMS services used by the MessageTaskInfo objects, and any resources used within any of the UserCalendar, TaskHandler, or NotificationSink beans that have a transaction setting of ″Requires″.
- One resource can be single-phase, if last participant support is enabled for the application that created the transaction.

  Enable last participant support using the Application Assembly Tool. On the **WAS Enterprise** tab for your enterprise application, select the **Accept heuristic hazard** checkbox.

All unexpected exceptions are logged to the activity log and all events participating in the task's global transaction are rolled back. This includes changes to the task's database record, which force the task to be executed again when the scheduler daemon polls the database during the next poll cycle. The UserCalendar, TaskHandler, and NotificationSink beans can choose not to participate in the global transaction by setting the bean's transaction setting to ″Requires new″.

Transactions and the scheduler interface

All Scheduler interface methods participate in a single global transactional context. If a global transactional context is already present on the thread when the create(), suspend(), resume(), cancel(), and purge() methods are executed, the existing global transaction is used. Otherwise, a new global transaction begins.

If the method participates in the caller's global transaction and an unexpected error occurs, the transaction is marked to roll back. If the exception is a declared exception, then the exception is rethrown to the caller, and the transaction is left alone for the caller to commit or roll back.

If the method starts its own global transaction and any exception occurs, the transaction is rolled-back, and the exception is rethrown to the caller.

Scheduler interface

A scheduler object exists in the JNDI namespace for each scheduler configuration. A reference to a scheduler can be obtained by performing a lookup on the JNDI name; however, the lookup is valid only from the server process where the scheduler instance exists. Once a reference has been obtained, tasks can be created, suspended, cancelled, and so on, if the caller has access to the scheduler instance.

For details, see the Interface Scheduler in the Javadoc. The Javadoc is located in the InfoCenter.

Task creation
The task is created in the persistent store using the caller's global transactional context if present. See the topic "Transactions and the scheduler service" on page 254 for more details. Since this is a transactional operation, the task cannot be run or modified from another thread until the current transaction commits.

Task modification
Tasks that have been created can be modified with the suspend(), resume(), cancel(), and purge() methods. These methods take a Task Identifier string as a parameter, which is generated by the create() method and can be found in the TaskStatus object. If a task is currently running or being modified by another thread, an operation that attempts to modify the state of the task does not block on the attempt, but a TaskPending exception is thrown. The operation can be reattempted at another time. Tasks can only be modified by the same application (EAR file) that was used to create the task.

Task execution
Tasks are executed in the thread pool specified by the configuration's work manager, under the security ID of the task creator. If multiple schedulers are configured to share the same database table, the tasks found in the table can be executed on any of the schedulers, whether or not they are in the same server, node, or cell.

Task lookup
Tasks can be located using the Name property that was assigned at creation time. This is useful when you need to modify a group of tasks and tracking individual task ID's is not convenient.

TaskInfo interface

TaskInfo objects contain the information that can be used to create a task. Several implementations of this class exist, one for each type of task that can be run. Available TaskInfo implementations include:

BeanTaskInfo
Calls a stateless session bean.

MessageTaskInfo
Sends a JMS message to a queue or publishes a message to a topic.

For details, see the Interface TaskInfo in the Javadoc. The Javadoc is located in the InfoCenter.

After a TaskInfo object is created, it can be submitted to the scheduler for task creation by calling the Scheduler.create() method.

Specifying time intervals. setStartTimeInterval(), setStartByInterval(), and setRepeatInterval() methods all take a String parameter that represents time interval. Time intervals are calculated using *user calendars*.

TaskHandler interface

A task handler is a user-defined stateless session bean that is called by tasks created using a BeanTaskInfo object. A task handler bean uses the following home and remote interfaces, which are defined in the deployment descriptor using the Application Assembly Tool or WebSphere Studio Application Developer:

```
com.ibm.websphere.scheduler.TaskHandlerHome
com.ibm.websphere.scheduler.TaskHandler
```

The bean itself needs to implement the process() method defined in the remote interface. For details, see the Interface TaskHandler in the Javadoc. The javadoc is located in the InfoCenter.

If a task is created using a BeanTaskInfo object, the process() method on the TaskHandler session bean is called whenever the task runs. Because the TaskStatus object for the task is passed as a parameter, the handler can make use of the saved UserContext field, as well as determine information about the task, such as when it will fire next, the number of repeats remaining, and so on.

NotificationSink interface

A notification sink is a user-defined stateless session bean that is called by tasks when their state changes throughout the bean's lifecycle. A notification sink bean uses the following home and remote interfaces, which are defined in the deployment descriptor using the Application Assembly Tool or WebSphere Studio Application Developer:

```
com.ibm.websphere.scheduler.NotificationSinkHome
com.ibm.websphere.scheduler.NotificationSink
```

The bean itself needs to implement the handleEvent() method defined in the remote interface. For details, see the Interface NotificationSink in the Javadoc. The Javadoc is located in the InfoCenter.

A NotificationSink provides an event notification callback on a task-by-task basis. A notification sink is set on the TaskInfo interface, using the setNotificationSink() method. If a notification sink is not specified on a task, all notifications are lost; however, the status of a task can be determined by calling the getStatus() method from the Scheduler interface. A notification callback is made for each of the following events:

- Scheduled
- Suspended
- Resumed
- Fired
- Fire
- Failed
- Complete
- Purged

UserCalendar interface

A user calendar is a user-defined stateless session bean that is called by tasks when they need to calculate date-related values. A user calendar bean uses the following home and remote interfaces, which are defined in the deployment descriptor using the Application Assembly Tool or WebSphere Studio Application Developer:

```
com.ibm.websphere.scheduler.UserCalendarHome
com.ibm.websphere.scheduler.UserCalendar
```

The bean itself needs to implement the applyDelta() and validate() methods defined in the remote interface. For details, see the Interface UserCalendar in the Javadoc. The Javadoc is located in the InfoCenter.

User calendars can be used to calculate time intervals, such as the time between when a repeating task fires and the next time it fires. A user calendar takes a java.util.Date object and applies the interval string. The resulting object is a java.util.Date object that is an incremented date.

User calendars are set by the setUserCalendar() method on the TaskInfo interface, and called by the scheduler run-time code when a delta calculation is necessary.

The following methods on the TaskInfo interface specify delta strings that use the user calendar for calculation:

- setStartTimeInterval
- setStartByInterval
- setRepeatInterval

Default user calendar
If a user calendar has not been specified using the TaskInfo.setUserCalendar() method, a default user calendar is used. The default calendar allows for simple delta specifications, such as seconds, minutes, hours, days, and months. See the Javadoc for details on the default calendar. The Javadoc is located in the InfoCenter.

Calendar specifiers
A single user calendar can contain logic for multiple calendars. Which calendar is used is determined by a string that acts as the specifier. For example, a bean might be implemented to recognize the interval ″day″, with a specifier that determines whether to calculate ″day″ as a standard calendar day, or as a business day.

Internationalization and timezones
Scheduler makes use of the java.util.Date class when storing and processing dates. Internally, this class saves the time as milliseconds since the Epoch, Greenwhich Mean Time. Since the Date is not converted to local time until converted to a string, scheduler respects the timezone where the date was created.

Writing user calendars
Because the user calendar is a stateless session bean, the same J2EE Programming model available to other session beans is available to the user calendar as well.

Chapter 11. Using shared work areas

The WorkArea service enables application developers to implicitly propagate information beyond the information passed in remote calls. Applications can create a work area, insert information into it, and make remote invocations. The work area is propagated with each remote method invocation, eliminating the need to explicitly include an appropriate argument in the definition of each method. The methods on the server side can use or ignore the information in the work area as appropriate.

Before proceeding with the steps to implement work areas, as described below, review the topic “WorkArea service - Overview”.

Steps for this task

1. “Developing applications that use work areas” on page 264.

   Applications interact with the WorkArea service by implementing the UserWorkArea interface.
2. “Managing the work area service” on page 273.

   The WorkArea service is managed using the administrative console.

WorkArea service - Overview

One of the foundations of distributed computing is the ability to pass information, typically in the form of arguments to remote methods, from one process to another. When application-level software is written over middleware services, many of the services rely on information beyond that passed in the application’s remote calls. Such services often make use of the implicit propagation of private information in addition to the arguments passed in remote requests; two typical users of such a feature are security and transaction services. Security certificates or transaction contexts are passed without the knowledge or intervention of the user or application developer. The implicit propagation of such information means that application developers do not have to manually pass the information in method invocations, which makes development less error-prone, and the services requiring the information do not have to expose it to application developers. Information such as security credentials can remain secret.

The WorkArea service gives application developers a similar facility. Applications can create a work area, insert information into it, and make remote invocations. The work area is propagated with each remote method invocation, eliminating the need to explicitly include an appropriate argument in the definition of every method. The methods on the server side can use or ignore the information in the work area as appropriate. If methods in a server receive a work area from a client and subsequently invoke other remote methods, the work area is transparently propagated with the remote requests. When the creating application is done with the work area, it terminates it.

There are two prime considerations in deciding whether to pass information explicitly as an argument or implicitly by using a work area. These considerations are:

- Pervasiveness: Is the information used in a majority of the methods in an application?
- Size: Is it reasonable to send the information even when it will not be used?

When information is sufficiently pervasive that it is easiest and most efficient to make it available everywhere, application programmers can use the WorkArea service to simplify programming and maintenance of code. The argument does not need to go onto every argument list. It is much easier to put the value into a work area and propagate it automatically. This is especially true for methods that simply pass the value on but do nothing with it. Methods that make no use of the propagated information simply ignore it.

Work areas can hold any kind of information, and they can hold an arbitrary number of individual pieces of data, each stored as a property.

Work area property modes

The information in a work area consists of a set of properties; a property consists of a key-value-mode triple. The key-value pair represents the information contained in the property; the key is a name by which the associated value is retrieved. The mode determines whether the property can be removed or modified.

Property modes

There are four possible mode values for properties, as shown in the following code example:

Code example: The PropertyModeType definition

```
public final class PropertyModeType {
   public static final PropertyModeType normal;
   public static final PropertyModeType read_only;
   public static final PropertyModeType fixed_normal;
   public static final PropertyModeType fixed_readonly;
};
```

A property’s mode determines three things:

- Whether the value associated with the key can be modified
- Whether the property can be deleted
- Whether the mode associated with the key-value pair can be modified

The two read-only modes forbid changes to the information in the property; the two fixed modes forbid deletion of the property.

The WorkArea service does not provide methods specifically for the purpose of modifying the value of a key or the mode associated with a property. To change information in a property, applications simply rewrite the information in the property; this has the same effect as updating the information in the property. The mode of a property governs the changes that can be made. Modifying key-value pairs describes the restrictions each mode places on modifying the value and deleting the property. Changing modes describes the restrictions on changing the mode.

Changing modes

The mode associated with a property can be changed only according to the restrictions of the original mode. The read-only and fixed read-only properties do not permit modification of the value or the mode. The fixed normal and fixed read-only modes do not allow the property to be deleted. This set of restrictions leads to the following permissible ways to change the mode of a property within the lifetime of a work area:

- If the current mode is normal, it can be changed to any of the other three modes: fixed normal, read-only, fixed read-only.
- If the current mode is fixed normal, it can be changed only to fixed read-only.
- If the current mode is read-only, it can be changed only by deleting the property and re-creating it with the desired mode.
- If the current mode is fixed read-only, it cannot be changed.
- If the current mode is not normal, it cannot be changed to normal. If a property is set as fixed normal and then reset as normal, the value is updated but the mode remains fixed normal. If a property is set as fixed normal and then reset as either read-only or fixed read-only, the value is updated and the mode is changed to fixed read-only.

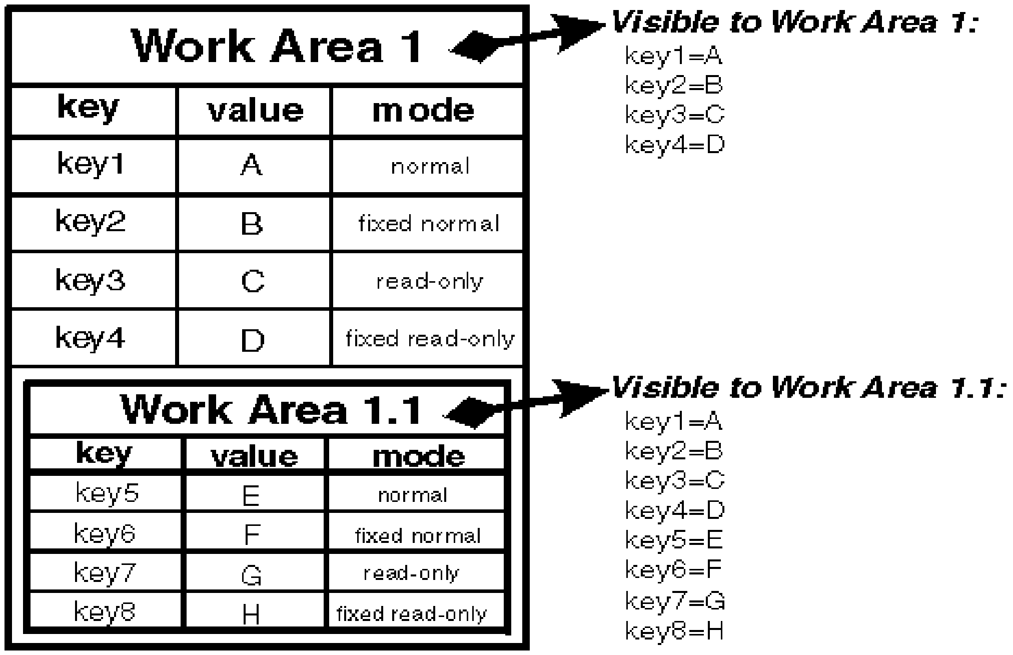
Note: The key, value, and mode of any property can be effectively changed by terminating (completing) the work area in which the property was created and creating a new work area. Applications can then insert new properties into the work area. This is not precisely the same as changing the value in the original work area, but some applications can use it as an equivalent mechanism.

Nested work areas

Applications can nest work areas. When an application creates a work area, a work area context is associated with the creating thread. If the application thread creates another work area, the new work area is nested within the existing work area and becomes the current work area. Nested work areas allow applications to define and scope properties for specific tasks without having to make them available to all parts of the application. All properties defined in the original, enclosing work area are visible to the nested work area. The application can set additional properties within the nested work area that are not part of the enclosing work area.

An application working with a nested work area does not actually see the nesting of enclosing work areas. The current work area appears as a flat set of properties that includes those from enclosing work areas. In the figure below, the enclosing work area holds several properties and the nested work area holds additional properties. From the outermost work area, the properties set in the nested work area are not visible. From the nested work area, the properties in both work areas are visible.

Defining new properties in nested work areas



Work Area 1

key	value	mode
key1	A	normal
key2	B	fixed normal
key3	C	read-only
key4	D	fixed read-only

Work Area 1.1

key	value	mode
key5	E	normal
key6	F	fixed normal
key7	G	read-only
key8	H	fixed read-only

Visible to Work Area 1:
key1=A
key2=B
key3=C
key4=D

Visible to Work Area 1.1:
key1=A
key2=B
key3=C
key4=D
key5=E
key6=F
key7=G
key8=H

Nesting can also affect the apparent settings of the properties. Properties can be deleted from or directly modified only within the work areas in which they were set, but nested work areas can also be used to temporarily override information in the property without having to modify the property. Depending on the modes associated with the properties in the enclosing work area, the modes and the values of keys in the enclosing work area can be overridden within the nested work area.

The mode associated with a property when it is created determines whether nested work areas can override the property. From the perspective of a nested work area, the property modes used in enclosing work areas can be grouped as follows:

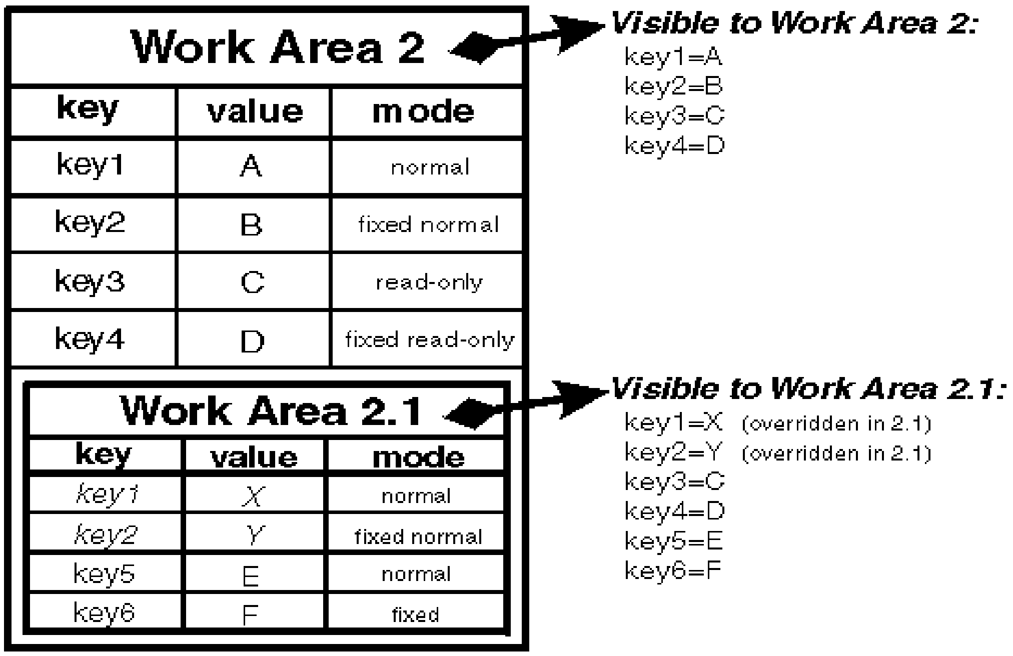
- Modes that permit a nested work area to override the mode or the value of a key locally. The modes that permit overriding are:
  - Normal
  - Fixed normal
- Modes that do not permit a nested work area to override the mode or the value of a key locally. The modes that do not permit overriding are:
  - Read-only
  - Fixed read-only

If an enclosing work area defines a property with one of the overridable modes, a nested work area can specify a new value for the key or a new mode for the property. The new value or mode becomes the value or mode seen by subsequently nested work areas. Changes to the mode are governed by the restrictions described in Changing modes. If an enclosing work area defines a property with one of the modes that cannot be overridden, no nested work area can specify a new value for the key.

A nested work area can delete properties from enclosing work areas, but the changes persist only for the duration of the nested work area. When the nested work area is completed, any properties that were added in the nested area vanish and any properties that were deleted from the nested area are restored.

