WebSphere software

IBM

Redbooks Paper

Peter Kovari
Daniel Cerecedo Diaz

Transactional services in WebSphere Application Server Enterprise V5

The transactional support provided by J2EE architecture can be too constrained for some application models. WebSphere Application Server Enterprise V5 extends the architecture with new features for transactions.

This IBM® Redpaper introduces all transaction-related concepts needed to understand Last Participant Support and ActivitySession services. It then explains how these individual services work, including configuration issues, runtime management and troubleshooting. The details provided here apply to the WebSphere Application Server Enterprise V5.0 product.

This Redpaper is a sequel to the IBM Redpaper *Transactions in J2EE,* REDP3659; you can find the Redpaper at the following URL:

http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpaperAbstracts/redp3659.html

This Redpaper is an excerpt from the IBM Redbook *WebSphere Application Server Enterprise V5 and Programming Model Extensions WebSphere Handbook Series*, SG24-6932, which you can find at the following URL:

http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246932.html

Transactions in WebSphere®

The following transactional scenarios are possible in a WebSphere Application Server Enterprise V5 environment:

- A transaction involving a single one-phase commit resource.

  This model is supported by J2EE as a Local Transaction.

- A transaction involving a single two-phase commit resource.

  This model is supported under J2EE as a Local Transaction.

- A transaction involving several one-phase commit resources.

  This model is not supported by the J2EE architecture. WebSphere Application Server Enterprise V5 provides the ActivitySession service to support this model.

- A transaction involving several two-phase commit resources.

  This model is supported under J2EE as a global transaction.

- A transaction involving several two-phase commit resources and a single one-phase commit resource.

  This model is not supported by the J2EE architecture. WebSphere Application Server Enterprise V5 provides the Last Participant Support to cover this model.

Transactions overview

In this Redpaper, we use the following terms:

- Resource Manager Local Transaction (RMLT)
- Local Transaction Containment (LTC)
- Global transaction

Resource Manager Local Transaction (RMLT)

An RMLT is a resource adapter's view of a local transaction. It represents the unit of recovery on a single connection that is managed by the resource manager. The resource manager offers javax.resource.cci.LocalTransaction and java.sql.Connection interfaces to enable a bean or the container to request that the resource adapter commit or roll back its RMLT.

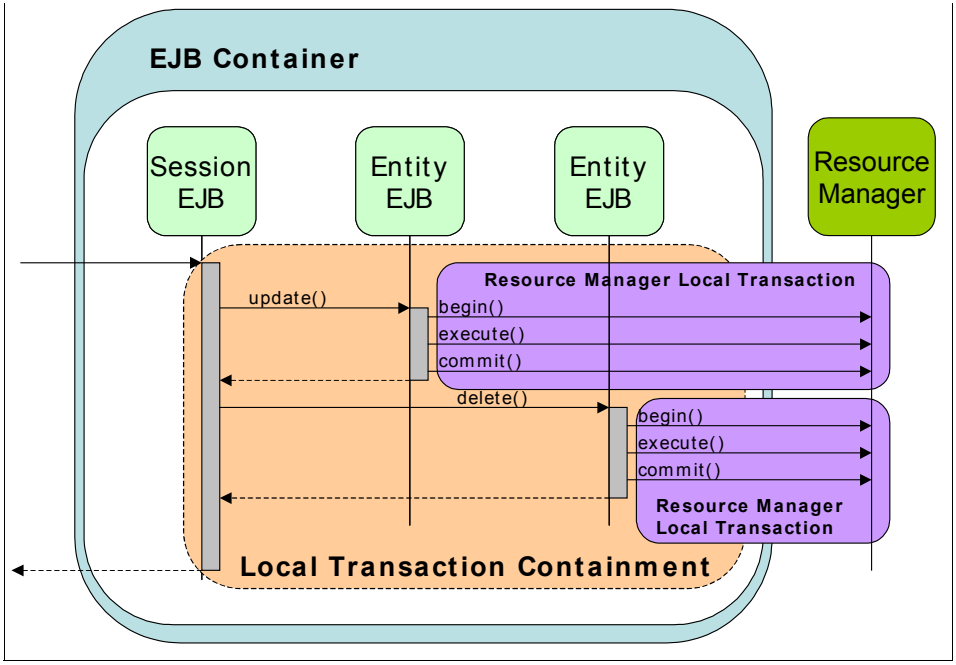


Figure 1 Resource Manager Local Transaction sample

The diagram in Figure 1 illustrates two RMLTs invoked within the same LTC. Both RMLTs involve the same resource manager. The Entity EJBs that trigger the RMLTs are BMP Entity EJBs, so explicit invocation of begin and commit methods are needed.

Local Transaction Containment (LTC)

A LTC is a bounded *unit of work scope* where zero, one, or more Resource Manager Local Transactions (RMLT) may be accessed. The LTC defines the boundary at which all RMLTs must be complete. Any incomplete RMLT may be resolved according to the resolution policy specified at deployment time. An LTC context is always established by the container in the absence of a global transaction. An LTC is local to a bean instance. LTCs are not shared across beans even if those beans are managed by the same container. The J2EE specification defines the LTC on the bean method level.

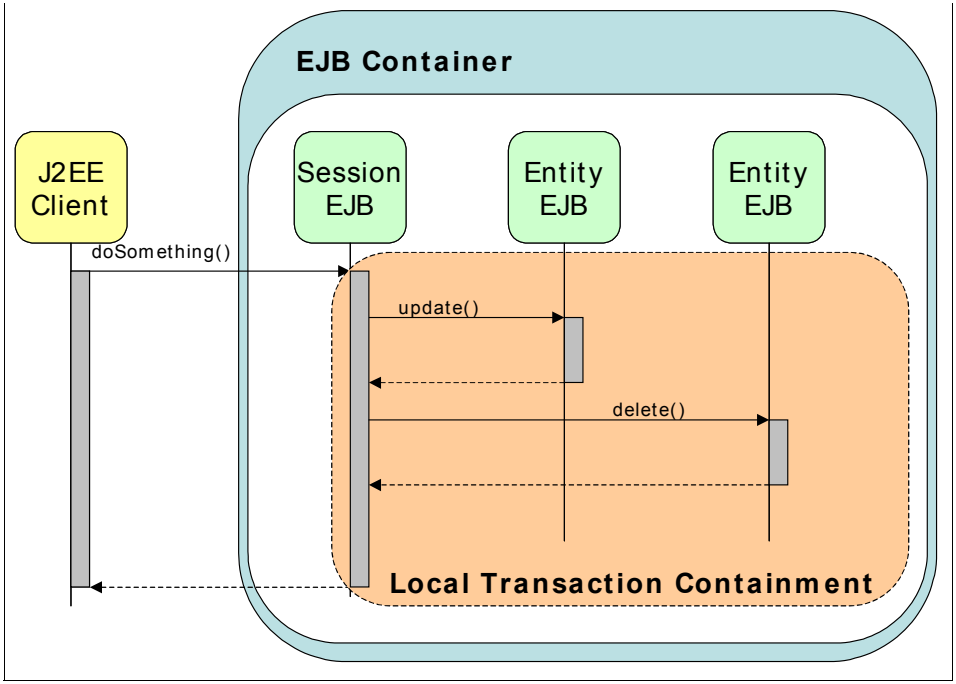


Figure 2 Local Transaction Containment sample

The diagram in Figure 2 illustrates an LTC where a Session EJB starts a Container Managed Transaction on a request of a J2EE client call. Two different Entity EJBs are involved in the LTC.

Global transaction

Global transactions are supported by resource managers capable of dealing with the two-phase commit protocol. These resource managers implement the javax.transaction.xa.XAResource interface. The application server uses a Transaction Manager to coordinate the transaction. The Transaction Manager is external to any of the resource managers and it is provided by the application server.

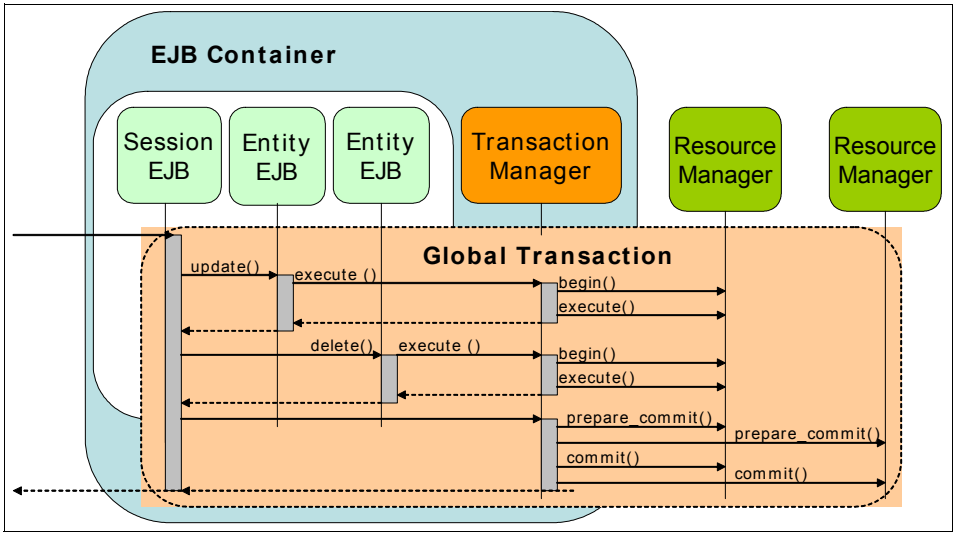


Figure 3 Global transaction

Figure 3 shows a transaction where two Container Managed (CMP) entity beans are backed up by two different XA resource managers. A global transaction context is needed. The Transaction Manager coordinates both resource managers using the two-phase commit protocol. There is no RMLT in a global transaction. From the resource manager's point of view, the transaction cannot be local, since it is externally managed by the Transaction Manager.

The application server will provide the following functions:

- Inform the Transaction Manager when a transaction begins.
- Perform the work of the transaction.
- Tell the Transaction Manager to commit the transaction.

The Transaction Manager uses the XAResource interface to coordinate the two-phase commit process across multiple resource managers. A two-phase commit, as seen in Figure 4 on page 6, works as follows:

1. In phase 1, the Transaction Manager asks all resource managers to prepare to commit their work. If a resource manager can commit its work, it replies affirmatively, and hardens its recoverable data to the permanent storage. A negative reply reports an inability to commit for any reason.
2. In phase 2, the Transaction Manager directs all resource managers either to commit or roll back the work done on behalf of the global transaction, based on the replies from phase 1.

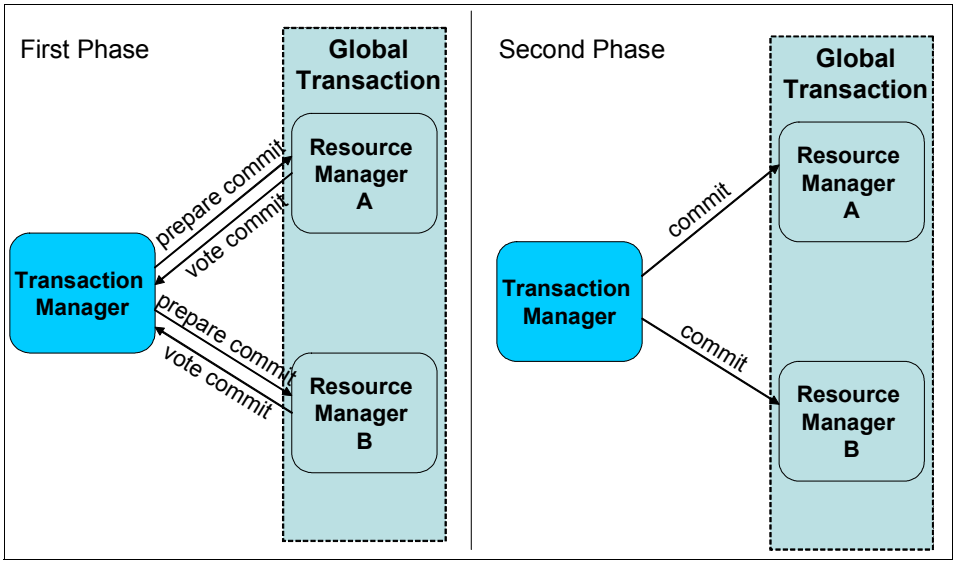


Figure 4 Two-phase commit

Last Participant Support

Last Participant Support allows the single one-phase commit resource to be involved in a global transaction where multiple two-phase commit resources execute. Multiple interactions may occur involving the one-phase commit resource in the same transaction, but only one such resource may be involved.