The following figure illustrates the overriding of properties from an enclosing work area. The nested work area redefines two of the properties set in the enclosing work area. The other two cannot be overridden. The nested work area also defines two new properties. From the outermost work area, the properties set or redefined in the nested work are not visible. From the nested work area, the properties in both work areas are visible, but the values seen for the redefined properties are those set in the nested work area.

Redefining existing properties in nested work areas



Distributed work areas

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, by creating additional nested work areas; this information will be propagated to any remote objects it invokes. However, no changes made to a nested work area on a target object are propagated back to the calling object. The caller's work area is unaffected by changes made in the remote method.

WorkArea service: Special considerations

Developers who use work areas should consider the following issues that could potentially cause problems: interoperability between the EJB and CORBA programming models; and the use of work areas with Java's Abstract Windowing Toolkit.

EJB and CORBA interoperability

Although the WorkArea service can be used across the EJB and CORBA programming models, many composed data types cannot be successfully used across those boundaries. For example, if a SimpleSampleCompany instance is passed from the WebSphere environment into a CORBA environment, the CORBA application can retrieve the SimpleSampleCompany object encapsulated within a CORBA Any object from the work area, but it cannot extract the value from it. Likewise, an IDL-defined struct defined within a CORBA application and set into a work area will not be readable by an application using the UserWorkArea class. Applications can avoid this incompatibility by directly setting only primitive types, like integers and strings, as values in work areas, or by implementing complex values with structures designed to be compatible, like CORBA valuetypes. Also, CORBA Anys that contains either the tk_null or tk_void typecode can be set into the work area by using the CORBA interface, but the work-area specification cannot allow the J2EE implementation to return null on a lookup that retrieves these CORBA-set properties without incorrectly implying that there is no value set for the corresponding key. If a J2EE application tries to retrieve CORBA-set properties that are non-serializable, or contain CORBA nulls or void references, the com.ibm.websphere.workarea.IncompatibleValue exception is raised.

Using work areas with Java's Abstract Windowing Toolkit (AWT)

Work areas must be used cautiously in applications that use Java's Abstract Windowing Toolkit (AWT). The AWT implementation is multithreaded, and work areas begun on one thread are not available on another. For example, if a program begins a work area in response to an AWT event, such as pressing a button, the work area might not be available to any other part of the application after the execution of the event completes.

Developing applications that use work areas

Applications interact with the WorkArea service by implementing the "UserWorkArea interface". This interface defines all of the methods used to create, manipulate, and complete work areas:

Steps for this task

1. "Accessing the WorkArea service" on page 266.
2. "Beginning a new work area" on page 266.
3. "Setting properties in a work area" on page 267.
4. "Using a work area to manage local work" on page 268.
5. "Completing a work area" on page 272.

Usage scenario

An example application, the "Example: WorkArea SimpleSample application" on page 265, is used throughout this documentation to illustrate these tasks

UserWorkArea interface

Applications interact with the WorkArea service by implementing the UserWorkArea interface. This interface, shown below, defines all of the methods used to create, manipulate, and terminate work areas:

```
package com.ibm.websphere.workarea;

public interface UserWorkArea {
   void begin(String name);
   void complete() throws NoWorkArea, NotOriginator;

   String getName();
   String[] retrieveAllKeys();
   void set(String key, java.io.Serializable value)
      throws NoWorkArea, NotOriginator, PropertyReadOnly;
   void set(String key, java.io.Serializable value, PropertyModeType mode)
      throws NoWorkArea, NotOriginator, PropertyReadOnly;
   java.io.Serializable get(String key);
   PropertyModeType getMode(String key);
   void remove(String key)
      throws NoWorkArea, NotOriginator, PropertyFixed;
}
```

Note: EJB applications can use the UserWorkArea interface only within the implementation of methods in the remote interface; likewise, servlets can use the interface only within the service method of the HTTPServlet class. Use of work areas within any lifecycle method of a servlet or enterprise bean is considered a deviation from the work area programming model and is not supported.

Exceptions

The WorkArea service defines the following exceptions for use with the UserWorkArea interface:

NoWorkArea
: Thrown when a request requires an associated work area but none is present.

NotOriginator
: Raised when a request attempts to manipulate the contents of an imported work area.

PropertyReadOnly
: Raised when a request attempts to modify a read-only or fixed read-only property.

PropertyFixed
: Raised by the remove method when the designated property has one of the fixed modes.

Example: WorkArea SimpleSample application

In this example, the client creates a work area and inserts two properties into the work area: a site identifier and a priority. The site-identifier is set as a read-only property; the client does not allow recipients of the work area to override the site identifier. This property consists of the key company and a static instance of a SimpleSampleCompany object. The priority property consists of the key priority and a static instance of a SimpleSamplePriority object. The object types are defined as shown in the following code example

```
public static final class SimpleSampleCompany {
   public static final SimpleSampleCompany Main;
   public static final SimpleSampleCompany NewYork_Sales;
   public static final SimpleSampleCompany NewYork_Development;
   public static final SimpleSampleCompany London_Sales;
   public static final SimpleSampleCompany London_Development;
}

public static final class SimpleSamplePriority {
   public static final SimpleSamplePriority Platinum;
   public static final SimpleSamplePriority Gold;
   public static final SimpleSamplePriority Silver;
   public static final SimpleSamplePriority Bronze;
   public static final SimpleSamplePriority Tin;
}
```

The client then makes an invocation on a remote object. The work area is automatically propagated; none of the methods on the remote object take a work area argument. On the remote side, the request is first handled by the SimpleSampleBean; the bean first reads the site identifier and priority properties from the work area. The bean then intentionally attempts, and fails, both to write directly into the imported work area and to override the read-only site-identifier property.

The SimpleSampleBean successfully begins a nested work area, in which it overrides the client's priority, then calls another bean, the SimpleSampleBackendBean. The SimpleSampleBackendBean reads the properties from the work area, which contains the site identifier set in the client and priority set in the SimpleSampleBean. Finally, the SimpleSampleBean completes its nested work area, writes out a message based on the site-identifier property, and returns.

The implementation of this application is discussed in the topic, "Developing applications that use work areas" on page 264.

Accessing the WorkArea service

The WorkArea service provides a JNDI binding to an implementation of the UserWorkArea interface under the name java:comp/websphere/UserWorkArea. Applications that need to access the service can perform a lookup on that JNDI name, as shown in the following code example:

Usage scenario

```
import com.ibm.websphere.workarea.*;
import javax.naming.*;

public class SimpleSampleServlet {
  ...

  InitialContext jndi = null;
  UserWorkArea userWorkArea = null;
  try {
     jndi = new InitialContext();
     userWorkArea = (UserWorkArea)jndi.lookup(
        "java:comp/websphere/UserWorkArea");
  }
  catch (NamingException e) { ... }
}
```

What to do next

The next step is to use the begin() method to create a new work area and associate it with the calling thread, as described in the topic "Beginning a new work area".

Beginning a new work area

Before you begin

Be sure that your client has a reference to the UserWorkArea interface, as described in the topic "Accessing the WorkArea service"

Use the begin() method to create a new work area and associate it with the calling thread. The begin() method takes a string as an argument; the string is used to name the work area. The argument must not be null, which causes the java.lang.NullPointer exception to be raised. In the following code example, the application begins a new work area with the name SimpleSampleServlet:

```
public class SimpleSampleServlet {
...
   try {
      ...
      userWorkArea = (UserWorkArea)jndi.lookup(
         "java:comp/websphere/UserWorkArea");
   }
   ...

   userWorkArea.begin("SimpleSampleServlet");
   ...
}
```

The begin() method is also used to create nested work areas; if a work area is associated with a thread when the begin() method is called, the method creates a new work area nested within the existing work area.

The WorkArea service makes no use of the names associated with work areas; You can name work areas in any way that you choose. Names are not required to be unique, but the usefulness of the names for debugging is enhanced if the names are distinct and meaningful within the application. Applications can use the getName() method to return the name associated with a work area by the begin() method.

What to do next

"Using a work area to manage local work" on page 268.

Setting properties in a work area

An application with a current work area can insert properties into the work area and retrieve the properties from the work area. The UserWorkArea interface provides two set() methods for setting properties and a get() method for retrieving properties. The two-argument set() method inserts the property with the property mode of normal. The three-argument set() method takes a property mode as the third argument. (See ″Setting property modes″, later in this topic.)

Both set() methods take the key and the value as arguments. The key is a String; the value is an object of the type java.io.Serializable. None of the arguments can be null, which causes the java.lang.NullPointer exception to be raised.

The "Example: WorkArea SimpleSample application" on page 265 uses objects of two classes, the SimpleSampleCompany class and the SimpleSampleProperty class, as values for properties. The SimpleSampleCompany class is used for the site identifier, and the SimpleSamplePriority class is used for the priority. These classes are shown in following code example:

```
public class SimpleSampleServlet {
   ...
   userWorkArea.begin("SimpleSampleServlet");

   try {
      // Set the site-identifier (default is Main).
      userWorkArea.set("company",
         SimpleSampleCompany.Main, PropertyModeType.read_only);

      // Set the priority.
      userWorkArea.set("priority", SimpleSamplePriority.Silver);
   }

   catch (PropertyReadOnly e) {
     // The company was previously set with the read-only or
```

```
      // fixed read-only mode.
      ...
    }

     catch (NotOriginator e) {
      // The work area originated in another process,
      // so it can't be modified here.
      ...
    }

    catch (NoWorkArea e) {
       // There is no work area begun on this thread.
       ...
    }

    // Do application work.
    ...
}
```

The get() method takes the key as an argument and returns a Java Serializable object as the value associated with the key. For example, to retrieve the value of the company key from the work area, the code example above uses the get() method on the work area to retrieve the value.

Setting property modes. The two-argument set() method on the UserWorkArea interface takes a key and a value as arguments and inserts the property with the default property mode of normal. To set a property with a different mode, applications must use the three-argument set() method, which takes a property mode as the third argument. The values used to request the property modes are as follows:

- **Normal**: PropertyModeType.normal
- **Fixed normal**: PropertyModeType.fixed_normal
- **Read-only**: PropertyModeType.read_only
- **Fixed read-only**: PropertyModeType.fixed_readonly

Using a work area to manage local work

Before you begin

Be sure that your client has a reference to the UserWorkArea interface, as described in the topic “Accessing the WorkArea service” on page 266

In a business application that uses work areas, server objects typically retrieve the work area properties and use them to guide local work.

Steps for this task

1. “Retrieving the name of the active work area” on page 269
   This step determines whether the calling thread is associated with a work area.
2. “Overriding work area properties” on page 269
   Server objects can override client work area properties by creating their own, nested work area.
3. “Retrieving work area properties” on page 271
4. “Retrieving a list of all keys in a work area” on page 271
5. “Querying the mode of a work area property” on page 272
6. “Deleting a work area property” on page 272
7. “Completing a work area” on page 272

Usage scenario

The server side of the "Example: WorkArea SimpleSample application" on page 265 accepts remote invocations from clients. With each remote call, the server also gets a work area from the client if the client has created one. The work area is propagated transparently. None of the remote methods includes the work area on its argument list.

In the example application, the server objects use the work area interface for demonstration purposes only. For example, the SimpleSampleBean intentionally attempts to write directly to an imported work area, which triggers the NotOriginator exception. Likewise, the bean intentionally attempts to mask the read only SimpleSampleCompany, which triggers the PropertyReadOnly exception. The SimpleSampleBean also nests a work area and successfully overrides the priority property before invoking the SimpleSampleBackendBean. A true business application would extract the work area properties and use them to guide the local work. The SimpleSampleBean mimics this by writing a message that function is denied when a request emanates from a sales environment.

Retrieving the name of the active work area

Applications use the getName() method on the UserWorkArea interface to retrieve the name of the current work area. This is the recommended method for determining whether the thread is associated with a work area; if the thread is not associated with a work area, the getName() method returns null. In the following code example, the name of the work area corresponds to the name of the class in which the work area was begun.

Usage scenario

```
public class SimpleSampleBeanImpl implements SessionBean {

    ...

    public String [] test() {
       // Get the work-area reference from JNDI.
       ...

       // Retrieve the name of the work area. In this example,
       // the name is used to identify the class in which the
       // work area was begun.
       String invoker = userWorkArea.getName();
       ...
   }
}
```

Overriding work area properties

Work areas are inherently associated with the process that creates them. In the sample application, the client begins a work area and sets into it the site-identifier and priority properties. This work area is propagated to the server when the client makes a remote invocation.

Applications nest work areas in order to temporarily override properties imported from a client process. The nesting mechanism is automatic; invoking begin on the UserWorkArea interface from within the scope of an existing work area creates a nested work area that inherits the properties from the enclosing work area. Properties set into the nested work area are strictly associated with the process in which the work area was begun; the nested work area must be completed within the process that created them. If a work area is not completed by the creating process, the work-area facility terminates the work area when the process exits. After a nested work area is completed, the original view of the enclosing work area is restored. However, the view of the complete set of work areas associated with a thread cannot be decomposed by downstream processes.

Applications set properties into a work area using property modes in ensure that a particular property is fixed (not removable) or read-only (not overrideable) within the scope of the given work area.

Usage scenario

In the following code example, the server-side sample bean attempts to write directly to the imported work area; this action is not permitted, and the NotOriginator exception is thrown. The sample bean must begin its own work area in order to override any imported properties, as shown in the second code example.

```
public class SimpleSampleBeanImpl implements SessionBean {

   public String [] test() {
     ...
     String invoker = userWorkArea.getName();

     try {
       userWorkArea.set("key", "value");
     }
     catch (NotOriginator e) {
     }
     ...
  }
}
```

The following code example demonstrates beginning a nested work area, using the name of the creating class to identify the nested work area.

```
public class SimpleSampleBeanImpl implements SessionBean {

   public String [] test() {
      ...
      String invoker = userWorkArea.getName();
      try {
        userWorkArea.set("key", "value");
      }
      catch (NotOriginator e) {
      }

      // Begin a nested work area. By using the name of the creating
      // class as the name of the work area, we can avoid having
      // to explicitly set the name of the creating class in
      // the work area.
      userWorkArea.begin("SimpleSampleBean");

      ...
   }
}
```

In the example application, the client sets the site-identifier property as read-only; that guarantees that the request will always be associated with the client's company identity. A server cannot override that value in a nested work area. In the following code example, the SimpleSampleBean attempts to change the value of the site-identifier property in the nested work area it created.

```
public class SimpleSampleBeanImpl implements SessionBean {

  public String [] test() {
      ...
      String invoker = userWorkArea.getName();
      try {
        userWorkArea.set("key", "value");
      }
      catch (NotOriginator e) {
      }

      // Begin a nested work area.
      userWorkArea.begin("SimpleSampleBean");

      try {
        userWorkArea.set("company",
                         SimpleSampleCompany.London_Development);
      }
      catch (NotOriginator e) {
      }
      ...
  }
}
```

Retrieving work area properties

Properties can be retrieved from a work area by using the get() method. This method is intentionally lightweight; there are no declared exceptions to handle. If there is no active work area, or if there is no such property set in the current work area, the get() method returns null.

Note: The get() method can raise a NotSerializableError in the relatively rare scenario in which CORBA clients set composed data types and invoke enterprise-bean interfaces.

Usage scenario

The following example shows the retrieval of the site-identifier and priority properties by the SimpleSampleBean. Recall that one property was set into an outer work area by the client, and the other property was set into the nested work area by the server-side bean; the nesting is transparent to the retrieval of the properties.

```
public class SimpleSampleBeanImpl implements SessionBean {

    public String [] test() {
      ...

      // Begin a nested work area.
      userWorkArea.begin("SimpleSampleBean");
      try {
        userWorkArea.set("company",
                         SimpleSampleCompany.London_Development);
      }
      catch (NotOriginator e) {
      }

      SimpleSampleCompany company =
         (SimpleSampleCompany) userWorkArea.get("company");
      SimpleSamplePriority priority =
         (SimpleSamplePriority) userWorkArea.get("priority");
       ...
   }
}
```

Retrieving a list of all keys in a work area

The UserWorkArea interface provides the retrieveAllKeys() method for retrieving a list of all the keys visible from a work area. This method takes no arguments and returns an array of strings. This method returns null if there is no work area associated with the thread. If there is an associated work area containing no properties, the method returns an array of size 0.

Querying the mode of a work area property

The UserWorkArea interface provides the getMode() method for determining the mode of a specific property. This method takes the property's key as an argument and returns the mode as a PropertyModeType object. (See "Setting properties in a work area" on page 267 for more information on names of mode types.) If the specified key does not exist in the work area, the method returns PropertyModeType.normal, indicating that the property can be set and removed without error.

Deleting a work area property

The UserWorkArea interface provides the remove() method for deleting a property from the current scope of a work area. If the property was initially set in the current scope, removing it deletes the property. If the property was initially set in an enclosing work area, removing it deletes the property until the current scope is completed. When the current work area is completed, the deleted property is restored.

The remove() method takes the property's key as an argument. Only properties with the modes normal and read-only can be removed. Attempting to remove a fixed property causes the PropertyFixed exception to be thrown. Attempting to remove properties in work areas that originated in other processes causes the NotOriginator exception to be thrown.

Completing a work area

After an application has finished using a work area, it must complete the work area by calling the complete() method on the UserWorkArea interface. This terminates the association with the calling thread and destroys the work area. If the complete method is called on a nested work area, the nested work area is terminated and the parent work area becomes the current work area. If there is no work area associated with the calling thread, a NoWorkArea exception is thrown.

Every work area must be completed, and work areas can be completed only by the originating process. For example, if a server attempts to call the complete() method on a work area that originated in a client, a NotOriginator exception is thrown. Work areas created in a server process are never propagated back to an invoking client process.

Note: The WorkArea service claims full local-remote transparency. Even if two beans happen to be deployed in the same server, and therefore the same JVM and process, a work area begun on an invocation from another is completed and the bean in which the request origininated is always in the same state after any remote call.