It is possible for a resource adapter that does not implement the XAResource interface to participate in a global transaction using Last Participant Support. This allows the use of a single one-phase commit resource in a global transaction, along with any number of two-phase commit resources. At transaction commit, the two-phase commit resources will first be prepared. If this is successful, the one-phase commit resource will be called to commit, followed by the call to commit for two-phase commit resources. See Figure 5 on page 7.

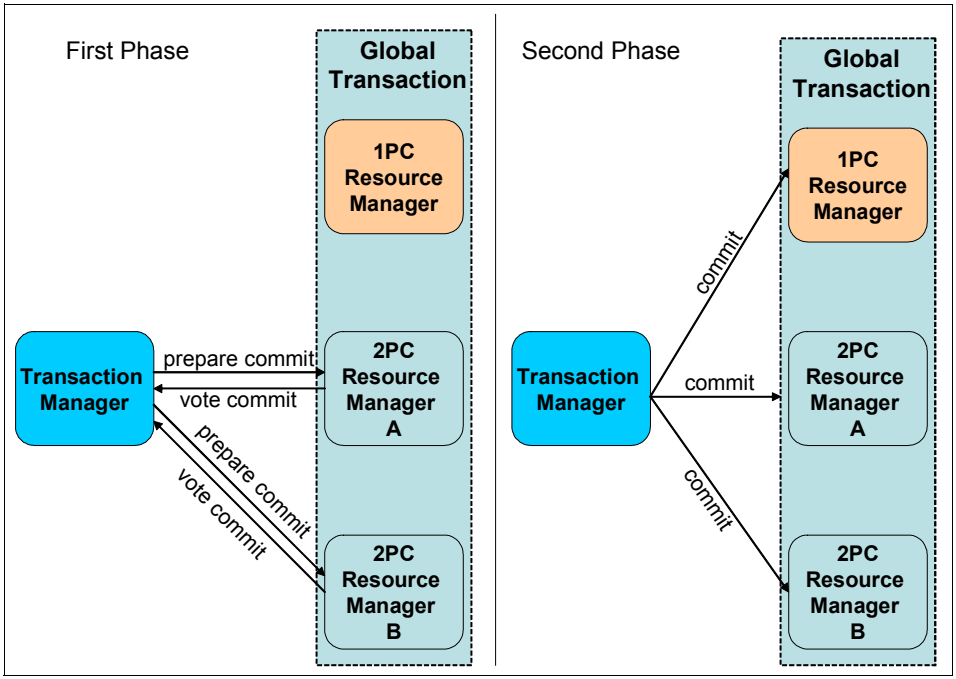


Figure 5 Last Participant Support

Last Participant Support introduces the hazard of a mixed resolution of the global transaction if the one-phase commit resource succeeds and one or more of the two-phase commit resources fails during the commit phase. The one-phase commit resource cannot be recovered.

Last Participant Support cannot be considered a substitute for the two-phase commit protocol. Applications that use Last Participant Support must be structured to handle the hazard mentioned above.

Configuration

The configuration of an Enterprise Application for Last Participant Support can be achieved using the Application Assembly Tool or the Administrative Console. In both cases, the configuration tasks are reduced to a check box that enables or disables the acceptance of a heuristic hazard.

Application Assembly Tool

To enable Last Participant Support from the Application Assembly Tool, follow these steps:

1. Open the sample enterprise application archive in the Application Assembly Tool.
2. On the left pane, select the application node.
3. On the right pane, select **WAS Enterprise** tab.
4. On the WAS Enterprise tab, select **Accept heuristic hazard**. See Figure 6.

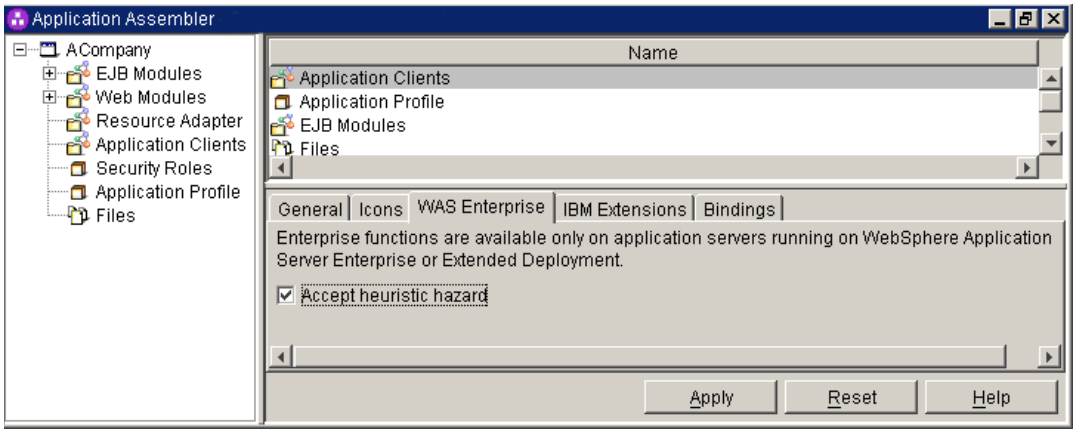


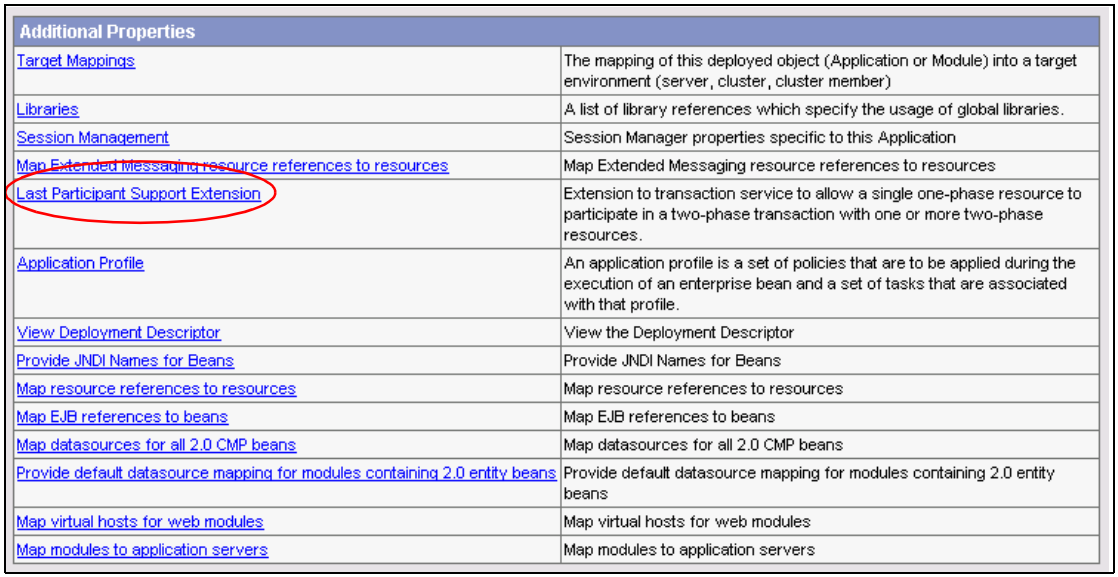
Figure 6 Application Assembly Tool: Last Participant Support enablement

5. Click **Apply** to make the configuration update.
6. Save and close the EAR file.

Administrative Console

To enable Last Participant Support from the Administrative Console, follow these steps:

1. Launch the Administrative Console and log in.
2. Select **Applications -> Enterprise Applications**.
3. On the right pane, select the name of your application: **ACompany**.
4. Under the Configuration tab, scroll down if necessary until you can see the Additional Properties table. Under Additional Properties, select **Last Participant Support** extension. See Figure 7 on page 9.



Additional Properties	
Target Mappings	The mapping of this deployed object (Application or Module) into a target environment (server, cluster, cluster member)
Libraries	A list of library references which specify the usage of global libraries.
Session Management	Session Manager properties specific to this Application
Map Extended Messaging resource references to resources	Map Extended Messaging resource references to resources
Last Participant Support Extension	Extension to transaction service to allow a single one-phase resource to participate in a two-phase transaction with one or more two-phase resources.
Application Profile	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.
View Deployment Descriptor	View the Deployment Descriptor
Provide JNDI Names for Beans	Provide JNDI Names for Beans
Map resource references to resources	Map resource references to resources
Map EJB references to beans	Map EJB references to beans
Map datasources for all 2.0 CMP beans	Map datasources for all 2.0 CMP beans
Provide default datasource mapping for modules containing 2.0 entity beans	Provide default datasource mapping for modules containing 2.0 entity beans
Map virtual hosts for web modules	Map virtual hosts for web modules
Map modules to application servers	Map modules to application servers

Figure 7 Administrative Console - Last Participant Support extension

5. Select **Accept Heuristic Hazard**.
6. Click **OK** to confirm the configuration change.
7. Save the configuration for WebSphere.
8. Restart the application for the changes to take effect.

Troubleshooting

Problems under the Last Participant Support extension can be derived from:

1. Mixing multiple single one-phase commit resources with one or more two-phase commit resources under the same transaction.

   This is not a supported scenario. Last Participant Support only accepts a single one-phase commit resource.

2. Mixing a single one-phase commit resource with one or more two-phase commit resources under the same transaction without enabling Last Participant Support.
3. A combination of the previous two problems.

Tracing

When you work on a scenario similar to the one explained in “Troubleshooting” on page 9 under the second (2) point, default traces are enough to diagnose the

problem. You can view traces in the file <WebSphere_root>/logs/<servicemen>/SystemOut.log or using the Administrative Console. The Administrative Console not only shows traces but gives a complete diagnosis, and sometimes a solution, for the referred problem.

If you try to run your application without Last Participant Support enabled, the runtime environment will throw an exception if the application has a heuristic hazard and you have not chosen to accept it.

Look at the bottom of the Administrative Console to check for errors. Clicking the link of new errors will show a detailed description of the exceptions thrown, as shown in Figure 8. Three exceptions are thrown related to this misuse of a single one-phase commit resource manager using one or more two-phase resource managers without enabling Last Participant Support.

Timestamp	Message Originator	Message
Thu Apr 17 15:18:37 EDT 2003	com.ibm.ejs.container.util.ExceptionUtil	CNTR0020E: Non-application exception occurred
Thu Apr 17 15:18:36 EDT 2003	com.ibm.ejs.j2c.LocalTransactionWrapper	J2CA0030E: Method enlist caught java.lang.Illega
Thu Apr 17 15:18:36 EDT 2003	com.ibm.ejs.jts.jta.TransactionImpl	WTRN0063E: An illegal attempt to enlist a one ph

Figure 8 Administrative Console exceptions

The first exception is triggered by the container. In the list shown in Figure 8, you can see time stamps and realize that exceptions are ordered first to last from bottom to top. Click the first exception to see the details, as shown in Figure 9.

General Properties		
Message	WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred.	Message text as received from the server runtime
Message type	Error	Type of message
Explanation	The transaction service has refused an attempt to enlist a one phase capable resource with a transaction already involving other two phase capable resources, as the application does not accept the heuristic risk that this would involve.	Explanation
User action	Ensure that one and two phase capable resources are not involved in the same transaction, or reconfigure the application to accept the heuristic risk that this would involve.	Recommendation
Message Originator	com.ibm.ejs.jts.jta.TransactionImpl	Originator of the event
Source object type	RasLoggingService	Type of the source object
Timestamp	Thu Apr 17 15:18:36 EDT 2003	Time when the event was fired
Thread Id	7fb2240f	Java runtime thread ID where the event was encountered
Node name	mka0knmy	Node which fired the event
Server name	server1	Server which fired the event

Figure 9 Administrative Console: error details

The Explanation row of the General Properties table has enough information to recognize the problem. The User Action row suggests corrective actions.