Usage scenario

The following code example shows the completion of the work area created in the client application.

```
public class SimpleSampleServlet {
   ...
   userWorkArea.begin("SimpleSampleServlet");
   userWorkArea.set("company",
       SimpleSampleCompany.Main, PropertyModeType.read_only);
   userWorkArea.set("priority", SimpleSamplePriority.Silver);
   ...

   // Do application work.
   ...

   // Terminate the work area.
   try {
      userWorkArea.complete();
   }

   catch (NoWorkArea e) {
      // There is no work area associated with this thread.
      ...
   }

   catch (NotOriginator e) {
      // The work area was imported into this process.
      ...
   }
  ...
}
```

The following code example shows the sample application completing the nested work area it created earlier in the remote invocation.

```
public class SimpleSampleBeanImpl implements SessionBean {

    public String [] test() {
      ...

      // Begin a nested work area.
      userWorkArea.begin("SimpleSampleBean");
      try {
        userWorkArea.set("company",
                        SimpleSampleCompany.London_Development);
      }
      catch (NotOriginator e) {
      }

      SimpleSampleCompany company =
         (SimpleSampleCompany) userWorkArea.get("company");
      SimpleSamplePriority priority =
         (SimpleSamplePriority) userWorkArea.get("priority");

      // Complete all nested work areas before returning.
      try {
        userWorkArea.complete();
      }
      catch (NoWorkArea e) {
      }
      catch (NotOriginator e) {
      }
   }
}
```

Managing the work area service

The WorkArea service is managed using the administrative console. There are two administrative tasks associated with work areas:

- "Enabling the WorkArea service" on page 274. The WorkArea Service is enabled by default on both clients and servers.
- "Managing the size of work areas" on page 275. Applications can set maximum sizes on each work area to be sent and to be accepted.

Enabling the WorkArea service

For an application to take advantage of work areas, the WorkArea service must be enabled for both clients and servers. In both cases, the service is enabled by default.

Steps for this task

1. Enable (or disable) the use of work areas on a server:
   a. Start the administrative console.
   b. Select **Servers** > *server_name* > **WorkArea Service**.
   c. Select or clear the **Startup** checkbox.

      This specifies whether or not the server should automatically start the WorkArea service when the server starts.
2. Enable (or disable) the use of work areas on a client:

   Set the com.ibm.websphere.workarea.enabled property to TRUE or FALSE before starting the client. For example, edit the launchClient script in the $WAS_HOME/bin directory and add the following to the Java invocation:

   ```
   -Dcom.ibm.websphere.workarea.enabled=false
   ```

WorkArea service settings

Use this page to manage the work area service.

The work area service manages the scope and implicit propagation of application context.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Work Area Service** .

Startup: Specifies whether the server will attempt to start the work area service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the work area service automatically.

Cleared
The server does not try to start the work area service. If work areas are to be used on this application server, the system administrator must start the service manually or select this property then restart the server.

Maximum Send Size: Specifies the maximum size of data that can be sent within a single work area.

Data type
Integer

Units Bytes

Default
32767

Range -1 to no limit

-1 Default.

0	No limit.

Maximum Receive Size: Specifies the maximum size of data that can be received by a single work area.

Data type
Integer

Units Bytes

Default
32767

Range -1 to no limit

-1	Default.
0	No limit.

Managing the size of work areas

Applications can set maximum sizes on each work area to be sent or received. By default, the maximum size of a work area that is sent by a client and received, then possibly re-sent, by a server is 32,768 bytes. You can change this size as described in this topic.

Steps for this task

1. Change the size of the work area that can be sent or received by a server:
   a. Start the administrative console.
   b. Select **Servers** > *server_name* > **WorkArea Service**.
   c. Enter a new value in the **maxSendSize** field to modify the size of the work area that this server can send, or enter a new value in the **maxReceiveSize** field to modify the size of the work area that this server can accept.
2. Change the size of the work area that can be sent by a client:

   Set the com.ibm.websphere.workarea.maxSendSize property to the desired number of bytes before starting the client. This can be done in several ways. For example, to set the maximum size to 10,000 bytes, edit the launchClient script in the $WAS_HOME/bin directory and add the following to the Java invocation:

   ```
   -Dcom.ibm.websphere.workarea.maxSendSize=10000
   ```

Results

The maximum size that can be specified is determined by the maximum value expressible in the Java Integer data type, 2,147,483,647. The smallest maximum size that can be specified is 1. Using a maximum size of 1 byte effectively means that no requests associated with the work area can leave the system or enter another system. A value of 0 means that no limit is imposed. A value of -1 means that the default value is to be honored. The default value is also used if an invalid value or a malformed property is specified.

Chapter 12. Using the transaction service

These topics provide information about using transactions with WebSphere applications

WebSphere applications can use transactions to coordinate multiple updates to resources as atomic units (as indivisible units of work) such that all or none of the updates are made permanent.

In IBM WebSphere Application Server, transactions are handled by three main components:

- A transaction manager that supports the enlistment of recoverable XAResources and ensures that each such resource is driven to a consistent outcome either at the end of a transaction or after a failure and restart of the application server.
- A container in which the J2EE application runs. The container manages the enlistment of XAResources on behalf of the application when the application performs updates to transactional resource managers (for example, databases). Optionally, the container can control the demarcation of transactions for enterprise beans configured for container-managed transactions.
- An application programming interface (UserTransaction) that is available to bean-managed enterprise beans and servlets. This allows such application components to control the demarcation of their own transactions.

For more information about using transactions with WebSphere applications, see the following topics:

- "Transaction support in IBM WebSphere Application Server"
- "Developing components to use transactions" on page 285
- "Configuring transaction properties for an application server" on page 293
- "Using local transactions" on page 280
- "Setting transactional attributes in the deployment descriptor" on page 285
- "Using bean-managed transactions" on page 289
- "Managing active transactions" on page 299
- "Managing transaction logging for optimum server availability" on page 300
- "Troubleshooting transactions" on page 304
- "Transaction service exceptions" on page 305
- "UserTransaction interface - methods available" on page 306
- "Coordinating access to 1-PC and 2-PC-capable resources within the same transaction" on page 306
- Chapter 13, "Using the ActivitySession service", on page 311

Transaction support in IBM WebSphere Application Server

A transaction is unit of activity within which multiple updates to resources can be made atomic (as an indivisible unit of work) such that all or none of the updates are made permanent. For example, multiple SQL statements to a relational database are committed atomically by the database during the processing of an SQL COMMIT statement. In this case, the transaction is contained entirely within the database manager and can be thought of as a *resource manager local transaction (RMLT)*. In some contexts, a transaction is referred to as a logical unit of work (LUW).

The way that applications use transactions depends on the type of application component, as follows:

- A session bean can either use container-managed transactions (where the bean delegates management of transactions to the container) or bean-managed transactions (where the bean manages transactions itself).
- Entity beans use container-managed transactions.
- Web components (servlets) use bean-managed transactions.

IBM WebSphere Application Server is a transaction manager that supports the coordination of resource managers through their XAResource interface and participates in distributed global transactions with other OTS 1.2 compliant transaction managers (for example J2EE 1.3 application servers). WebSphere applications can also be configured to interact with databases, JMS queues, and JCA connectors through their *local transaction* support when distributed transaction coordination is not required.

Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface) and those that support only one-phase coordination (for example through a LocalTransaction interface). The IBM WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case.

With the Last Participant Support of IBM WebSphere Application Server Enterprise, you can coordinate the use of a single one-phase commit (1PC) capable resource with any number of two-phase commit (2PC) capable resources in the same global transaction. At transaction commit, the two-phase commit resources are prepared first using the two-phase commit protocol, and if this is successful the one-phase commit-resource is then called to commit(one_phase). The two-phase commit resources are then committed or rolled back depending on the response of the one-phase commit resource.

The ActivitySession service of IBM WebSphere Application Server Enterprise provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. It is a distributed context that can be used to coordinate multiple one-phase resource managers. The WebSphere EJB container and deployment tooling support ActivitySessions as an extension to the J2EE programming model. EJBs can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An application can then interact with a resource manager through its LocalTransaction interface for the period of a client-scoped ActivitySession rather than just the duration of an EJB method.

Resource manager local transaction (RMLT)

A resource manager local transaction (RMLT) is a resource manager's view of a local transaction; that is, it represents a unit of recovery on a single connection that is managed by the resource manager.

Resource managers include:

- Enterprise Information Systems that are accessed through a resource adapter, as described in the J2EE Connector Architecture 1.0.
- Relational databases that are accessed through a JDBC datasource.
- JMS queue and topic destinations.

Resource managers offer specific interfaces to enable control of their RMLTs. J2EE connector resource adapters that include support for local transactions provide a LocalTransaction interface to enable applications to request that the resource adapter commit or rollback RMLTs. JDBC datasources provide a Connection interface for the same purpose.

The boundary at which all RMLTs must be complete is defined in IBM WebSphere Application Server by a “Local transaction containment (LTC)” on page 280.

Global transactions

If an application uses two or more resources, then an external transaction manager is needed to coordinate the updates to both resource managers in a *global tansaction*.

Global transaction support is available to web and enterprise bean J2EE components. Enterprise bean components can be subdivided into beans that exploit container-managed transactions (CMT) or bean-managed transactions (BMT).

BMT enterprise beans and web components can use the Java Transaction API (JTA) UserTransaction interface to define the demarcation of a global transaction. The UserTransaction interface is obtained by a JNDI lookup of `java:comp/UserTransaction`. The UserTransaction is not available to the following components:

- CMT enterprise beans. Any attempt by such beans to obtain the interface results in an exception in accordance with the EJB specification.
- Client applications running outside the Web and EJB containers.

Ensure that programs that perform a JNDI lookup of the UserTransaction interface, use an InitialContext that resolves to a local implementation of the interface. Also ensure that such programs use a JNDI location appropriate for the EJB version.

Before the EJB 1.1 specification, the JNDI location of the UserTransaction interface was not specified. Each EJB container implementor defined it in an implementation-specific manner. Earlier versions of IBM WebSphere Application Server, up to and including Version 3.5.x (without EJB 1.1), bind the UserTransaction interface to a JNDI location of jta/usertransaction. IBM WebSphere Application Server, Version 4, and later releases, bind the UserTransaction interface at the location defined by EJB 1.1, which is java:comp/UserTransaction. IBM WebSphere Application Server, Version 5 no longer provides the jta/usertransaction binding within Web and EJB containers to applications at a J2EE level of 1.3 or later. For example, EJB 2.0 applications can use only the java:comp/UserTransaction location.

A web component or enterprise bean (CMT or BMT) can get the ExtendedJTATransaction interface through a lookup of `java:comp/websphere/ExtendedJTATransaction`. This interface provides access to the transaction identity and a mechanism to receive notification of transaction completion.

Local transaction containment (LTC)

A *local transaction containment (LTC)* is used to define the application server behavior in an unspecified transaction context.

(Unspecified transaction context is defined in the Enterprise JavaBeans 2.0 Specification.)

A LTC is a bounded unit-of-work scope within which zero, one, or more resource manager local transactions (RMLTs) can be accessed. The LTC defines the boundary at which all RMLTs must be complete; any incomplete RMLTs are resolved, according to policy, by the container. An LTC is local to a bean instance; it is not shared across beans even if those beans are managed by the same container. LTCs are started by the container before dispatching a method on a J2EE component (such as an enterprise bean or servlet) whenever the dispatch occurs in the absence of a global transaction context. LTCs are completed by the container depending on the application-configured LTC boundary; for example at the end of the method dispatch. There is no programmatic interface to the LTC support; rather LTCs are managed exclusively by the container and configured by the application deployer through transaction attributes in the application deployment descriptor.

A local transaction containment cannot exist concurrently with a global transaction. If application component dispatch occurs in the absence of a global transaction, the container always establishes an LTC. The only exceptions to this are as follows:

- Where application component dispatch occurs without container interposition; for example, for a stateless session bean `create`.
- J2EE 1.2 web components.
- J2EE 1.2 BMT enterprise beans.

A local transaction containment can be scoped to an ActivitySession context that lives longer than the enterprise bean method in which it is started, as described in “ActivitySession and transaction contexts” on page 315.

Using local transactions

Local transaction containment (LTC) support, and its configuration through local transaction extended deployment descriptors, gives IBM WebSphere Application Server application programmers a number of advantages. This topic describes those advantages and how they relate to the settings of the local transaction extended deployment descriptors. This topic also describes points to consider to help you best configure transaction support for some example scenarios that use local transactions.

Develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not need to use global transactions, it is often more efficient to deploy the bean with the Container Transaction deployment descriptor **Transaction** attribute set to `Not supported` instead of `Required`.

With the extended local transaction support of IBM WebSphere Application Server, applications can perform the same business logic in an unspecific transaction context as they can under a global transaction. An enterprise bean, for example, runs under an unspecified transaction context if it is deployed with a **Transaction** attribute of `Not supported` or `Never`.

The extended local transaction support provides a container-managed, implicit local transaction boundary within which application updates can be committed and their connections cleaned up by the container. Applications can then be designed with a greater degree of independence from deployment concerns. This makes using a **Transaction** attribute of `Supports` much simpler, for example, when the business logic may be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage regardless of whether or not the application runs under a transaction. The application can depend on the close behaving in the same way and not causing a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios running business logic under a **Transaction** policy of `Not supported` performs better than if it had been run under a `Required` policy. This benefit is exploited through the **Local Transactions - Resolution-control** extended deployment setting of `ContainerAtBoundary`. With this setting, application interactions with resource providers (such as databases) are managed within implicit RMLTs that are both started and ended by the container. The RMLTs are committed by the container at the configured **Local Transactions - Boundary**; for example at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

Use local transactions in a managed environment that guarantees clean-up.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default **Local Transactions - Resolution-control** extended deployment setting of `Application`. In this case, the container ensures connection cleanup at the boundary of the local transaction context.

J2EE specifications that describe application use of local transactions do so in the manner provided by the default setting of **Local Transactions - Resolution-control**=`Application` and **Local Transactions - Unresolved-action**=`Rollback`. By configuring the **Local Transactions - Unresolved-action** extended deployment setting to `Commit`, then any RMLTs started by the application but not completed when the local transaction containment ends (for example, when the method ends) are committed by the container. This usage applies to both servlets and enterprise beans.

Extend the duration of a local transaction beyond the duration of an EJB component method.

The J2EE specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. In IBM WebSphere Application Server Enterprise, you can exploit the **Local Transactions - Boundary** extended deployment setting to give the following advantages:

- Significantly extend the use-cases of RMLTs
- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

An ActivitySession is an IBM WebSphere Application Server Enterprise programming model extension that provides a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which can be controlled by a client. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is exploited through the **Local Transactions - Boundary** extended deployment setting of `ActivitySession`. Such extended RMLTs can remain under the control of the application or be managed by the container depending on the use of the **Local Transactions - Resolution-control** deployment descriptor setting.

Coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can exploit ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans under that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

To determine how best to configure the transaction support for an application, depending on what you want to do with transactions, consider the following points.

General points

- You want to start and end global transactions explicitly in the application (BMT session beans and servlets only).

  For a session bean, set the **Transaction type** to `Bean` (to use bean-managed transactions) in the component's deployment descriptor. (You do not need to do this for servlets.)
- You want to access only one XA or non-XA resource in a method.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Supports`.
- You want to access several XA resources atomically across one or more bean methods.

  In the Container transaction deployment descriptor, set **Transaction** to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resource in a method without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.
- You want to access several non-XA resource in a method and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application` and set **Local Transactions - Unresolved-action** to **Rollback**. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.

Points specific to IBM WebSphere Application Server Enterprise

- You want to access one of more non-XA resources across multiple EJB method calls without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container Transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to use a single non-XA resource and one or more XAResources.

  Use the Last Participant Support of IBM WebSphere Application Server Enterprise.

Local and global transaction considerations

Applications use resources, such as JDBC data sources or connection factories, that are configured through the Resources view of the IBM WebSphere Application Server Administrative Console. How these resources participate in a global transaction depends on the underlying transaction support of the resource provider. For example, a JDBC provider can provide either XA or non-XA versions of a data source. A non-XA data source can support only resource manager local transactions (RMLTs), but an XA data source can support two-phase commit coordination.

If an application uses two or more resource providers that support only RMLTs, then atomicity cannot be assured because of the one-phase nature of these resources. To ensure atomic behavior, the application should use resources that support XA coordination and should access them within a global transaction.

If an application uses only one RMLT, the atomic behavior can be guaranteed by the resource manager, which can be accessed under a local transaction containment context.

An application can also access a single resource manager under a global transaction context, even if that resource manager does not support the XA coordination. An application can do this, because IBM WebSphere Application Server performs an "only resource optimization" and interacts with the resource manager under a RMLT. Within a global transaction context, any attempt to use more than one resource provider that supports only RMLTs causes the global transaction to be rolled back.

At any moment, an instance of an enterprise bean can have work outstanding in either a global transaction context or a local transaction containment context, but never both. An instance of an enterprise bean can change from running under one type of context to the other (in either direction), if all outstanding work in the original context is complete. Any violation of this principle causes an exception to be thrown when the enterprise bean tries to start the new context.

Extended JTA support

Extended JTA support provides application programming interfaces additional to the UserTransaction interface that is defined in the JTA as part of the J2EE specification. Specifically, the API extensions provide the following functionality:

- Access to global and local transaction identifiers associated with the thread.

  The global id is based on the tid in CosTransactions::PropagationContext: and the local id identifies the transaction uniquely within the local JVM.
- A transaction synchronization callback that enables any J2EE component to register an interest in transaction completion.

  This can be used by advanced applications to flush updates before transaction completion and clear up state after transaction completion. J2EE (and related) specifications position this function generally as the domain of the J2EE containers. The exception is for CMT session beans, for which the EJB specification defines the SessionSynchronization interface. WebSphere provides this Enterprise functionality in recognition that more advanced applications can profit from the ability to receive such notifications.

An application uses a JNDI lookup of java:comp/websphere/ExtendedJTATransaction to get an ExtendedJTATransaction object, which it then uses as follows:

```
ExtendedJTATransaction exJTA = (ExtendedJTATransaction)ctx.lookup("
 java:comp/websphere/ExtendedJTATransaction");
SynchronizationCallback sync = new SynchronizationCallback();
exJTA.registerSynchronizationCallback(sync);
```

The ExtendedJTATransaction object supports the registration of one or more application-provided SynchronizationCallbacks. Each registered callback is called at the end of every transaction that runs on the application server (whether the transaction is started locally or imported).

The following information provides an overview of the interfaces provided by the Extended JTA support. For more detailed information, see the Javadoc provided with WAS Enterprise.

SynchronizationCallback interface

An object implementing this interface is enlisted once through the ExtendedJTATransaction interface, and receives notification of the completion of each subsequent transaction mediated by the transaction manager in the local JVM.

Although an object implementing this interface can run in a J2EE server, there is no specific J2EE component active when this object is called. So, the object has limited direct access to any J2EE resources. Specifically, it has no access to the java: namespace or to any container-mediated resource. Such an object can cache a reference to a J2EE component (for example, a stateless session bean) that it delegates to. The object would then have all the normal access to J2EE resources and could be used, for example, to acquire a JDBC connection and flush updates to a database during beforeCompletion.