Heuristic reporting

Using Last Participant Support introduces the hazard of a mixed output if one or more two-phase resources fail to commit in the commit phase. At this moment, the *prepare commit* has been issued and the one-phase commit resource has been committed. No rollback can be performed on the one-phase commit resource. The output is mixed, since there will be committed and uncommitted resources.

More unusual is the heuristic condition that occurs when the system crashes when resolving the one-phase commit resource. In this case, the application server will be unable to determine how the one-phase commit resource was resolved. The two-phase commit resources will be rolled back. When the application server is restarted, it will inform you that such a heuristic condition has been reached, as seen in Example 1.

Example 1 Heuristic condition

```
[7/14/03 17:23:19:659 CEST] 6deed3fc TrLog         E WTRN0061W: A heuristic condition may have
occurred for transaction 000000010214b4786c9ae46a3c7e78287e72bc22039d18461f8a
```

It can be helpful, when using Last Participant Support, as well as with the ActivitySession service, to enable heuristic reporting. Heuristic reporting provides tracking of one-phase commit resources so that you can be sure that these resources have been correctly resolved or a mixed output state has been reached.

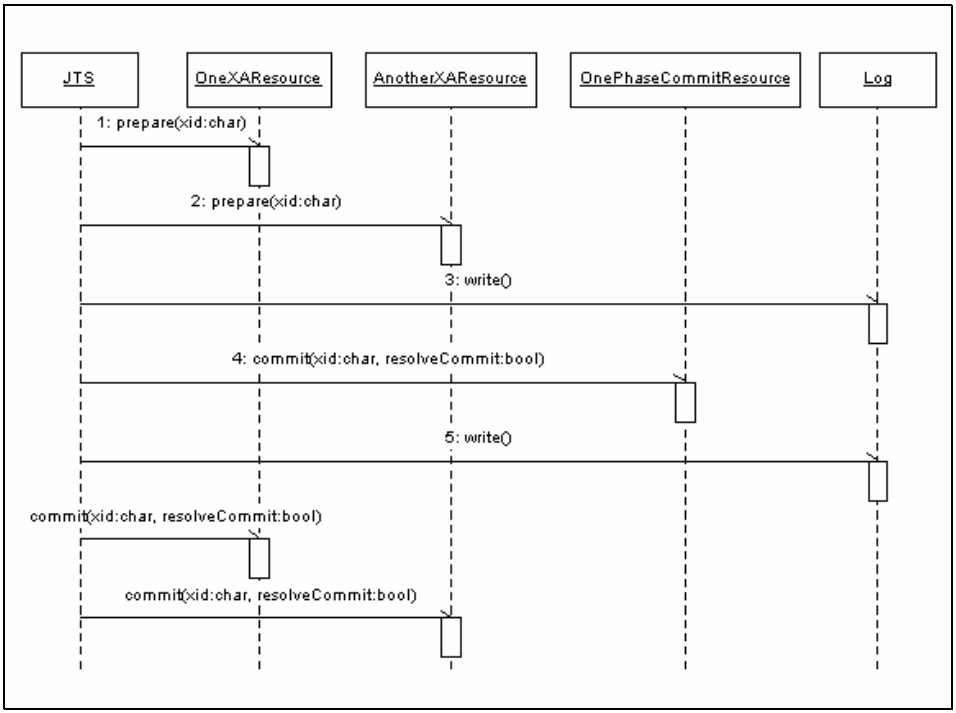


Figure 10 Heuristic reporting sequence diagram

Figure 10 shows how heuristic reporting works. The Java™ Transaction Service (JTS) coordinates the global transaction across resources. In this sample, there are two XA capable resources (two-phase commit) and a one-phase commit resource. The JTS starts with the prepare phase, as seen also in Figure 4 on page 6. In this phase, the JTS asks the XA resources for their disposition to commit. The second phase consists of committing the resources. The first to be committed is the one-phase resource. If heuristic reporting is enabled, a line is written to the log before and after committing the one-phase resource. If any of the XA resources is unable to commit, it will provoke a rollback in all the two-phase commit resources. The heuristic reporting will inform you of whether the one-phase commit resource succeeded in committing, provoking a mixed outcome, or was also rolled back. Using heuristic reporting, you will be able to recover from inconsistent states induced by mixed outcome.

To configure heuristic reporting, open the Administrative Console:

1. Select **Servers -> Application Servers**.
2. In the right pane, under the Configuration tab, scroll down through the Additional Properties table to select **Transaction service**.

3. Select the **enableLoggingForHeuristicReporting** check box.

ActivitySession

EJBs cannot extend a Resource Manager Local Transaction (RMLT) beyond the boundary of an EJB method invocation. There is a need to extend support for ACID transactions. A J2EE application accessing one or more EJBs backed by non-transactional one-phase commit resources is not capable of coordinating these resources.

ActivitySession provides a way to both extend transaction boundaries beyond method invocation and specify EJB activation and passivation by means of these new boundaries. ActivitySession becomes a new entity in the transactional scenario, defining new boundaries for a unit of work.

ActivitySession can be associated with an HTTP session; in this way, not only can the unit of work boundaries be defined by the client, but the EJB life cycle can be scoped to that client.

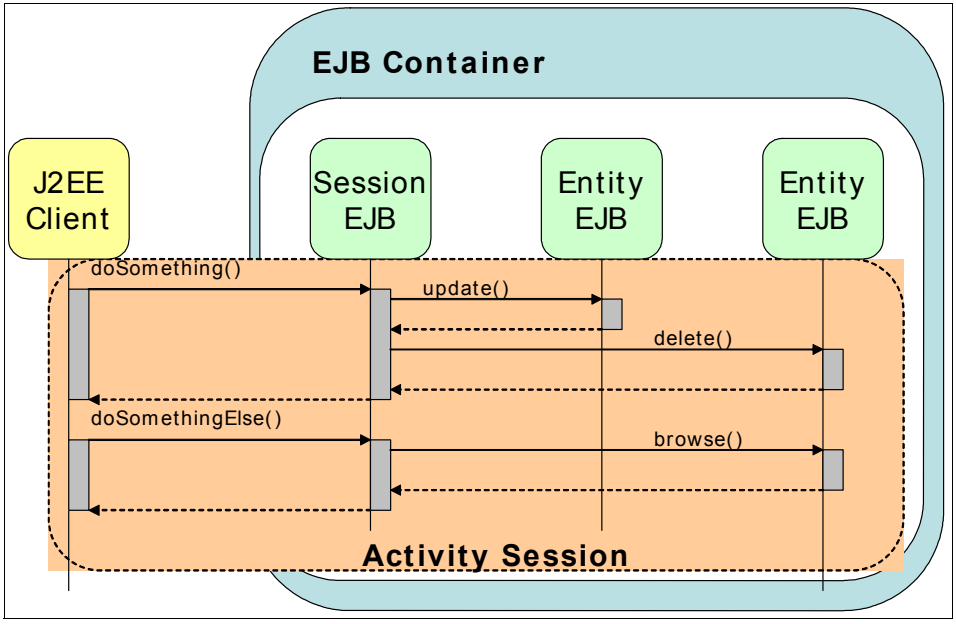


Figure 11 ActivitySession

Figure 11 shows how ActivitySession can extend the LTC. In this particular scenario, the ActivitySession is scoped to the client life cycle. Entity EJBs can be activated at the beginning of the ActivitySession and passivated at the end,

reducing the overhead of activating and passivating the EJBs on each method call.

There are a few usage rules that have to do with RMLTs, LTCs, global transactions and ActivitySessions:

- Nested ActivitySessions are not supported.
- ActivitySession boundaries cannot overlap.
- ActivitySessions may encapsulate one or more global transactions.
- Global transactions within an ActivitySession are independent of each other.
- It is not possible for an LTC to coexist with a global transaction.
- No one can wrap an ActivitySession; ActivitySessions exist on their own.

Extended Local Transaction

ActivitySession service extends the J2EE LTC beyond the method invocation providing Extended Local Transactions. Extended Local Transactions are offered in WebSphere Enterprise V5 by means of the ActivitySession as LTCs.

There are two patterns of LTC usage:

- Containment

  RMLTs within the LTC are started and completed by the application. RMLTs are said to be *contained* by the LTC. RMLTs that are not completed by the application by the end of the LTC boundary are cleaned up by the container in a direction (commit or rollback) determined by the unresolved action policy. This is a bean-managed or *programmatic* approach.

- Enlistment

  RMLTs are started by the container when the application first uses a connection. RMLTs are said to be *enlisted* in the LTC. RMLTs are completed by the container at the end of the LTC. This is a container-managed or *declarative* approach.

The ActivitySession service captures the commit operations of each local and global transaction and upholds them. At the end of the ActivitySession, the container will effectively commit or roll back each individual transaction.

The ActivitySession is not a substitute for the two-phase commit protocol. An ActivitySession may result in a mixed outcome if any single-phase resource is successfully committed before another resource fails to commit. In this case, the ActivitySession service will allow the programmer to retrieve the list of resources that were committed and those whose state is uncertain.

Configuration

The configuration of the ActivitySession service requires two steps that resemble the transactional configuration specified by the Java Transaction Architecture (JTA):

1. Configure the ActivitySession component level properties

   Components that support ActivitySession are:

   – EJB Components: Session and Entity beans
   – Web Components: servlets

2. Configure the Container ActivitySessions

   This step is only needed if the application is using an enlistment pattern. This is done in a manner very similar to the definition of Container Transactions in J2EE. ActivitySession boundaries are defined by specifying the ActivitySession support given by each EJB method.

ActivitySession component level properties

Some attributes have been added to the IBM extensions to manage LTCs:

- Boundary

  Identifies the containment boundary at which all contained RMLTs must be completed. Possible values are:

  – Bean method

    RMLTs must be resolved within the scope of the bean method where they were started. It is the default value.

  – ActivitySession

    If an ActivitySession context is present, RMLTs must be resolved within the ActivitySession scope. If no ActivitySession context is present, the policy will be the same as if the BeanMethod value were assigned.

    Restriction: A value of ActivitySession is not valid for Stateless Session Beans.

- Resolution control

  Assigns the responsibility of initiating and ending RMLTs to a component. Possible values are:

  – Application

    The application code is responsible for explicitly starting RMLTs and completing them before reaching the LTC boundary. An enlistment pattern is used.

If connections are used without calling begin()::LocalTransaction or setAutoCommit(false)::Connection, then the connections will be auto-committed by the resource adapter or the underlying resource manager. This execution model may introduce the need for an application to be notified when the LTC boundary is ending in order to fulfill its responsibility to complete RMLTs it started. A synchronization interface is exposed to implement this need.

Any incomplete RMLTs at the end of the LTC boundary will be cleaned up by the container according to the value of the unresolved action attribute.

– Container

The container is responsible for both starting RMLTs and completing them within the LTC boundary. The container begins an RMLT when a connection is first used within the LTC scope and automatically completes it at the end of the LTC scope. The boundary may be either an ActivitySession or a bean method.

- Unresolved action

Indicates the action the container will request any RMLT to take in case it is unresolved at the end of the LTC boundary.

Important: This attribute should only be applied when *resolution control* has a value of `application`. Unresolved action does not make sense in a context where it is the container that decides what action to request on the resource managers with incomplete RMLTs. The Application Assembly Tool will not check that you apply this constraint.

– Commit

Pending RMLTs will be instructed to commit by the container.

– Rollback

Pending RMLTs will be instructed to roll back by the container. This is the default value.

These parameters can be configured using the Application Assembly Tool or in WebSphere Studio IE. LTCs can be configured in EJBs as well as in Servlets.

- WebSphere Studio IE

To configure LTCs from WebSphere Studio IE, follow these steps:

– For EJBs:

i. Open the EJB Deployment Descriptor.

ii. Switch to the Beans tab.

iii. Select the EJB to configure.

iv. Scroll down the right pane to access the WebSphere Extensions.

v. Under WebSphere Extensions, the three attributes (Boundary, Resolver and Unresolved action) appear below the Local Transaction 2.0 title.

– For Servlets:

i. Open the Web Deployment Descriptor.

ii. Switch to the Servlets tab.

iii. Select the Servlet to configure.

iv. Scroll down the right pane to access WebSphere Extensions.

v. The three attributes (Boundary, Resolver and Unresolved action) appear at the bottom.