ExtendedJTATransaction interface

A WebSphere programming model extension to the J2EE JTA support. An object implementing this interface is bound, by WebSphere J2EE containers that support this interface, at java:comp/websphere/ExtendedJTATransaction. Access to this object, when called from an EJB container, is not restricted to bean-managed transactions.

Developing components to use transactions

These topics provide information about developing WebSphere application components to use transactions

The way that applications use transactions depends on the type of application component, as follows:

- A session bean can either use container-managed transactions (where the bean delegates management of transactions to the container) or bean-managed transactions (where the bean manages transactions itself).
- Entity beans use container-managed transactions.
- Web components (servlets) use bean-managed transactions.

You configure whether a component uses container- or bean-managed transactions by setting an appropriate value on the Transaction type deployment attribute, as described in “Setting transactional attributes in the deployment descriptor”. You can also configure other transactional deployment descriptor attributes.

If you want a session bean to manage its own transactions, you must write the code that explicitly demarcates the boundaries of a transaction as described in “Using bean-managed transactions” on page 289.

Similarly, if you want a Web component to use transactions, you must write the code that explicitly demarcates the boundaries of a transaction as described in “Using bean-managed transactions” on page 289.

Setting transactional attributes in the deployment descriptor

Use this task to configure the transactional deployment descriptor attributes associated with an EJB or Web module, to enable a J2EE application to use transactions.

To set transactional attributes in the deployment descriptor for an application component (enterprise bean or servlet), complete the following steps:

Steps for this task

1. Start the Application Assembly Tool.
2. Create or edit the application EAR file.

   For example, to change attributes of an existing application, click **File-> Open** then select the EAR file.
3. In the navigation pane, select the component instance; for example:
   - For a session bean, expand ***ejb_module_instance*-> Session beans** then select the bean instance.
   - For a servlet, expand ***web_application*-> Web Components** then select the servlet instance.

   A property dialog notebook for the component is displayed in the property pane.

4. In the property pane, select the Advanced tab.
5. Set the **Transaction type** attribute, which defines the transactional manner in which the container invokes a method.

   You can set this attribute to Container or Bean, as follows:
   - For a session bean to use container-managed transactions, set Container
   - For a session bean to use bean-managed transactions, set Bean
   - For an entity bean, set Container
   - For a Web component (servlet), set Bean
6. In the property pane, select the IBM Extensions tab.
7. Configure J2EE component extensions attributes for extended local transaction containment.

   To enable management of local transaction containments, configure the following EJB extensions attributes. These attributes configure, for the component, the behaviour of the container's local transaction containment (LTC) environment that the container establishes whenever a global transaction is not present.

   Boundary
   Specifies the duration of a local transaction context. You can set this attribute to **Bean method** or **ActivitySession**, as described in ″Entity bean assembly settings″ (not in this document-see the InfoCenter for more information).

   Note: The ActivitySession option is not supported in the web container.

   Resolution control
   Specifies how the local transaction is to be resolved before the local transaction context ends: by the application through user code or by the EJB container. You can set this attribute to either **Application** or **ContainerAtBoundary**, as described in ″Entity bean assembly settings″ (not in this document-see the InfoCenter for more information).

   Unresolved action
   Specifies the action that the container must take when the local transaction context scope ends, if resources are uncommitted by an application in a local transaction and the **Resolution control** is set to Application. You can set this attribute to either **Commit** or **Rollback**, as described in ″Entity bean assembly settings″ (not in this document-see the InfoCenter for more information).
8. [For EJB components only] For container-managed transactions, configure how the container must manage the transaction boundaries when delegating a method invocation to an enterprise bean's business method:
   a. In the navigation pane, select **Container Transactions**.

      This displays a table of the methods for enterprise beans.
   b. For each method of the enterprise bean set the **Transaction attribute** attribute to an appropriate value, as defined in ″Container transaction assembly settings. ″ (not in this document-see the InfoCenter for more information).

   If the application uses ActivitySessions, how the container manages transaction boundaries when delegating a method invocation depends on both the **Transaction attribute** attribute, set here, and the **ActivitySession kind** attribute, as described in "Configuring ActivitySession deployment attributes for an enterprise bean" on page 325. For more detail about the relationship between these two properties, see "Combining transaction and ActivitySession container policies" on page 316.

Using local transactions

Local transaction containment (LTC) support, and its configuration through local transaction extended deployment descriptors, gives IBM WebSphere Application Server application programmers a number of advantages. This topic describes those advantages and how they relate to the settings of the local transaction extended deployment descriptors. This topic also describes points to consider to help you best configure transaction support for some example scenarios that use local transactions.

Develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not need to use global transactions, it is often more efficient to deploy the bean with the Container Transaction deployment descriptor **Transaction** attribute set to `Not supported` instead of `Required`.

With the extended local transaction support of IBM WebSphere Application Server, applications can perform the same business logic in an unspecific transaction context as they can under a global transaction. An enterprise bean, for example, runs under an unspecified transaction context if it is deployed with a **Transaction** attribute of `Not supported` or `Never`.

The extended local transaction support provides a container-managed, implicit local transaction boundary within which application updates can be committed and their connections cleaned up by the container. Applications can then be designed with a greater degree of independence from deployment concerns. This makes using a **Transaction** attribute of `Supports` much simpler, for example, when the business logic may be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage regardless of whether or not the application runs under a transaction. The application can depend on the close behaving in the same way and not causing a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios running business logic under a **Transaction** policy of `Not supported` performs better than if it had been run under a `Required` policy. This benefit is exploited through the **Local Transactions - Resolution-control** extended deployment setting of `ContainerAtBoundary`. With this setting, application interactions with resource providers (such as databases) are managed within implicit RMLTs that are both started and ended by the container. The RMLTs are committed by the container at the configured **Local Transactions - Boundary**; for example at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

Use local transactions in a managed environment that guarantees clean-up.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default **Local Transactions - Resolution-control** extended deployment setting of `Application`. In this case, the container ensures connection cleanup at the boundary of the local transaction context.

J2EE specifications that describe application use of local transactions do so in the manner provided by the default setting of **Local Transactions - Resolution-control**=`Application` and **Local Transactions - Unresolved-action**=`Rollback`. By configuring the **Local Transactions - Unresolved-action** extended deployment setting to `Commit`, then any RMLTs started by the application but not completed when the local transaction containment ends (for example, when the method ends) are committed by the container. This usage applies to both servlets and enterprise beans.

Extend the duration of a local transaction beyond the duration of an EJB component method.

The J2EE specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. In IBM WebSphere Application Server Enterprise, you can exploit the **Local Transactions - Boundary** extended deployment setting to give the following advantages:

- Significantly extend the use-cases of RMLTs
- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

An ActivitySession is a IBM WebSphere Application Server Enterprise programming model extension that provides a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which can be controlled by a client. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is exploited through the **Local Transactions - Boundary** extended deployment setting of `ActivitySession`. Such extended RMLTs can remain under the control of the application or be managed by the container depending on the use of the **Local Transactions - Resolution-control** deployment descriptor setting.

Coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can exploit ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans under that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

To determine how best to configure the transaction support for an application, depending on what you want to do with transactions, consider the following points.

General points

- You want to start and end global transactions explicitly in the application (BMT session beans and servlets only).

For a session bean, set the **Transaction type** to `Bean` (to use bean-managed transactions) in the component's deployment descriptor. (You do not need to do this for servlets.)

- You want to access only one XA or non-XA resource in a method.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Supports`.
- You want to access several XA resources atomically across one or more bean methods.

  In the Container transaction deployment descriptor, set **Transaction** to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resource in a method without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.
- You want to access several non-XA resource in a method and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application` and set **Local Transactions - Unresolved-action** to **Rollback**. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.

Points specific to IBM WebSphere Application Server Enterprise

- You want to access one of more non-XA resources across multiple EJB method calls without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container Transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to use a single non-XA resource and one or more XAResources.

  Use the Last Participant Support of IBM WebSphere Application Server Enterprise.

Using bean-managed transactions

This topic describes how to enable a session bean or servlet to use bean-managed transactions, to manage its own transactions directly instead of letting the container manage the transactions.

Note: Entity beans cannot manage transactions (so cannot use bean-managed transactions).

To enable a session bean or servlet to use bean-managed transactions, complete the following steps:

Steps for this task

1. Set the **Transaction type** attribute in the component's deployment descriptor to `Bean`, as described in ″Setting transactional attributes in the deployment descriptor″ (not in this document).
2. Write the component code to actively manage transactions

   When writing the code required by a component to manage its own transactions, remember the following basic rules:

   - An instance of a stateless session bean cannot reuse the same transaction context across multiple methods called by an EJB client.
   - An instance of a stateful session bean can reuse the same transaction context across multiple methods called by an EJB client.

   The following code extract shows the standard code required to obtain an object encapsulating the transaction context. There are three basics steps involved:

   - The component class must set the value of the javax.ejb.SessionContext object reference in the setSessionContext method.
   - A javax.transaction.UserTransaction object is created by calling a lookup on ″java:comp/UserTransaction″.
   - The UserTransaction object is used to participate in the transaction by calling transaction methods such as begin and commit as needed. If an enterprise bean begins a transaction, it must also complete that transaction either by invoking the commit method or the rollback method.

```
...
import javax.transaction.*;
...
public class MyStatelessSessionBean implements SessionBean {
private SessionContext mySessionCtx =null;
...
public void setSessionContext (SessionContext ctx)throws EJBException {
mySessionCtx =ctx;
}
...
    public float doSomething(long arg1)throws FinderException,EJBException {
        UserTransaction userTran = (UserTransaction)initCtx.lookup(
           "java:comp/UserTransaction");
        ...
        //User userTran object to call transaction methods
        userTran.begin ();
        //Do transactional work
        ...
        userTran.commit ();
        ...
    }
    ...
}
```

Using local transactions

Local transaction containment (LTC) support, and its configuration through local transaction extended deployment descriptors, gives IBM WebSphere Application Server application programmers a number of advantages. This topic describes those advantages and how they relate to the settings of the local transaction extended deployment descriptors. This topic also describes points to consider to help you best configure transaction support for some example scenarios that use local transactions.

Develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not need to use global transactions, it is often more efficient to deploy the bean with the Container Transaction deployment descriptor **Transaction** attribute set to `Not supported` instead of `Required`.

With the extended local transaction support of IBM WebSphere Application Server, applications can perform the same business logic in an unspecific transaction context as they can under a global transaction. An enterprise bean, for example, runs under an unspecified transaction context if it is deployed with a **Transaction** attribute of `Not supported` or `Never`.

The extended local transaction support provides a container-managed, implicit local transaction boundary within which application updates can be committed and their connections cleaned up by the container. Applications can then be designed with a greater degree of independence from deployment concerns. This makes using a **Transaction** attribute of `Supports` much simpler, for example, when the business logic may be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage regardless of whether or not the application runs under a transaction. The application can depend on the close behaving in the same way and not causing a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios running business logic under a **Transaction** policy of `Not supported` performs better than if it had been run under a `Required` policy. This benefit is exploited through the **Local Transactions - Resolution-control** extended deployment setting of `ContainerAtBoundary`. With this setting, application interactions with resource providers (such as databases) are managed within implicit RMLTs that are both started and ended by the container. The RMLTs are committed by the container at the configured **Local Transactions - Boundary**; for example at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

Use local transactions in a managed environment that guarantees clean-up.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default **Local Transactions - Resolution-control** extended deployment setting of `Application`. In this case, the container ensures connection cleanup at the boundary of the local transaction context.

J2EE specifications that describe application use of local transactions do so in the manner provided by the default setting of **Local Transactions - Resolution-control**=`Application` and **Local Transactions - Unresolved-action**=`Rollback`. By configuring the **Local Transactions - Unresolved-action** extended deployment setting to `Commit`, then any RMLTs started by the application but not completed when the local transaction containment ends (for example, when the method ends) are committed by the container. This usage applies to both servlets and enterprise beans.

Extend the duration of a local transaction beyond the duration of an EJB component method.
The J2EE specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. In IBM WebSphere Application Server Enterprise, you can exploit the **Local Transactions - Boundary** extended deployment setting to give the following advantages:

- Significantly extend the use-cases of RMLTs
- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

An ActivitySession is an IBM WebSphere Application Server Enterprise programming model extension that provides a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which can be controlled by a client. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is exploited through the **Local Transactions - Boundary** extended deployment setting of `ActivitySession`. Such extended RMLTs can remain under the control of the application or be managed by the container depending on the use of the **Local Transactions - Resolution-control** deployment descriptor setting.

Coordinate multiple one-phase resource managers.
For resource managers that do not support XA transaction coordination, a client can exploit ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans under that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

To determine how best to configure the transaction support for an application, depending on what you want to do with transactions, consider the following points.

General points

- You want to start and end global transactions explicitly in the application (BMT session beans and servlets only).

  For a session bean, set the **Transaction type** to `Bean` (to use bean-managed transactions) in the component's deployment descriptor. (You do not need to do this for servlets.)
- You want to access only one XA or non-XA resource in a method.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Supports`.
- You want to access several XA resources atomically across one or more bean methods.

In the Container transaction deployment descriptor, set **Transaction** to `Required`, `Requires new`, or `Mandatory`.

- You want to access several non-XA resource in a method without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.
- You want to access several non-XA resource in a method and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application` and set **Local Transactions - Unresolved-action** to **Rollback**. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.

Points specific to IBM WebSphere Application Server Enterprise

- You want to access one of more non-XA resources across multiple EJB method calls without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container Transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to use a single non-XA resource and one or more XAResources.

  Use the Last Participant Support of IBM WebSphere Application Server Enterprise.

Configuring transaction properties for an application server

Use this task to configure the transaction properties for an application server; for example, to define the location of the directory that contains the transaction log or to change default timeouts associated with transactions.

To configure the transaction properties for an application server, complete the following steps:

Steps for this task

1. Start the Administrative console
2. In the navigation pane, select **Servers-> Manage Application Servers->** ***your_app_server***

   This displays the properties of the application server, *your_app_server*, in the content pane.

3. Select the Transaction Service tab, to display the properties page for the transaction service, as two notebook pages:

   Configuration
      The values of properties defined in the configuration file. If you change these properties, the new values are applied when the application server next starts.

   Runtime
      The runtime values of properties. If you change these properties, the new values are applied immediately, but are overwritten with the Configuration values when the application server next starts.
4. Select the Configuration tab, to display the transaction-related configuration properties.
5. **(Optional)** If you want to change the directory in which transaction logs are written, type the full pathname of the directory in the **Transaction log directory** field.

   You can check the current runtime value of **Transaction log directory**, by clicking the Runtime tab.

   You can also specify a size for the transaction logs, as described in the following step.

   Note: If you change the transaction log directory, you should apply the change and restart the application server as soon as possible, to minimize the risk of problems caused that might occur before the application server is restarted. For example, if a problem causes the server to fail (with in-flight transactions), the server next starts with the new log directory and is unable to automatically resolve in-flight transactions that were recorded in the old log directory.
6. **(Optional)** If you want to change the default file size of transaction log files, modify the **Transaction log directory** field to include a file size setting, in the following format:

   ```
   directory_name;file_size
   ```

   Where
   - *directory_name* is the name of the transaction log directory
   - *file_size* is the new default size specified in bytes. The *n*K or *n*M suffix can be used to indicate kilobytes or megabytes. If you do not specify a file size value, the default value of 1M is used.

   For example, `c:\tranlogs;2M` indicates the files are to be created with 2M bytes size and stored in the directory c:\tranlogs.

   In a non-production environment, you can use the transaction log directory value of `;0` to disable transaction logging. (There must be no directory name element before the size element of 0.) You should not disable transaction logging in a production environment, because this prevents recovery after a system failure and, therefore, data integrity cannot be guaranteed.
7. In the **Total transaction lifetime timeout** field, type the number of milliseconds a transaction can remain inactive before it is ended by the transaction service. A value of 0 (zero) indicates that there is no timeout limit.
8. In the **Client inactivity timeout** field, type the number of seconds after which a client is considered inactive and the transaction service ends any transactions associated with that client. A value of 0 (zero) indicates that there is no timeout limit.

9. Click **OK**.
10. Stop then restart the application server.

    If you change the transaction log directory configuration property to an incorrect directory name, the application server will restart but be unable to open the transaction logs. You should change the configuration property to a valid directory name, then restart the application server.

Transaction service settings

Use this page to modify transaction service settings.

To view this administrative console page, click **Servers** > **Application Servers** > *server* > **Transaction Service**.

Transaction log directory

Specifies the name of a directory for this server where the transaction service stores log files for recovery.

A blank value in the server configuration is expanded by the transaction log at startup as the directory (install_root)/tranlog/(*server_name*).

When the application running on the WebSphere product accesses more then one resource, the WebSphere product stores transaction information to properly coordinate and manage the distributed transaction. In a higher transaction load, this persistence slows down performance of the application server due to its dependency on the operating system and the underlying storage systems.

To achieve better performance, move the transaction log files to a storage device with more physical disk drives, or preferably RAID disk drives. When the log files are moved to the file systems on the raided disks, the task of writing data to the physical media is shared across the multiple disk drives. This allows more concurrent access to persist transaction information and faster access to that data from the logs. Depending upon the design of the application and storage subsystem, performance gains can range from 10% to 100%, or even more in some cases.

This change is applicable only to the configuration where the application uses distributed resources or XA transactions, for example, multiple databases and resources are accessed within a single transaction. Consider setting this property when the application server shows one or more of following signs:

- CPU utilization remains low despite an increase in transactions
- Transactions fail with several time outs
- Transaction rollbacks occur with ″unable to enlist transaction″ exception
- Application server hangs in middle of a run and requires the server to be restarted
- The disk on which an application server is running shows higher utilization

Data type
String

Default
Initial value is the *%WAS_HOME%*\tranlog directory and a default size of 1MB.

Recommended
Create a file system with at least 3-4 disk drives raided together in a

RAID-0 configuration. Then, create the transaction log on this file system with the default size. When the server is running under load, check the disk input and output. If disk input and output time is more then 5%, consider adding more physical disks to lower the value. If disk input and output is low, but the server is still high, consider increasing the size of the log files.

Total transaction lifetime timeout

Specifies the maximum duration, in seconds, for transactions on this application server.

Any transaction that is not requested to complete before this timeout is rolled back. If set to 0, there is no timeout limit.

Data type	Integer
Units	Seconds
Default	120
Range	0 to 2 147 483 647

Client inactivity timeout

Specifies the maximum duration, in seconds, between transactional requests from a remote client.