Important: In the Servlet configuration, these attributes only have meaning if ActivitySession is to be used. The Boundary attribute should always be set to `ActivitySession` since the Bean Method value does not apply. The Boundary attribute value should be inferred from the usage or non-usage of the ActivitySession.

► Application Assembly Tool

To configure LTCs from the Application Assembly Tool, follow these steps:

– For EJBs:

i. On the left pane, expand the tree by selecting **<enterprise_application> -> EJB Modules -> <ejb_module>** and then selecting either **Session Beans** or **Entity Beans**.

ii. Select the Session or Entity bean you want to configure.

iii. On the right pane, switch to the IBM Extensions tab.

iv. The three properties to be configured appear under the Local Transactions title.

– For Servlets:

i. On the left pane, expand the tree by selecting **<enterprise_application> -> Web Modules -> <web_module> -> Web Components**.

ii. Select the Servlet you want to configure.

Important: In the Application Assembly Tool, the ActivitySession service configuration attributes are in two different tabs: WAS Enterprise and IBM Extensions. The name of one of the attributes is also changed. The Resolver attribute appears under the ActivitySession control.

iii. Open the IBM Extensions tab to access the Unresolved action attribute.

iv. Switch to the WAS Enterprise tab to access the Resolver attribute. The Resolver attribute appears named under the ActivitySession control. When set to `None`, no ActivitySession is used. If `Application` or `Container` values are applied, then Boundary is set to `ActivitySession`, since no other value may apply to Servlets.

Container ActivitySession

After configuring how the ActivitySession service is going to behave in each component, you should configure what support is expected by each method. A method may be associated with the following ActivitySession attribute values:

- Never

  An ActivitySession context is forbidden. Any violation causes a RemoteException to be thrown.

- Supports

  If an ActivitySession context is received, it will be used. If no ActivitySession is present at invocation, none will be used.

- Not Supported

  No ActivitySession is needed. If one is received, it will be suspended. Upon method return, the ActivitySession will be resumed.

- Requires New

  If an ActivitySession context is received, it is suspended and a new ActivitySession is created. If no ActivitySession is present, a new one is created.

- Required

  If an ActivitySession context is received, it will be used. If no ActivitySession is active when a method is called, a new ActivitySession is created.

- Mandatory

  An ActivitySession must be present at method invocation. Otherwise, an ActivitySessionRequiredException is thrown.

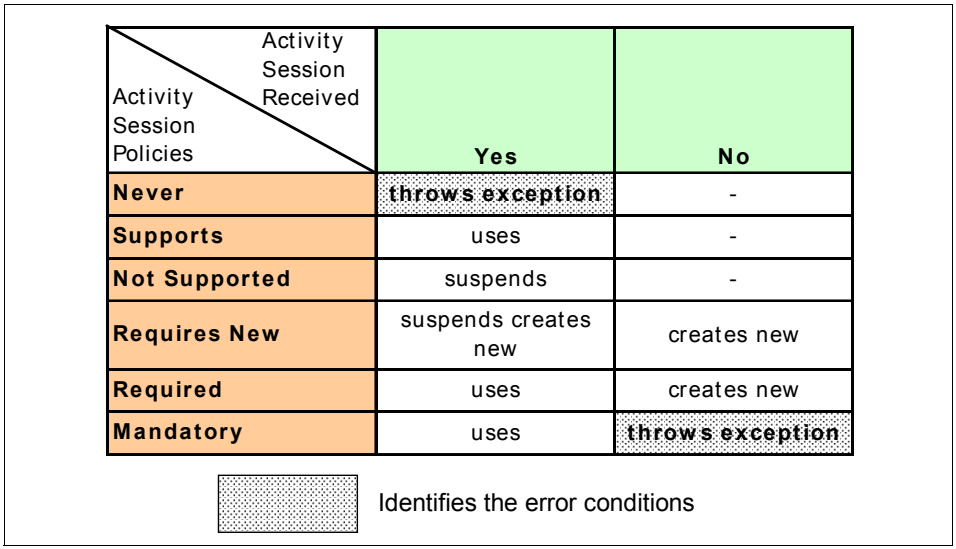


Figure 12 ActivitySession attribute values

Figure 12 shows the possible ActivitySession attribute values and their effect when an ActivitySession is received and when it is not.

Container ActivitySessions can only be configured in the Application Assembly Tool. There is no tool support for this in WebSphere Studio IE. The Container ActivitySession configuration groups methods from different components and assigns this group a support level for the ActivitySession. To create a Container ActivitySession, follow these steps:

1. In the Application Assembly Tool, expand the tree on the left pane by selecting **<enterprise_application> -> EJB Modules -> <ejb_module> -> Container ActivitySession**.
2. From the menu, select **File -> New -> Selected Object**. You will see the window shown in Figure 13 on page 20.

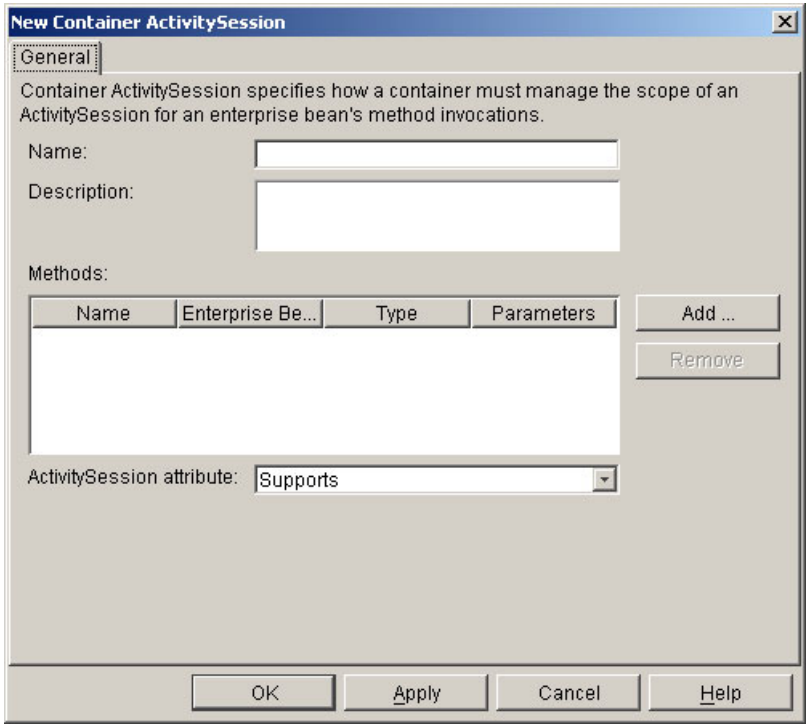


Figure 13 Application Assembly Tool - New Container ActivitySession

You must group methods with the same ActivitySession attribute under the same Container ActivitySession definition. Nevertheless, you can have different Container ActivitySessions with the same ActivitySession attribute value but a different naming. The name can add semantics to the Container ActivitySession, so you can identify the purpose of the associated methods.