Any period of client inactivity that exceeds this timeout results in the transaction rolling back in this application server. If set to 0, there is no timeout limit.

Data type	Integer
Units	Seconds
Default	60
Range	0 to 2 147 483 647

Maximum Transaction Timeout

Specifies the maximum duration, in seconds, that transactions started by or propagated into this application server are allowed to execute.

Data type	Integer
Units	Seconds
Default	300
Range	0 to 2 147 483 647

Using local transactions

Local transaction containment (LTC) support, and its configuration through local transaction extended deployment descriptors, gives IBM WebSphere Application Server application programmers a number of advantages. This topic describes those advantages and how they relate to the settings of the local transaction extended deployment descriptors. This topic also describes points to consider to help you best configure transaction support for some example scenarios that use local transactions.

Develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not need to use global transactions, it is often more efficient to deploy the bean with the Container Transaction deployment descriptor **Transaction** attribute set to `Not supported` instead of `Required`.

With the extended local transaction support of IBM WebSphere Application Server, applications can perform the same business logic in an unspecific transaction context as they can under a global transaction. An enterprise bean, for example, runs under an unspecified transaction context if it is deployed with a **Transaction** attribute of `Not supported` or `Never`.

The extended local transaction support provides a container-managed, implicit local transaction boundary within which application updates can be committed and their connections cleaned up by the container. Applications can then be designed with a greater degree of independence from deployment concerns. This makes using a **Transaction** attribute of `Supports` much simpler, for example, when the business logic may be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage regardless of whether or not the application runs under a transaction. The application can depend on the close behaving in the same way and not causing a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios running business logic under a **Transaction** policy of `Not supported` performs better than if it had been run under a `Required` policy. This benefit is exploited through the **Local Transactions - Resolution-control** extended deployment setting of `ContainerAtBoundary`. With this setting, application interactions with resource providers (such as databases) are managed within implicit RMLTs that are both started and ended by the container. The RMLTs are committed by the container at the configured **Local Transactions - Boundary**; for example at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

Use local transactions in a managed environment that guarantees clean-up.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default **Local Transactions - Resolution-control** extended deployment setting of `Application`. In this case, the container ensures connection cleanup at the boundary of the local transaction context.

J2EE specifications that describe application use of local transactions do so in the manner provided by the default setting of **Local Transactions - Resolution-control**=`Application` and **Local Transactions - Unresolved-action**=`Rollback`. By configuring the **Local Transactions - Unresolved-action** extended deployment setting to `Commit`, then any RMLTs started by the application but not completed when the local transaction containment ends (for example, when the method ends) are committed by the container. This usage applies to both servlets and enterprise beans.

Extend the duration of a local transaction beyond the duration of an EJB component method.

The J2EE specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. In IBM WebSphere Application Server Enterprise, you can exploit the **Local Transactions - Boundary** extended deployment setting to give the following advantages:

- Significantly extend the use-cases of RMLTs
- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

An ActivitySession is an IBM WebSphere Application Server Enterprise programming model extension that provides a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which can be controlled by a client. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is exploited through the **Local Transactions - Boundary** extended deployment setting of `ActivitySession`. Such extended RMLTs can remain under the control of the application or be managed by the container depending on the use of the **Local Transactions - Resolution-control** deployment descriptor setting.

Coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can exploit ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans under that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

To determine how best to configure the transaction support for an application, depending on what you want to do with transactions, consider the following points.

General points

- You want to start and end global transactions explicitly in the application (BMT session beans and servlets only).

  For a session bean, set the **Transaction type** to `Bean` (to use bean-managed transactions) in the component's deployment descriptor. (You do not need to do this for servlets.)
- You want to access only one XA or non-XA resource in a method.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Supports`.
- You want to access several XA resources atomically across one or more bean methods.

In the Container transaction deployment descriptor, set **Transaction** to `Required`, `Requires new`, or `Mandatory`.

- You want to access several non-XA resource in a method without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.
- You want to access several non-XA resource in a method and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application` and set **Local Transactions - Unresolved-action** to **Rollback**. In the Container transaction deployment descriptor, set **Transaction** to `Not supported`.

Points specific to IBM WebSphere Application Server Enterprise

- You want to access one of more non-XA resources across multiple EJB method calls without having to worry about managing your own local transactions.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `ContainerAtBoundary`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the component's deployment descriptor, set **Local Transactions - Resolution-control** to `Application`, **Local Transactions - Boundary** to `ActivitySession`, and **Bean Cache - Activate at** to `ActivitySession`. In the Container Transaction deployment descriptor, set **Transaction** to `Not supported` and set **ActivitySession** attribute to `Required`, `Requires new`, or `Mandatory`.
- You want to use a single non-XA resource and one or more XAResources.

  Use the Last Participant Support of IBM WebSphere Application Server Enterprise.

Managing active transactions

Use this task to manage transactions that are active on an application server.

You can use this task to display a snapshot of all the transactions currently running on an application server. For each transaction, the following properties are shown: its local ID, global ID, and current status. You can also choose to finish transactions manually.

Under normal circumstances, transactions should run and complete (commit or rollback) automatically, without the need for intervention. However, in some circumstances, you may need to finish a transaction manually. For example, you may want to finish a transaction that has become stuck polling a resource manager that you know will not become available again within the desired timeframe.

Note: If you choose to finish a transaction on an application server, it is recorded as having completed in the transaction service logs for that server, so will not be eligible for recovery during server start up. If you finish a transaction, you are responsible for cleaning up any in-doubt transactions on the resource managers affected.

To manage the active transactions for an application server, use the administrative console to complete the following steps:

Steps for this task

1. In the navigation pane, select **Servers-> Manage Application Servers**

   This displays a list of application servers in the content pane.
2. In the content pane, click *your_app_server*

   This displays the properties of the application server, *your_app_server*.
3. In the content pane, click the **Runtime** tab.

   This displays the runtime properties of the application server.
4. In the Additional Properties table, select **Transaction Service**

   This displays the runtime properties of the Transaction Service.
5. Click **Manage Transactions**.

   This displays a snapshot of all the transactions currently running on the server. For each transaction, the following properties are shown: its local ID, global ID, and current status.
6. **(Optional)** If you want to finish one or more transactions, select the checkbox provided on the entry for the transaction, then click **Finish**. Alternatively, to finish all transactions, select the checkbox in the header of the transactions table, then click **Finish**.

Managing transaction logging for optimum server availability

This topic describes some considerations and actions that you can use to manage transaction logging to help ensure that the availability of your application servers is optimized.

The transaction service writes information to the transaction log for every global transaction which involves two or more resources or is distributed across multiple servers. The transaction log is stored on disk and is used by the transaction service for recovery after a system or server crash. The transaction log for each application server consists of multiple files held in a single directory. You can change the directory that an application server uses to store the transaction log, as described in “Configuring transaction properties for an application server” on page 293.

When a global transaction is completed, the information in the transaction log is not needed anymore so is marked for deletion. Periodically, this redundant information is garbage collected and the space reused by new transactions. The log files are created of fixed size at server startup, thus no further disk space allocation is required during the lifetime of the server. The default allocation is suitable for around 500 concurrent transactions.

If all the log space is in use when a transaction needs to save information, the transaction is rolled back and the message ″WTRN0075W: The transaction log file is full. Transaction rolled back.″ is reported to the system error log. No more transactions can commit until more log space is made available when existing active transactions complete.

You can monitor the number of concurrent global transactions by using the performance monitoring counters for transactions. The ″Global transaction commit time″ counter is a measure of how long a transaction takes to complete and, therefore, how long the log is in use by a transaction. If this value is high, then transactions are taking a long time to complete, which can be due to resource manager or network failures. If you ensure this value is low, the log is more efficiently used and unlikely to become full.

You can change the default size of log files by updating the transaction log settings as described in “Configuring transaction properties for an application server” on page 293. Take care if you increase the size above the default 1Mbyte setting, because this extends the time of the log file garbage collection process, and can lead to undesireable periodic ″transaction stall″ effects.

Configuring transaction aspects of servers for optimum availability

This topic describes some considerations and actions that you can take to configure transaction-related aspects of application servers for optimum availability.

To configure transaction-related aspects of application servers for optimum availability, complete the following steps:

Steps for this task

1. Store the transaction log files on a fast disk in a highly-available file system, such as a RAID device.

   The transaction log may need to be accessed by every global transaction and be used for transaction recovery after a crash. Therefore, the disk the log files are being written to should be on a highly-available file system, such as a RAID device.

   The performance of the disk also directly affects the transaction performance. In general, a global transaction makes two disk writes, one after the prepare phase when the outcome of the transaction is known (this information is forced to disk) and a further disk write at transaction completion. Therefore, the transaction logs should be placed on the fastest disks available and not make use of network mounted devices.
2. Mirror the transaction log files by using hardware disk mirroring or dual-ported disks.

   If log files have been mirrored or can be recovered, they can be used when restarting a failed server or moved to an another machine and another server started there to perform recovery.

   Hardware disk mirroring or dual-ported disks can be used by specifiying the appropriate file system directory for the transaction logs using the WebSphere Administrative Console.
3. Specify the optimum location of the transaction log directory for application servers.

   The default transaction log directory for an application server configuration is unset. By default the application server places transaction log files in a subdirectory of the installed WebSphere tranlog directory (as defined by the WebSphere variable TRANLOG_ROOT), and the subdirectory name is the same as the server name. For example, the default directory for a server named `server1` on Windows 2000 is: `c:\WebSphere\AppServer\tranlog\server1`.

   You can specify an optimum location of the transaction log directory for all application servers, either on a node or cell scope, by setting the WebSphere variable TRANLOG_ROOT. You can specify a different location separately for each application server by setting the **Transaction Log Directory** property for the server.

4. Never allow more than one application server to concurrently use the same set of log files.

   Because the transaction logs record the state of global transactions within a server, if the logs become lost or corrupt, then transactions that are in the prepared state before failure can leave resources in an in-doubt state and prevent further updates or access to the resources by other users or servers. These transactions may need to be manually resolved by either committing or rolling back the transactions at the affected resource managers. The failed server can then be cold-started, which creates new empty transaction logs.

   If log files have been mirrored or can be recovered, they can be used when restarting the failed server or moved to an alternate server or machine and another server restarted to perform recovery, as described in the related tasks.

   Never allow more than one application server to concurrently use the same set of log files, because each server will destroy the information recorded by the other, resulting in corrupt log files that are unusable for future recovery purposes.

5. Configure application servers to always use the same listening port address at each startup.

   If you are running distributed transactions between multiple application servers, the remote object references saved in the transaction log need to be redirected to the originating server on recovery.

   On IBM WebSphere Application Server Network Deployment, the node agents automatically redirect such remote object references to the appropriate application servers on recovery. However, if the distributed transaction is between application servers that are not on IBM WebSphere Application Server Network Deployment, then you must handle the redirection of remote object references for transaction recovery to complete. For example, you must do this is if an application server is deployed on IBM WebSphere Application Server (not the Network Deployment edition) and runs distributed transactions with non-WebSphere EJB or Corba servers.

   In particular, the default restart action of an application server not on IBM WebSphere Application Server Network Deployment is to use a different listening port address to the port when the server shut down. This prevents transaction recovery completing. To overcome this, you should always configure application servers to always use the same listening port address at each startup (see the ORB property com.ibm.CORBA.ListenerPort in ″Object Request Broker service settings that can be added to the administrative console″″ (not in this document- See the InfoCenter)). You may need to make similar configuration changes to other application servers involved in transactions, to be able to access those servers during recovery.

Moving a transaction log from one server to another

This topic describes some considerations and actions that you can take to move the transaction logs for an application server to another server.

To move transaction logs from one application server to another, consider the following steps:

Steps for this task

1. Move all the transaction log files for the application server.

The transaction log directory for each server contains four log files; named tranlog1, tranlog2, XAresource1, and XAresource2. When moving transaction logs from one server to another you must move all four files; otherwise recovery may not complete resulting in data inconsistency.

2. For a single server configuration, move the transaction logs to any server that has access to the same resource managers.

   For a single server configuration (where there are no distributed transactions), the transaction logs can be moved to any server (on any node) that has access to the same resource managers as the original server. For example, the server needs communication and valid security access to databases or message queues.

   All the transaction log files for the original server need to be moved to a directory accessible by the new server. This can be accomplished by either renaming the transaction log directory or copying all the log files to the new server's transaction log directory before starting the new server.

   Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

3. For a network-deployed server configuration, move the transaction logs to a server that has the same name and host IP address, and access to the same resource managers.

   For a network-deployed server configuration, where there can be distributed transactions present in the logs, there are more restrictions. Distributed transactions that access multiple servers log information about each server involved in the transaction. This information includes the server name and the IP address of the machine on which the server is running. When recovery is taking place on server restart, the server uses this information to contact the distributed servers and similarly, the distributed servers try to contact the server with the same original name. So, if a server fails and the logs need to the recovered on an alternative server, that alternative server needs to have the same name and host IP address as the original server. The alternative server also needs to have access to the same resource managers as the original server. For example, the server needs communication and valid security access to databases or message queues.

   Notes:

   a. All servers within a cell must have unique names.
   b. To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

Restarting an application server on a different host

This topic describes some considerations and actions that you can take with transaction logs to restart an application server on a different host.

Moving transactions logs to a different host is similar to moving logs from one server to another, as described in "Moving a transaction log from one server to another" on page 302"Moving a transaction log from one server to another".

This involves moving an original application server on one host to an alternative server, which has access to the same resource managers, on another host. For a network-deployed server configuration, the alternative server must have the same name and host IP address as the original server.

Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

To restart an application server on a different host, complete the following steps:

Steps for this task

1. Ensure that the alternative application server is stopped.
2. Move all the transaction logs for the original server to the alternative application server, according to the considerations described in "Moving a transaction log from one server to another" on page 302.
3. Restart the alternative application server.

Transactional interoperation with non-WebSphere application servers

To interoperate transactionally with a non-WebSphere application server, IBM WebSphere Application Server switches dynamically between native transaction contexts and interoperable OTS contexts depending on the capability of the partner with which it is interoperating. The following system properties (that were needed to be set in IBM WebSphere Application Server before version 5.0 to enable transactional interoperation), and the use of native contexts, are deprecated:

```
com.ibm.ejs.jts.jts.ControlSet.nativeOnly=false
com.ibm.ejs.jts.jts.ControlSet.interoperabilityOnly=true
```

In a future release of IBM WebSphere Application Server only interoperable OTS contexts will be supported.

Troubleshooting transactions

Use this overview task to help resolve a problem that you think is related to the Transaction service.

To identify and resolve transaction-related problems, you can use the standard IBM WebSphere Application Server RAS facilities. If you encounter a problem that you think might be related to transactions, complete the following stages:

Steps for this task

1. Check for transaction messages in the admin console.

   The Transaction service produces diagnostic messages prefixed by ″WTRN″. The error message indicates the nature of the problem and provides some detail. The associated message information provides an explanation and any user actions to resolve the problem.
2. Check for Transaction messages in the activity log.

   Activity log messages produced by the Transaction service are accompanied by log analyzer descriptions.
3. Check for more messages in the application server's stdout.log.

For more information about a problem, check the stdout.log file for the application server, which should contain more error messages and extra details about the problem.

4. Check for messages in the application server's transaction log directory for information about the transactions in-flight when the problem occurred.

   Note: If you changed the transaction log directory and a problem caused the application server to fail (with in-flight transactions) before the server was restarted properly, the server will next start with the new log directory and be unable to automatically resolve in-flight transactions that were recorded in the old log directory. To resolve this, you can copy the transaction logs to the new directory then stop and restart the application server.

Transaction service exceptions

This topic lists the exceptions that can be thrown by the IBM WebSphere Application Server transaction service. The exceptions are listed in the following groups:

- Standard exceptions
- Heuristic exceptions

If the EJB container catches a system exception from the business method of an enterprise bean, and the method is running within a container-managed transaction, the container rolls back the transaction before passing the exception on to the client. For more information about how the container handles the exceptions thrown by the business methods for beans with container-managed transaction demarcation, see the section *Exception handling* in the Enterprise JavaBeans 2.0 specification. That section specifies the container's action as a function of the condition under which the business method executes and the exception thrown by the business method. It also illustrates the exception that the client receives and how the client can recover from the exception.

Standard exceptions

The standard exceptions such as TransactionRequiredException, TransactionRolledbackException, and InvalidTransactionException are defined in the Java Transaction API (JTA) 1.0.1 Specification.

InvalidTransactionException
This exception indicates that the request carried an invalid transaction context.

TransactionRequiredException exception
This exception indicates that a request carried a null transaction context, but the target object requires an active transaction.

TransactionRolledbackException exception
This exception indicates that the transaction associated with processing of the request has been rolled back, or marked for roll back. Thus the requested operation either could not be performed or was not performed because further computation on behalf of the transaction would be fruitless.

Heuristic exceptions

A heuristic decision is a unilateral decision made by one or more participants in a transaction to commit or rollback updates without first obtaining the consensus outcome determined by the Transaction Service. Heuristic decisions are an issue only after the participant has been prepared and the second phase of commit processing is underway. Heuristic decisions are normally made only in unusual circumstances, such as repeated failures by the transaction manager to communicate with a resource manage during two-phase commit. If a heuristic decision is taken, there is a risk that the decision differs from the consensus outcome, resulting in a loss of data integrity.

The following list provides a summary of the heuristic exceptions. For more detail, see the Java Transaction API (JTA) 1.0.1 Specification.

HeuristicRollback exception
This exception is raised on the commit operation to report that a heuristic decision was made and that all relevant updates have been rolled back.

HeuristicMixed exception
This exception is raised on the commit operation to report that a heuristic decision was made and that some relevant updates have been committed and others have been rolled back.

UserTransaction interface - methods available

For details about the methods available with the UserTransaction interface, see the IBM WebSphere Application Server Release 5 API Specification (found in the Javadoc section of the InfoCenter) or the Java Transaction API (JTA) 1.0.1 Specification.

Coordinating access to 1-PC and 2-PC-capable resources within the same transaction

Use these topics to help you coordinate the use of a single 1-phase commit (1PC) capable resource with any number of 2-phase commit (2PC) capable resources in the same global transaction.

You can coordinate the use of a single 1-phase commit (1PC) capable resource with any number of 2-phase commit (2PC) capable resources in the same global transaction.

At transaction commit, the 2-phase commit resources are prepared first using the 2-phase commit protocol, and if this is successful the 1-phase commit-resource is then called to commit(one_phase). The 2-phase commit resources are then committed or rolled back depending on the response of the 1-phase commit resource.

For more information about coordinating access to 1PC and 2PC-capable resources within the same transaction, see the following topics:

- "Coordinating access to 1-PC and 2-PC-capable resources within the same transaction" on page 307
- "Enabling an application to coordinate access to 1-PC and 2-PC-capable resources within the same transaction" on page 307
- "Configuring an application server to allow logging for heuristic reporting" on page 308

Coordinating access to 1-PC and 2-PC-capable resources within the same transaction

You can coordinate the use of a single 1-phase commit (1PC) capable resource with any number of 2-phase commit (2PC) capable resources in the same global transaction.

At transaction commit, the 2-phase commit resources are prepared first using the 2-phase commit protocol, and if this is successful the 1-phase commit-resource is then called to commit(one_phase). The 2-phase commit resources are then committed or rolled back depending on the response of the 1-phase commit resource.

Note: If the global transaction is distributed across multiple application servers, you cannot coordinate access to 1-PC and 2-PC-capable resources within the same transaction.

Coordinating access to 1-PC and 2-PC-capable resources within the same transaction introduces an increased risk of an heuristic outcome to the transaction. That is, the transaction manager cannot be sure that all resources were completed in the same direction (either committed or rolled back). For this reason, to enable an application to coordinate access to 1-PC and 2-PC-capable resources within the same transaction, you configure the application to accept the increased risk of an heuristic outcome.

An heuristic outcome occurs if the transaction service (JTS) receives no response from the commit one-phase flow on the 1PC resource. In this situation the transaction service cannot determine whether changes for the 1PC resource were committed or rolled back, so cannot drive reliably the correct outcome of the global transaction on the other 2PC resources.

You can configure the transaction service for an application server to indicate whether or not to log that it is about to commit the 1PC resource. This does not reduce the heuristic hazard, but ensures that any failure, and subsequent recovery, of the application server during the 1PC phase occurs with knowledge of whether or not the 1PC resource was asked to commit:

- If the 1PC resource was asked to commit, a heuristic outcome is reported to the activity log.
- If the 1PC resource was not asked to commit, then the transaction is rolled back consistently.

Enabling an application to coordinate access to 1-PC and 2-PC-capable resources within the same transaction

Use this task to enable an application to coordinate access to 1-phase and 2-phase commit capable resources within the same transaction.

To enable an application to coordinate access to 1-phase and 2-phase commit capable resources within the same transaction, you must configure the application to accept the increased risk of an heuristic outcome.

To configure an application to indicate that you accept the increased risk of an heuristic outcome, use the Application Assembly tool to complete the following steps:

Steps for this task

1. Launch the Application Assembly Tool.
2. Open the application EAR file.
3. In the navigation pane, select the application

   This displays the properties notebook in the property pane.
4. In the property pane, select the WAS Enterprise tab.
5. Select the **Accept heuristic hazard** checkbox.
6. To apply the changes and close the Application Assembly Tool, click **OK**. Otherwise, to apply the values but keep the property dialog open for additional edits, click **Apply**.
7. **(Optional)** To see changes reflected in your application, regenerate deployment code and re-install the deployable archive.

Last participant support extension settings

Use this page to configure last participant support extensions.

Last participant support is an extension to the transaction service to allow a single one-phase resource to participate in a two-phase transaction with one or more two-phase resources.

To view this administrative console page, click **Applications > Applications >** *application_name* **> Last Participant Support Extension**.

Accept Heuristic Hazard: Specifies whether the application accepts the possibility of an heuristic hazard occurring in a two-phase transaction containing a one-phase resource.

Default
: Cleared

Range

: **Selected**
  : The application accepts the increased risk of an heuristic outcome.

  Cleared
  : The application does not accept the increased risk of an heuristic outcome.

Configuring an application server to allow logging for heuristic reporting

To enable an application server to log ″about to commit 1PC resource″ events from transactions that involve a 1-phase commit resource and 2-phase commit resources, use the Administrative console to complete the following steps:

Steps for this task

1. Start the Administrative console
2. In the navigation pane, select **Servers-> Manage Application Servers->** ***your_app_server***

   This displays the properties of the application server, *your_app_server*, in the content pane.
3. Select the Transaction Service tab, to display the properties page for the transaction service, as two notebook pages:

Configuration
The values of properties defined in the configuration file. If you change these properties, the new values are applied when the application server next starts.

Runtime
The runtime values of properties. If you change these properties, the new values are applied immediately, but are overwritten with the Configuration values when the application server next starts.

4. Select the Configuration tab, to display the transaction-related configuration properties.
5. Select the **Enable logging for heuristic reporting** checkbox.
6. Click **OK**.
7. Stop then restart the application server.

Exceptions thrown for transactions involving both single- and two-phase commit resources

The exceptions that can be thrown by transactions that involve single- and two-phase commit resources are the same as those that can be thrown by transactions involving only two-phase commit resources, and are listed in the WebSphere API reference information (Javadoc).

Last Participant Support: Resources for learning

Use the following links to find relevant supplemental information about Last Participant Support. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- “Programming specifications”
- “Other”

Programming specifications
- J2EE Activity Service for Extended Transactions

  http://www.jcp.org/en/jsr/detail?id=95
- Java Transaction API (JTA) 1.0.1

  http://java.sun.com/products/jta/

Other
- WebSphere Application Server Enterprise Version 5 Overview: Advanced Transactional Connectivity

  http://www-3.ibm.com/software/info1/websphere/index.jsp?tab=products/appserv_enterprise#advanced
- Listing of PDF files to learn about WebSphere Application Server Version 5

  http://www-3.ibm.com/software/webservers/appserv/was/
- Listing of all IBM WebSphere Application Server Redbooks

http://publib-b.boulder.ibm.com/Redbooks.nsf/Portals/WebSphere

- Listing of all IBM WebSphere Application Server Whitepapers
  http://www-4.ibm.com/software/webservers/appserv/whitepapers.html
- WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide
  http://www.redbooks.ibm.com/redbooks/SG246504.html

Chapter 13. Using the ActivitySession service

These topics provide information about implementing WebSphere enterprise applications that use ActivitySessions.

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. ActivitySessions provide a scoping mechanism for units of work, and both an ActivitySession and a transaction has the same following characteristics:

- It can be bean-managed or container-managed
- It can be distributed across application servers
- It can be used as the context for managing EJB activation policy and lifecycle

An ActivitySession differs significantly from a transaction in the manner of its interaction with resource managers. An ActivitySession is used to scope or coordinate local transactions. That is, an ActivitySession can be used to request multiple 1-phase resource managers to come to an application- or container-determined outcome. Unlike a transaction, an ActivitySession has no notion of a prepare phase or any notion of recovery at a service level.

The WebSphere EJB container and deployment tools support ActivitySessions as an extension to the J2EE programming model. Enterprise beans can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An enterprise bean with an ActivitySession-scoped lifecycle can participate in a resource manager local transaction (RMLT) that has a duration of the ActivitySession rather than an individual method on the bean (which is all that is possible under the standard J2EE model). Applications can then be composed of several enterprise beans with ActivitySession-based activation, with each bean participating in extended local transactions with one or more resource managers. At the end of the ActivitySession each of the local transactions can be directed to a common outcome by the ActivitySession manager.

You can configure the WebSphere containers and deployable applications to support enterprise beans that operate under application- or container-initiated ActivitySessions rather than, or in addition to, transactions.

For more information about implementing WebSphere enterprise applications that use ActivitySessions, see the following topics:

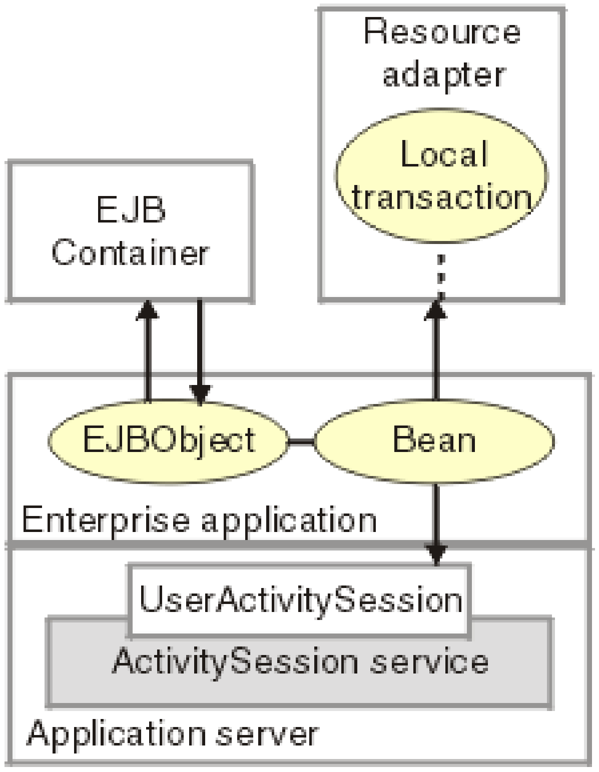
- "The ActivitySession service" on page 312
  - "ActivitySession and transaction contexts" on page 315
  - "Using ActivitySessions with HTTP sessions" on page 313
- "The ActivitySession service application programming interfaces" on page 334
- "Developing a J2EE application to use ActivitySessions" on page 322
- "Samples: ActivitySessions" on page 335
- "Configuring ActivitySession deployment attributes for an enterprise bean" on page 325
- "Configuring ActivitySession deployment attributes for a Web application" on page 329
- "Disabling or enabling the ActivitySession service" on page 330

- “Configuring the default ActivitySession timeout for an application server” on page 332
- “Troubleshooting ActivitySessions” on page 333

The ActivitySession service

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. An ActivitySession context can be longer-lived than a global transaction context and can encapsulate global transactions.

Support for the ActivitySession service is shown in the following figure:



The ActivitySession service. This figure show the main components of the ActivitySession service within IBM WebSphere Application Server. For an overview of these components, see the text that accompanies this figure.

Although the purpose of a global transaction is to coordinate multiple resource managers, global transaction context is often used by J2EE applications as a ″session″ context through which to access EJB instances. An ActivitySession context is such a session context, and can be used in preference to a global transaction in cases where coordination of two-phase commit resource managers is not needed. Further, an ActivitySession can be associated with an HttpSession to extend a ″client session″ to an HTTP client.

ActivitySession support is available to Web, EJB, and J2EE-client components. EJB components can be divided into beans that exploit container-managed ActivitySessions and beans that use bean-managed ActivitySessions.

The ActivitySession service provides a UserActivitySession application programming interface available to J2EE components that use bean-managed ActivitySessions for application-managed demarcation of ActivitySession context. The ActivitySession service also provides a system programming interface for container-managed demarcation of ActivitySession context and for container-managed enlistment of one-phase resources (RMLTs) in such contexts.

The UserActivitySession interface is obtained by a JNDI lookup of java:comp/websphere/UserActivitySession. This interface is not available to enterprise beans that use container-managed ActivitySessions, and any attempt by such beans to obtain the interface results in a NotFound exceptions.

Using ActivitySessions with HTTP sessions

A web application that runs in the WebSphere Web container can participate in an ActivitySession context.

If the web application is designed such that several servlet invocations occur as part of the same logical application, then the servlets can use the HttpSession to preserve state across servlet invocations. The ActivitySession context is one state that can be suspended into the HttpSession and resumed on a future invocation of a servlet that accesses the HttpSession.

An ActivitySession is associated automatically with an HttpSession, so can be used to extend access to the ActivitySession over multiple HTTP invocations, over inclusion or forwarding of servlets, and to support EJB activation periods that can be determined by the lifecycle of the web HTTP client. An ActivitySession context stored in an HttpSession can also be used to relate work for the ActivitySession back to a specific web HTTP client.

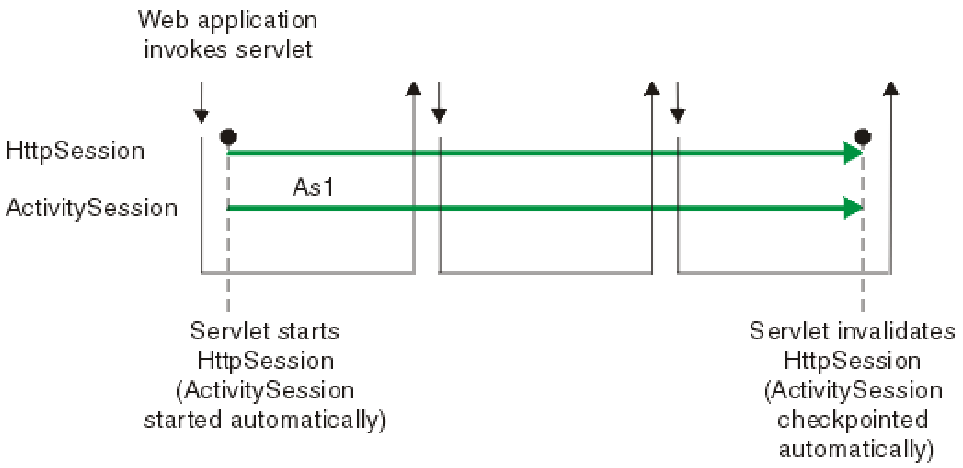
The Web container manages ActivitySessions based on deployment descriptor attributes associated with servlets in the Web application module, as described in "Configuring ActivitySession deployment attributes for a Web application" on page 329. The two usage models are:

- The Web container starts and ends ActivitySessions.

  The Web application invokes a servlet that has been configured for container control of ActivitySessions.

  - If an HttpSession exists then it has an associated ActivitySession.
  - If an HttpSession does not exist, the servlet can start an HttpSession, which causes an ActivitySession to be started automatically and associated with the HttpSession.

  A servlet cannot start a new HttpSession until an existing HttpSession has been ended. Within an HttpSession, the Web application can invoke other servlets that can use the associated ActivitySession context. When the Web application invokes a servlet that ends the HttpSession, the ActivitySession is ended automatically. This is shown in the following diagram:

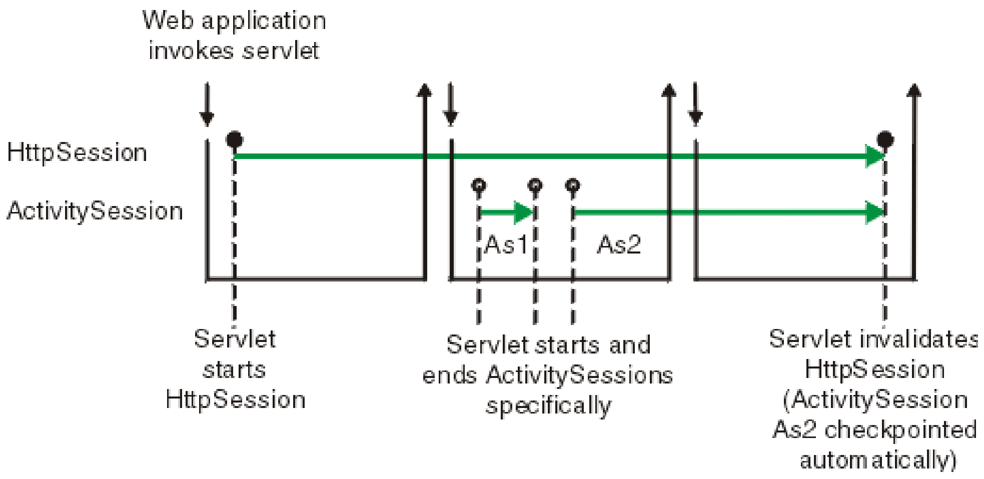

Web application invokes servlet
HttpSession
ActivitySession
As1
Servlet starts HttpSession (ActivitySession started automatically)
Servlet invalidates HttpSession (ActivitySession checkpointed automatically)

- The Web application starts and ends ActivitySessions.

  The Web application invokes a servlet that has been configured for application control of ActivitySesions.
  - If an HttpSession exists and has an associated ActivitySession, the servlet can use or end that ActivitySession context.
  - If an HttpSession does not exist, the servlet can start an HttpSession, but this does not automatically start an ActivitySession.
  - If an HttpSession exists but does not have an associated ActivitySession, the servlet can start a new ActivitySession. This automatically associates the ActivitySession with the HttpSession. The ActivitySession lasts either until the ActivitySession is specifically ended or until the HttpSession is ended.

  The servlet cannot start a new ActivitySession until an existing ActivitySession has been ended. The servlet cannot start a new HttpSession until an existing HttpSession has been ended.

  Within an HttpSession, the Web application can invoke other servlets that can use or end an existing ActivitySession context or, if no ActivitySession exists start a new ActivitySession. When the Web application invokes a servlet that ends the HttpSession, the ActivitySession is ended automatically. This is shown in the following diagram:


Web application invokes servlet
HttpSession
ActivitySession
As1
As2
Servlet starts HttpSession
Servlet starts and ends ActivitySessions specifically
Servlet invalidates HttpSession (ActivitySession As2 checkpointed automatically)

A Web application can invoke servlets configured for either usage model.

The following points apply to both usage models:
- To end an HttpSession (and any associated ActivitySession), the Web application must invalidate that session. This causes the ActivitySession to be checkpointed.
- Any downstream EJBs activated within the context of an ActivitySession can be held in memory rather than passivated between servlet invocations, because the client effectively becomes the web HTTP client.
- Web applications can be composed of many Web components, but each component in the Web application must be configured with the same value for ActivitySessionControl. ActivitySessionControl determines whether the Web component or its container starts any ActivitySessions.
- An ActivitySession context that encapsulates an active transaction context cannot be associated with an HttpSession, because a transaction can hold database locks and should be designed to be shortlived. If an application moves an active transaction to an HttpSession, the transaction is rolled back and the ActivitySession is suspended into the HTTPSession. In general, you should design applications to use ActivitySessions or other constructs as the long-lived entities and ACID transactions as short-duration entities within these.
- Only one ActivitySession can be associated with an HttpSession at any time, for the duration of the ActivitySession. An ActivitySession associated with an HttpSession remains associated for the duration of that ActivitySession, and cannot be replaced with another until the first ActivitySession is completed. The ActivitySession can be accessed by multiple servlets if they have shared access to the HttpSession.
- ActivitySessions are not persistent. If a persistent HttpSession exists longer than the server hosting it, any cached ActivitySession is terminated when the hosting server ends.
- If the HttpSession times out before the associated ActivitySession has ended, the ActivitySession is reset.
- If the ActivitySession times out, it is reset then the HttpSession is ended.

ActivitySession and transaction contexts

The ActivitySession service defines a hierarchical relationship between transaction and ActivitySession context, requiring that any transaction context be either wholly inside or wholly outside an ActivitySession context.

An ActivitySession context is very similar to a transaction context and extends the lifecycle choices for activation of enterprise beans; it can encapsulate one or more transactions. The ActivitySession context is a distributed context that, like the transaction context, can be bean- or container-managed. An ActivitySession context is used mainly by a client to scope the lifecycle of an enterprise bean that it uses either beyond or in the absence of individual transactions started by that client.

ActivitySessions have a lower overhead than transactions and can be used instead of transactions that are only used to scope the lifecycle of a called enterprise bean. For a bean with an activation policy of ActivitySession, the duration of any resource manager local transactions (RMLTs) started by that bean can be bounded by the duration of the ActivitySession instead of the bean method in which the RMLT was started. This provides flexibility and potential for using RMLTs in an enterprise bean beyond the scenarios described in the J2EE specifications. The J2EE specifications define that RMLTs need to be completed before the end of the bean method, because the bean method is the only containment boundary for local transactions available in those specifications.

The following rules defines the relationship between transactions and ActivitySessions.

- The EJB or Web container always uses a local transaction containment (LTC) if there is no global transaction present. An LTC can be method-scoped or ActivitySession-scoped.
- Before a method dispatch, the container ensures that there is always either an LTC or global transaction context, but never both contexts.
- ActivitySessions cannot be nested within each other. Any attempt to start a nested ActivitySession results in a com.ibm.websphere.ActivitySession.NotSupportedException on UserActivitySession.beginSession().
- An ActivitySession can wholly encapsulate one or more global transactions.
- An ActivitySession cannot be encapsulated by a global transaction nor should ActivitySession and global transaction boundaries overlap. Any attempt to start an ActivitySession in the presence of a global transaction context results in a com.ibm.websphere.ActivitySession.NotSupportedException on UserActivitySession.beginSession(). Any attempt to call endSession(EndModeCheckpoint) on an ActivitySession that contains an incomplete global transaction results in a com.ibm.websphere.ActivitySession.ContextPendingException. Neither the global transaction nor the ActivitySession context are affected. If endSession(EndModeReset) is called then the ActivitySession is reset and the global transactions marked rollback_only.
- Each global transaction wholly encapsulated by an ActivitySession is independent of every other global transaction within that ActivitySession. A rollback of one global transaction does not affect any others or the ActivitySession itself.
- ActivitySession and global transaction contexts can coexist with an ActivitySession encapsulating one or more serially-executing global transactions.

Combining transaction and ActivitySession container policies

This topic provides details about the relationship between the deployment descriptor properties that determine how the container manages ActivitySession boundaries.

If an enterprise bean uses ActivitySessions, how the EJB container manages ActivitySession boundaries when delegating a method invocation depends on both the **ActivitySession kind** and **Transaction attribute** deployment descriptor attributes configured for the enterprise bean. The following table lists the relationship between these two properties.

In each row, the final column describes the behavior that the EJB container takes with respect to global transaction and ActivitySession context, based on the following abbreviations:

S*n*	An ActivitySession, where *n* indicates the ActivitySession instance.
T*n*	A transaction, where *n* indicates the transaction instance.

In every case where the container does not start or leave a global transaction context associated with the thread, it starts (or obtains from the bean instance) a local transaction containment and associates that with the thread. The duration of the local transaction containment is determined by a combination of the local-transaction boundary descriptor (configured as part of the application deployment descriptor, and not shown in the following table) and the presence or not of an ActivitySession context, as described in "ActivitySession and transaction contexts" on page 315.

The rows highlighted in bold are not allowed.

Container behavior for activitysession and transaction policies deployment settings

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Transaction attribute)	Received contexts	Container behaviour
Required	Required	None	Start S1, Start T1
		S1	Start T1
		T1	Suspend T1, Start S1, Start T2
		S1, T1	No Action
	Requires new	None	Start S1, Start T1
		S1	Start T1
		T1	Suspend T1, Start S1, Start T2
		S1, T1	Suspend T1, Start T2
	Supports	None	Start S1
		S1	No Action
		T1	Suspend T1, Start S1
		S1, T1	No Action
	Not supported	None	Start S1
		S1	No Action
		T1	Suspend T1, Start S1
		S1, T1	Suspend T1
	Mandatory	None	Exception
		S1	Exception
		T1	Exception
		S1, T1	No action
	Never	None	Start S1
		S1	No Action
		T1	Suspend T1, Start S1
		S1, T1	Exception

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Transaction attribute)	Received contexts	Container behaviour
Requires new	Required	None	Start S1 + T1
		S1	Suspend S1, Start S2 + T1
		T1	Suspend T1, Start S1 + T2
		S1 + T1	Suspend S1 + T1, Start S2 + T2
	Requires new	None	Start S1 + T1
		S1	Suspend S1, Start S2 + T1
		T1	Suspend T1, Start S1 + T2
		S1 + T1	Suspend S1 + T1, Start S2 + T2
	Supports	None	Start S1
		S1	Suspend S1, Start S2
		T1	Suspend T1, Start S1
		S1, T1	Suspend S1 + T1, Start S2
	Not supported	None	Start S1
		S1	Suspend S1, Start S2
		T1	Suspend T1, Start S1
		S1, T1	Suspend S1 + T1, Start S2
	Mandatory	**None**	**Exception**
		S1	**Exception**
		T1	**Exception**
		S1, T1	**Exception**
	Never	None	Start S1
		S1	Suspend S1, Start S2
		T1	Suspend T1, Start S1
		S1, T1	Suspend S1 + T1, Start S2

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Transaction attribute)	Received contexts	Container behaviour
Supports	Required	None	Start T1
		S1	Start T1
		T1	No Action
		S1, T1	No Action
	Requires new	None	Start T1
		S1	Start T1
		T1	Suspend T1, Start T2
		S1, T1	Suspend T1, Start T2
	Supports	None	No Action
		S1	No Action
		T1	No Action
		S1, T1	No Action
	Not supported	None	No Action
		S1	No Action
		T1	Suspend T1
		S1, T1	Suspend T1
	Mandatory	None	Exception
		S1	Exception
		T1	No Action
		S1, T1	No Action
	Never	None	No Action
		S1	No Action
		T1	Exception
		S1, T1	Exception

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Transaction attribute)	Received contexts	Container behaviour
Not supported	Required	None	Start T1
		S1	Suspend S1, Start T1
		T1	No Action
		S1, T1	Suspend S1 + T1, Start T2
	Requires new	None	Start T1
		S1	Suspend S1, Start T1
		T1	Suspend T1, Start T2
		S1, T1	Suspend S1 + T1, Start T2
	Supports	None	No Action
		S1	Suspend S1
		T1	No Action
		S1, T1	Suspend S1 + T1
	Not supported	None	No Action
		S1	Suspend S1
		T1	Suspend T1
		S1, T1	Suspend S1 + T1
	Mandatory	None	Exception
		S1	Exception
		T1	No Action
		S1,T1	Exception
	Never	None	No Action
		S1	Suspend S1
		T1	Exception
		S1, T1	Suspend S1 + T1

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Transaction attribute)	Received contexts	Container behaviour
Mandatory	Required	None	Exception
		S1	Start T1
		T1	Exception
		S1, T1	No Action
	Requires new	None	Exception
		S1	Start T1
		T1	Exception
		S1, T1	Suspend T1, Start T2
	Supports	None	Exception
		S1	No Action
		T1	Exception
		S1, T1	No Action
	Not supported	None	Exception
		S1	No Action
		T1	Exception
		S1, T1	Suspend T1
	Mandatory	None	Exception
		S1	Exception
		T1	Exception
		S1, T1	No Action
	Never	None	Exception
		S1	No Action
		T1	Exception
		S1,T1	Exception
Never	Required	None	Start T1
		S1	Exception
		T1	No Action
		S1, T1	Exception
	Requires new	None	Start T1
		S1	Exception
		T1	Suspend T1, Start T2
		S1,T1	Exception
	Supports	None	No Action
		S1	Exception
		T1	No Action
		S1,T1	Exception
	Not supported	None	No Action
		S1	Exception
		T1	Suspend T1
		S1,T1	Exception
	Mandatory	None	Exception
		S1	Exception
		T1	No Action
		S1,T1	Exception
	Never	None	No Action
		S1	Exception
		T1	Exception
		S1,T1	Exception

Bean ActivitySession policy(ActivitySession kind)	Bean transaction policy(Transaction attribute)	Received contexts	Container behaviour
Bean managed	Bean managed	None	No Action
		S1	Suspend S1
		T1	Suspend T1
		S1, T1	Suspend S1 + T1

Developing a J2EE application to use ActivitySessions

This topic provides an overview of the scenarios for which you would develop a J2EE application to use an ActivitySession.

The following common J2EE application scenarios make use of an ActivitySession:

- Developing a J2EE application to use one or more enterprise beans that are persisted to non-transactional datastores.

  This scenario can be used by an application that needs to coordinate multiple 1-phase resource managers; for example, for two or more entity EJBs whose persistence is delegated to LocalTransaction resource adapters.

  In this scenario, the enterprise beans used by the application have an Activation policy of ActivitySession and a local transaction containment policy with a boundary of ActivitySession and resolution-control of ContainerAtBoundary. These configuration attributes are described in "Configuring ActivitySession deployment attributes for an enterprise bean" on page 325. The synchronization of the EJB state data is synchronized, by the container, with the 1-phase resource managers at ActivitySession completion and no application code is required to be aware of ActivitySession support.
- Developing a J2EE application in which an enterprise bean accesses a resource manager multiple times in different business methods.

  This scenario can be used by an application that needs to extend a resource manager local transaction (RMLT) over several business methods of an enterprise bean instance.

  In this scenario, the enterprise beans used by the application have an Activation policy of ActivitySession and a local transaction containment policy with a boundary of ActivitySession and resolution-control of Application. These configuration attributes are described in "Configuring ActivitySession deployment attributes for an enterprise bean" on page 325. The application logic starts and ends the RMLTs, for example using the javax.resource.cci.LocalTransaction interface offered by a LocalTransaction Connector, but is not constrained to start and commit the LocalTransaction in the same method.
- Developing a J2EE client application to use an ActivitySession to scope EJB activation and load-balancing.

  This scenario can be used by an application client that needs to access an entity bean instance several times in the same client session, either without needing to run under a transaction context, or with the need to run under a number of distinct and serially-executed transactions.

  In this scenario, the enterprise beans used by the application client have an Activation policy of ActivitySession and a local transaction containment policy appropriate to the function of the enterprise bean. These configuration attributes are described in "Configuring ActivitySession deployment attributes for an enterprise bean" on page 325. The J2EE client application can represent a period of user activity, for example a signon period, during which a number of interactions occur with one or more enterprise beans. If the J2EE client application begins an ActivitySession and invokes the enterprise beans within the scope of the UOW represented by the ActivitySession, then the enterprise bean instances are activated by the container on the ActivitySession boundary and remain in the active state until passivated by the container at the end of the ActivitySession. Workload affinity management based on the ActivitySession is a platform quality of service. Global transactions can begin and end within the ActivitySession, if they are wholly encapsulated by the ActivitySession and run serially. EJB instances activated at the ActivitySession boundary remain active across the serial global transactions.

- Developing a Web application client to participate in an ActivitySession context.

  A web application that runs in the WebSphere Web container can participate in an ActivitySession context. Web applications can use the UserActivitySession interface to begin and end an ActivitySession context. Also, the ActivitySession can be associated with an HttpSession, thereby extending access to the ActivitySession over multiple HTTP invocations and supporting EJB activation periods that can be determined by the lifecycle of the web HTTP client.

  The Web container manages ActivitySessions based on deployment descriptor attributes associated with the Web application module, as described in "Configuring ActivitySession deployment attributes for a Web application" on page 329.

General considerations:

- An application that is accessed under an ActivitySession context can receive a javax.transaction.InvalidTransactionException RemoteException, thrown by the EJB container when servicing any application method. This exception occurs when an instance of an enterprise bean that has an ActivitySession-based activation policy becomes involved with concurrent global and local transactions.
- To enable an enterprise bean to participate in an ActivitySession context and support ActivitySession-based operations, it must be configured with an ActivationPolicy of ACTIVITY_SESSION. A bean configured with ActivationPolicy of either TRANSACTION or ONCE cannot participate in ActivitySession context.
- A session bean can either use container-managed ActivitySessions or implement bean-managed ActivitySessions; entity beans can only use container-managed ActivitySessions. A bean is deployed to be bean-managed or container-managed with respect to ActivitySession management by setting its transaction type deployment attribute to be bean-managed or container-managed when deploying the enterprise bean, as described in "Configuring ActivitySession deployment attributes for an enterprise bean" on page 325. A bean that uses bean-managed transactions can use bean-managed ActivitySessions; a bean that uses container-managed transactions can use container-managed ActivitySessions.
- If you want a session bean or J2EE client to manage its own ActivitySessions, you must write the code that explicitly demarcates the boundaries of an ActivitySession, as described in "Developing an enterprise bean or J2EE client to manage ActivitySessions" on page 324.

For examples of using ActivitySessions in J2EE applications, see "Samples: ActivitySessions" on page 335.

Developing an enterprise bean or J2EE client to manage ActivitySessions

Use this task to write the code needed by a session EJB or J2EE client application to manage an ActivitySession, based on the example code extract provided.

In most situations, an enterprise bean can depend on the EJB container to manage ActivitySessions within the bean. In these situations, all you need to do is set the appropriate ActivitySession attributes in the EJB module deployment descriptor, as described in "Configuring ActivitySession deployment attributes for an enterprise bean" on page 325. Further, in general, it is practical to design your enterprise beans so that all ActivitySession management is handled at the enterprise bean level.

However, in some cases you may need to have a session bean or J2EE client participate directly in ActivitySessions. You then need to write the code needed by the session bean or J2EE client application to manage its own ActivitySessions.

Note: Session beans that use BMT and have an **Activate at** setting of `Activity session` can manage ActivitySessions. Entity beans cannot manage ActivitySessions; the EJB container always manages ActivitySessions within entity beans.

When preparing to write code needed by a session bean or J2EE client application to manage ActivitySessions, consider the points described in "ActivitySession and transaction contexts" on page 315.

To write the code needed by a session EJB or J2EE client application to manage an ActivitySession, complete the following steps based on the example code extract below:

Steps for this task

1. Get an initial context for the ActivitySession.
2. Get an implementation of the UserActivitySession interface, by a JNDI lookup of the URL `java:comp/websphere/UserActivitySession`. The UserActivitySession interface is used to begin and end ActivitySessions and to query various attributes of the active ActivitySession associated with the thread.
3. **(Optional)** Set the timeout, in seconds, after which any subsequently started ActivitySessions are automatically completed by the ActivitySession service. If the session bean or J2EE client does not specifically set this value, the default timeout (300 seconds) is used.

   The default timeout can also be overridden for each application server, on the ***server*-> Activity Session Service** panel of the administrative console.
4. Start the ActivitySession, by calling the beginSession() method of the UserActivitySession.
5. Within the ActivitySession, call business methods to do the work needed. You can also call UserActivitySession methods to manage the ActivitySession; for example, to get the status of the ActivitySession or to checkpoint all the ActivitySession resources involved in the ActivitySession.
6. End the ActivitySession, by calling the endSession() method of the UserActivitySession.

Usage scenario

The following code extract provides a basic example of using the UserActivitySession interface:

```
// Get initial context
  InitialContext ic = new InitialContext();
// Lookup UserActivitySession
  UserActivitySession uas = (UserActivitySession)ic.lookup
("java:comp/websphere/UserActivitySession");

// Set the ActivitySession timeout to 60 seconds
  uas.setSessionTimeout(60);
// Start a new ActivitySession context
  uas.beginSession();
// Do some work under this context
  MyBeanA beanA.doSomething();
  ...
  MyBeanB beanB.doSomethingElse();
// End the context
  uas.endSession(EndModeCheckpoint);
```

Note: The fourth line of the preceeding example wrapped onto a second line due to the width of the page.

Configuring ActivitySession deployment attributes for an enterprise bean

Use this task to configure the ActivitySession deployment attributes for an enterprise bean to enable the bean to participate in an ActivitySession context and support ActivitySession-based operations.

You can specify ActivitySession deployment attributes as part of the deployment of an enterprise bean.

To configure the ActivitySession deployment attributes for an enterprise bean, use the Application Assembly Tool to complete the following steps:

Steps for this task

1. Launch the Application Assembly Tool.
2. Create or edit the application EAR file.

   For example, to change attributes of an existing application, click **File-> Open** then select the EAR file.
3. In the navigation pane, select the enterprise bean instance; for example, for an entity bean expand ***ejb_module_instance*-> Entity beans** then select the bean instance.

   A property dialog notebook for the enterprise bean is displayed in the property pane.
4. In the property pane, select the IBM Extensions tab.
5. In the Bean Cache group box, set the **Activate at** attribute to **Activity Session**:

   An enterprise bean with this activation policy is activated and passivated as follows:
   - On an ActivitySession boundary, if an ActivitySession context is present on activation.
   - On a transaction boundary, if a transaction context, but no ActivitySession context, is present on activation.
   - Otherwise on an invocation boundary.
6. **(Optional)** In the Local Transactions group box, set the **Boundary** attribute to **ActivitySession**:

When this setting is used, the local transaction must be resolved within the scope of any ActivitySession in which it was started or, if no ActivitySession context is present, within the same bean method in which it was started.

7. For entity beans, or session beans, set the ActivitySessions properties for each EJB method.
   a. In the navigation pane, select **Container ActivitySessions**.

      This displays a table of the methods for enterprise beans.
   b. For each method of the enterprise bean set the **ActivitySession kind** attribute to specify how the container must manage the ActivitySession boundaries when delegating a method invocation to an enterprise bean's business method:

      Supports
      If the client invokes the bean method within an ActivitySession, the container invokes the bean method within an ActivitySession context. If the client invokes the bean method without a ActivitySession context, the container invokes the bean method without an ActivitySession context. The ActivitySession context is passed to any enterprise bean objects or resources that are used by this bean method.

      Not supported
      The container invokes bean methods without an ActivitySession context. If a client invokes a bean method from within an ActivitySession context, the container suspends the association between the ActivitySession and the current thread before invoking the method on the enterprise bean instance. The container then resumes the suspended association when the method invocation returns. The suspended ActivitySession context is not passed to any enterprise bean objects or resources that are used by this bean method.

      Never The container invokes bean methods without an ActivitySession context.
      - If the client invokes a bean method from within an ActivitySession context, the container throws an InvalidActivityException exception, which is a javax.rmi.RemoteException.
      - If the client invokes a bean method from outside an ActivitySession context, the container behaves in the same way as if the **Not Supported** value was set. The client must call the method without an ActivitySession context.

      Required
      The container invokes the bean method within an ActivitySession context. If a client invokes a bean method from within an ActivitySession context, the container invokes the bean method within the client ActivitySession context. If a client invokes a bean method outside an ActivitySession context, the container creates a new ActivitySession context and invokes the bean method from within that context. The ActivitySession context is passed to any enterprise bean objects or resources that are used by this bean method.

      Requires new
      The container always invokes the bean method within a new ActivitySession context, regardless of whether the client invokes the

method within or outside an ActivitySession context. The new ActivitySession context is passed to any enterprise bean objects or resources that are used by this bean method.

Any received ActivitySession context is suspended for the duration of the method and resumed after the method ends. The container starts a new ActivitySession before method dispatch and completes it after the method ends.

Mandatory
: The container always invokes the bean method within the ActivitySession context associated with the client. If the client attempts to invoke the bean method without an ActivitySession context, the container throws an ActivityRequiredException exception to the client. The ActivitySession context is passed to any EJB object or resource accessed by an enterprise bean method.

  The ActivityRequiredException exception is javax.rmi.RemoteException.

How the container manages the ActivitySession boundaries when delegating a method invocation depends on both the **ActivitySession kind** attribute, set here, and the **Transaction attribute** attribute, as described in"Setting transactional attributes in the deployment descriptor" (not in this document). For more detail about the relationship between these two properties, see "Combining transaction and ActivitySession container policies" on page 316.

8. To apply the changes and close the Application Assembly Tool, click **OK**. Otherwise, to apply the values but keep the property dialog open for additional edits, click **Apply**.
9. **(Optional)** To see changes reflected in your application, regenerate deployment code and reinstall the deployable archive.

Container ActivitySession assembly properties for EJB modules

Use this page to specify how a container must manage the scope of an ActivitySession for an enterprise bean's method invocations.

Name

Specifies a name for the mapping between an ActivitySession attribute and one or more methods.

Datatype
: String

Description

Contains text that describes the mapping

Datatype
: String

Methods

The methods to which the ActivitySession attribute applies.

To add a new method, click **Add**. Expand the tree to select the method or methods from the EJB module

ActivitySession attribute

How the container must manage the activity session boundaries when delegating a method invocation to an enterprise bean's business method

Default
: Supports

Range

Not supported
: The container invokes bean methods without an ActivitySession context. If a client invokes a bean method from within an ActivitySession context, the container suspends the association between the ActivitySession and the current thread before invoking the method on the enterprise bean instance. The container then resumes the suspended association when the method invocation returns. The suspended ActivitySession context is not passed to any enterprise bean objects or resources that are used by this bean method.

Required
: The container invokes the bean method within an ActivitySession context. If a client invokes a bean method from within an ActivitySession context, the container invokes the bean method within the client ActivitySession context. If a client invokes a bean method outside an ActivitySession context, the container creates a new ActivitySession context and invokes the bean method from within that context. The ActivitySession context is passed to any enterprise bean objects or resources that are used by this bean method.

Supports
: If the client invokes the bean method within an ActivitySession, the container invokes the bean method within an ActivitySession context. If the client invokes the bean method without a ActivitySession context, the container invokes the bean method without an ActivitySession context. The ActivitySession context is passed to any enterprise bean objects or resources that are used by this bean method.

Requires new
: The container always invokes the bean method within a new ActivitySession context, regardless of whether the client invokes the method within or outside an ActivitySession context. The new ActivitySession context is passed to any enterprise bean objects or resources that are used by this bean method.

  Any received ActivitySession context is suspended for the duration of the method and resumed after the method ends. The container starts a new ActivitySession before method dispatch and completes it after the method ends.

Mandatory
: The container always invokes the bean method within the ActivitySession context associated with the client. If the client attempts to invoke the bean method without an ActivitySession context, the container throws an ActivityRequiredException exception to the client. The ActivitySession context is passed to any EJB object or resource accessed by an enterprise bean method.

The ActivityRequiredException exception is defined as a javax.rmi.RemoteException that is propagated over an ORB boundary as a CORBA.ACTIVITY_REQUIRED system exception.

EJB clients that access these entity beans must do so within an existing ActivitySession. For other enterprise beans, the enterprise bean or bean method must implement bean-managed ActivitySessions or use the **Required** or **Requires New** value. For non-enterprise bean EJB clients, the client must invoke an ActivitySession by using the UserActivitySession interface.

Never The container invokes bean methods without an ActivitySession context.

- If the client invokes a bean method from within an ActivitySession context, the container throws an InvalidActivityException exception, which is defined as a javax.rmi.RemoteException that is propagated over an ORB boundary as a CORBA.INVALID_ACTIVITY system exception.
- If the client invokes a bean method from outside an ActivitySession context, the container behaves in the same way as if the **Not Supported** value was set. The client must call the method without an ActivitySession context.

Configuring ActivitySession deployment attributes for a Web application

Use this task to configure the ActivitySession deployment attributes for a Web application to start UserActivitySessions and perform work scoped within ActivitySessions.

You can specify ActivitySession deployment attributes as part of the deployment of a Web application.

To configure the ActivitySession deployment attributes for a Web application, use the Application Assembly Tool to complete the following steps:

Steps for this task

1. Launch the Application Assembly Tool.
2. Create or edit the Web module.

   For example, to change attributes of an existing module, click **File-> Open** then select the archive file for the module.
3. In the navigation pane, expand **web_application-> Web Components** then select the servlet instance.

   A property dialog box for the servlet instance is displayed in the property pane.
4. In the property pane, select the WAS Enterprise tab.

   This displays the Enterprise properties in the property pane.
5. Set the **ActivitySession control kind** attribute to either Application, Container, or None. All Web components in a Web application must be configured with the same value for ActivitySession control kind.

   Application
   The Web application is responsible for starting and ending ActivitySessions, as follows:

- If an HttpSession is active when an application begins an ActivitySession, then the container associates the ActivitySession with the HttpSession.
- If an ActivitySession is started in the absence of an HttpSession, then the application must ensure it is completed before the dispatched method completes; otherwise, an exception results.
- If an HttpSession is associated with a request dispatched to an application with this ActivitySession control value, and if that HttpSession has an ActivitySession associated with it, then the container dispatches the request in the context of that ActivitySession. For example, the container resumes the ActivitySession context onto the thread before the dispatch.
- A Web application can use both transactions and ActivitySessions. Any transactions started within the scope of an ActivitySession must be ended by the web component that started them and within the same request dispatch.

Container
A servlet has no access to UserActivitySessions. Any HttpSession started by the servlet has an ActivitySession automatically associated with it by the container, and this ActivitySession is put onto the thread of execution. If such a servlet is dispatched by a request that has an HttpSession containing no ActivitySession, then the container starts an ActivitySession and associates it with the HttpSession and the thread.

A Web application can use both transactions and ActivitySessions. Any transactions started within the scope of an ActivitySession must be ended by the web component that started them and within the same request dispatch.

None A servlet has no access to UserActivitySessions, and no participation in an ActivitySession is tolerated. Any HttpSession containing an ActivitySession that is associated with a request dispached on such a servlet is rejected with a ServletException.

6. To apply the changes and close the Application Assembly Tool, click **OK**. Otherwise, to apply the values but keep the property dialog open for additional edits, click **Apply**.
7. **(Optional)** To see changes reflected in your application, regenerate deployment code and re-install the deployable archive.

Disabling or enabling the ActivitySession service

Use this task to disable or enable the ActivitySession service for an application server.

You can use the ActivitySession **Startup** property to specify whether or not the ActivitySession service is started automatically for an application server.

To configure the ActivitySession **Startup** property for an application server, use the Administrative console to complete the following steps:

Steps for this task

1. Start the Administrative console.
2. In the navigation pane, expand **Servers-> Manage Application Servers**
   This displays a list of the application servers in the content pane.

3. In the Content pane, select the application server that you want to configure.

   This displays the properties for the application server in the content pane.
4. In the Additional Properties table, select **ActivitySession service**. This displays the ActivitySession service properties in the content pane.
5. Select or clear the **Startup** property as needed:

   Selected
      [Default] The ActivitySession service is started when the application server is started. This enables applications that specify use of ActivitySessions in their deployment descriptors to run on such an application server.

   Cleared
      The ActivitySession service is not started when the application server is started. Applications that specify use of ActivitySessions in their deployment descriptors cannot start on such an application server.

      Any attempt to start an application that uses ActivitySessions is rejected and a message similar to the following is issued:

      ```
      WACS0043E: Error found starting an application. application_name
      specified an ActivitySession attribute that is not allowed when the
      ActivitySession service is not enabled
      ```

      If this happens during server startup, the server continues to start without the application.
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

ActivitySession service settings

Use this page to administer the run-time properties of the ActivitySession service.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Activity Session Service**.

Startup

Specifies whether the server will attempt to start the ActivitySession service.

Default
   Selected

Range

   Selected
      When the application server starts, it attempts to start the ActivitySession service automatically.

   Cleared
      The server does not try to start the ActivitySession service. If ActivitySessions are to be used in applications that run on this server, the system administrator must start the service manually or select this property then restart the server.

Default timeout

The default timeout for an ActivitySession. A server resets an ActivitySession if a remote client has failed to complete the ActivitySession within this time period.

The Default ActivitySession timeout specifies the time after which an ActivitySession is completed automatically by the ActivitySession service, if a remote client has failed to complete the ActivitySession within the specified time. The initial default timeout can be configured separately for each application server, and can be overridden programmatically by the UserActivitySession interface (setSessionTimeout).

Data type
Integer

Units Seconds

Default
300 (5 minutes)

Range -1 through 2147483647 seconds
- -1 indicates that ActivitySessions never timeout
- 0 indicates that the default timeout applies
- Other values are an integer number of seconds

Configuring the default ActivitySession timeout for an application server

Use this task to configure the default ActivitySession timeout for an application server, after which any started ActivitySessions are completed automatically by the ActivitySession service.

The ActivitySession timeout is used to reset any ActivitySession whose remote client has failed to complete the ActivitySession in a timely fashion. The initial default timeout can be configured separately for each application server, and can be overridden programmatically by the UserActivitySession interface (setSessionTimeout). If an ActivitySession that contains a transaction reaches the timeout, the transaction's timeout is accelerated so that it is timed out (and rolled back) immediately before the ActivitySession is reset.

To configure the default ActivitySession timeout for an application server, use the WebSphere Administrative console to complete the following steps:

Steps for this task
1. Start the WebSphere Administrative console.
2. In the navigation pane, expand **Servers-> Manage Application Servers**
   This displays a list of the application servers in the content pane.
3. In the Content pane, select the application server that you want to configure.
   This displays the properties for the application server in the content pane.
4. In the Additional Properties table, select **ActivitySession service**. This displays the ActivitySession service properties in the content pane.
5. Set the **ActivitySession timeout** property to the default timeout as an integer number of seconds.
   - -1 indicates that ActivitySessions never timeout
   - 0 indicates that the default timeout, 300 seconds, applies
   - Other values are an integer number of seconds
6. Click **OK**.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.

8. **(Optional)** To have the changed configuration take effect, stop then restart the application server.

ActivitySession service settings

Use this page to administer the run-time properties of the ActivitySession service.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Activity Session Service**.

Startup

Specifies whether the server will attempt to start the ActivitySession service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the ActivitySession service automatically.

Cleared
The server does not try to start the ActivitySession service. If ActivitySessions are to be used in applications that run on this server, the system administrator must start the service manually or select this property then restart the server.

Default timeout

The default timeout for an ActivitySession. A server resets an ActivitySession if a remote client has failed to complete the ActivitySession within this time period.

The Default ActivitySession timeout specifies the time after which an ActivitySession is completed automatically by the ActivitySession service, if a remote client has failed to complete the ActivitySession within the specified time. The initial default timeout can be configured separately for each application server, and can be overridden programmatically by the UserActivitySession interface (setSessionTimeout).

Data type
Integer

Units Seconds

Default
300 (5 minutes)

Range -1 through 2147483647 seconds
- -1 indicates that ActivitySessions never timeout
- 0 indicates that the default timeout applies
- Other values are an integer number of seconds

Troubleshooting ActivitySessions

Use this overview task to help resolve a problem that you think is related to the ActivitySession service.

To identify and resolve ActivitySession-related problems, you can use the standard IBM WebSphere Application Server RAS facilities. If you encounter a problem that you think might be related to ActivitySessions, complete the following stages:

Steps for this task

1. Check for ActivitySession messages in the admin console.

   The ActivitySession service produces diagnostic messages prefixed by ″WACS″. The error message indicates the nature of the problem and provides some detail. The associated message information provides an explanation and any user actions to resolve the problem.
2. Check for ActivitySession messages in the activity log.

   Activity log messages produced by the ActivitySession service are accompanied by log analyzer descriptions.
3. Check for more messages in the application server's stdout.log.

   For more information about a problem, check the stdout.log file for the application server, which should contain more error messages and extra details about the problem.

The ActivitySession service application programming interfaces

The ActivitySession service consists of an application programming interface available to Web applications, session EJBs, and J2EE client applications for application-managed demarcation of ActivitySession context. Applications use the UserActivitySession interface, which provides demarcation scope methods.

ActivitySession API

The ActivitySession service provides the UserActivitySession interface for use by EJB Session beans using bean-managed context demarcation, Web application components configured with **ActivitySession control**=`Web Application`, and J2EE client applications. This UserActivitySession interface defines the set of ActivitySession operations available to an application component. An implementation of this interface is obtained via a JNDI lookup of the URL ″java:comp/websphere/UserActivitySession″. It is used to begin and end ActivitySessions and to query various attributes of the active ActivitySession associated with the thread.

For more information about the ActivitySession API, see the ActivitySession API (Javadoc), which is located in the InfoCenter.

The ActivitySession API and the implementation of its interfaces is contained in the com.ibm.websphere.ActivitySession package.

Programming Examples

The following code extract provides a basic example of using the UserActivitySession interface:

```
// Get initial context
  InitialContext ic = new InitialContext();
// Lookup UserActivitySession
  UserActivitySession uas = (UserActivitySession)ic.lookup
("java:comp/websphere/UserActivitySession");

// Set the ActivitySession timeout to 60 seconds
  uas.setSessionTimeout(60);
// Start a new ActivitySession context
  uas.beginSession();
// Do some work under this context
  MyBeanA beanA.doSomething();
  ...
  MyBeanB beanB.doSomethingElse();
// End the context
  uas.endSession(EndModeCheckpoint);
```

Note: The fourth line of the preceeding example wrapped onto a second line due to the width of the page.

Samples: ActivitySessions

The following ActivitySession samples are provided with IBM WebSphere Application Server:

MasterMind sample

This sample is based on the game MasterMind. It consists of the following components:

- A servlet, configured with Activity session contol set to Container, that accesses a stateful session bean.
- A stateful session bean, configured with an activation policy of ActivitySession containing transient state data.

The servlet begins an HttpSession at the start of each new game, and ends it at the end of each game; therefore an ActivitySession lasts for the duration of each game. The ActivitySession activation policy stops the bean from being passivated and therefore the transient data remains in memory. This is to demonstrate HttpSession/ActivationSession association in the web container, and an ActivitySession-scoped activation policy.

J2EE client container application and a CMP entity bean backed by a 1-phase-commit datasource

In this sample, the entity bean is configured with the following properties:

- TX_NOT_SUPPORTED
- An ActivitySession container managed policy of REQUIRES
- An LTC boundary of ActivitySession
- An LTC Resolution Control of ContainerAtBoundary

The client accesses the UserActivitySession, begins an ActivitySession, updates two instances of the bean, then ends the ActivitySession. It does this twice using EndModeReset then EndModeCheckpoint. This sample demonstrates the following functionality:

- Client access to the UserActivitySession interface
- Multiple RMLTs being scoped to the ActivitySession and automatically taking their completion direction from that of the ActivitySession

The entity bean also holds a transient variable incremented by each method call (gets and sets for the persistent data). This value is checked before the end of the ActivitySession to show that the same bean instance is used. The client checks for the correct results.

A J2EE client container application and two session beans with different ActivitySession types

This sample consists of a J2EE client container application and the following session beans:

- SLB1, a stateless session bean configured with an ActivitySession Type of Bean.
- SFB2, a stateful session bean configured with ActivitySession Type of Requires, an LTC boundary of ActivitySession, LTC Resolution Contol of APPLICATION, and an LTC Unresolved Action of ROLLBACK.

Both beans are configured with TX_NOTSUPPORTED.

This sample performs the following steps:

1. The client starts SLB1
2. SLB1 accesses the UserActivitySession interface, begins an ActivitySession, then calls a method on SFB2
3. SFB2 accesses the UserActivitySession interface, begins an ActivitySession, calls a method on SFB2
4. SFB2 gets a connection (setAutoCommit false) then uses JDBC to update a single-phase datasource.
5. SLB1 then optionally calls a seperate method on SFB2 to finish the work either committing or rolling-back the RMLT.
6. SLB1 then ends the ActivitySession with an EndModeCheckpoint.

This demonstrates that the ActivitySession completion direction is unconnected to the direction of the RMLTs, although their containment is bound to the ActivitySession, and the use of the container using the unresolved action when the RMLT is not completed. It also shows a bean-managed ActivitySessions bean using the UserActivitySession interface. The sample checks for correct results and reports them back to the client.

ActivitySession service: Resources for learning

Use the following links to find relevant supplemental information about ActivitySessions. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

- "Programming model and decisions"
- "Programming specifications"
- "Other"

Programming model and decisions

- ActivitySession API (Javadoc)

  See the Javadoc in the InfoCenter

Programming specifications

- J2EE Activity Service for Extended Transactions

  http://www.jcp.org/en/jsr/detail?id=95
- Java Transaction API (JTA) 1.0.1

  http://java.sun.com/products/jta/

Other

- WebSphereApplication Server Enterprise Version 5 Overview: Advanced Transactional Connectivity

  http://www-3.ibm.com/software/info1/websphere/index.jsp?tab=products/appserv_enterprise#advanced

- Listing of PDF files to learn about WebSphere Application Server Version 5
  http://www-3.ibm.com/software/webservers/appserv/was/
- Listing of all IBM WebSphere Application Server Redbooks
  http://publib-b.boulder.ibm.com/Redbooks.nsf/Portals/WebSphere
- Listing of all IBM WebSphere Application Server Whitepapers
  http://www-4.ibm.com/software/webservers/appserv/whitepapers.html
- WebSphere Application Server Enterprise Edition 4.0: A Programmer's Guide
  http://www.redbooks.ibm.com/redbooks/SG246504.html

Notices

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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 information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information 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.

Trademarks