3. Give a name to the Container ActivitySession.
4. Click the **Add** button. A window will open to browse the methods of the EJBs.
5. Select the methods you want to add and click **Apply** for each one.
6. Once you have finished adding methods, click **OK** to confirm the changes.
7. From the drop-down list, choose the ActivitySession attribute value required for these methods.
8. Click **OK** to create the new Container ActivitySession.

ActivitySession service API

If the Resolver is set to `Application`, a containment pattern is being used and coding using the ActivitySession service API is needed for the application. You

can achieve the same goals using ActivitySession programmatically through the API instead of configuring Container ActivitySessions.

To access the ActivitySession service API, you need to add the activitySession.jar to your project's build path. Follow these steps:

1. In WebSphere Studio IE, select **Window -> Open Perspective -> Java**.
2. In the Package Explorer view, located in the left pane, right-click the Web Project or EJB Project where you want to use the ActivitySession service. From the pop-up menu, select **Properties**.
3. On the left pane of the Properties window, select **Java Build Path**.
4. On the right pane of the Properties window, select **Libraries**.
5. Click the **Add** variable button. A window opens showing all available variables, as seen in Figure 14.

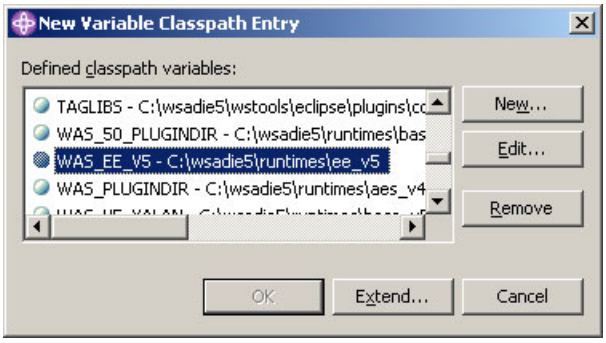


Figure 14 Build path variable

   a. Select the **WAS_EE_V5** variable. Click the **Extend** button to complete the path to the JAR file you need. A window opens and shows a directory tree.
   b. Navigate in the directory tree to the lib folder. Select **activitySession.jar**. Click **OK** to add the new library.
6. Click **OK** to confirm the changes.

WebSphere Studio IE will rebuild your project and the ActivitySession service will be available to your code.

The ActivitySession service is exposed to application programmers through the com.ibm.websphere.ActivitySession.UserActivitySession interface. The methods exposed by this interface are shown in Example 2 on page 22.

Example 2 UserActivitySession

```
public interface UserActivitySession {

   public static final int EndModeCheckPoint = 0;
   public static final int EndModeReset = 1;
   public static final int StatusSessionActive = 0;
   public static final int StatusSessionCompleting = 1;
   public static final int StatusSessionCompleted = 2;
   public static final int StatusNoSession = 3;
   public static final int StatusUnknown = 4;

   public abstract void beginSession();
   public abstract void endSession(int i);
   public abstract void resetSession();
   public abstract void checkpointSession();
   public abstract int getStatus();
   public abstract String getSessionName();
   public abstract void setSessionTimeout(int i);
   public abstract int getSessionTimeout();
   public abstract void setResetOnly();
}
```

Important: Throws clauses have been removed from the code in Example 2 for a clearer view.

An ActivitySession is started with a beginSession method call and ended with a endSession method call.

The UserActivitySession lets you checkpoint work to commit changes made up to a specific point without ending the ActivitySession.

The resetSession method call causes the ActivitySession service to roll back all changes to the last checkpoint.

You can programmatically set a timeout for the ActivitySession through the setTimeOut method call. The time is specified in seconds. It can also be set through the Administrative Console for each server instance, as seen in “Runtime” on page 28.

The implementation of this interface can only be accessed through a JNDI lookup. The UserActivitySession implementation is bound to the java:comp/websphere/UserActivitySession name. Example 3 on page 23 shows how to use the UserActivitySession.

Example 3 UserActivitySession usage

```
//perform lookup
InitialContext ic= new InitialContex();
UserActivitySession uas=
  (UserActivitySession)ic.lookup(“java:comp/websphere/UserActivitySession“);
uas.beginSession();

//do some work
myBean.doSomeWork();

//mid checkpoint
uas.checkpointSession();

//do some more work
myOtherBean.doSomeWork();

//end the session
uas.endSession(UserActivitySession.EndModeCheckPoint);
```

Important: Necessary try/catch statements have been removed for clarity. For a detailed description of exceptions that need to be caught, refer to “Exception handling” on page 29.

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.

Important: WebSphere Studio IE V5.0.1 has a bug that inhibits the use of the ActivitySession service under WebSphere Test Environment EE. To correct this configuration problem, edit the file named <WebSphere_Studio_IE>/runtimes/ee_v5/properties/implfactory.properties. Search for the key:

```
com.ibm.websphere.csi.ContainerExtensionFactory
```

Assign it the following value:

```
com.ibm.ejs.csi.ContainerExtensionFactoryPMEImpl.
```

Extended EJB life cycle

Without ActivitySession service support, there are only two different ways to keep an EJB within scope between multiple method calls:

- Using commit option A

  This option prevents the EJB from being workload-managed and disallows sharing the database tables with other applications.

- Starting a global transaction

  Global transactions need external resources (not only from the resource manager) to coordinate the different RMLTs. A transaction service such as the JTS must be used. This carries a performance weight.

ActivitySession service lets you extend the life cycle of an EJB without the performance drawback of a global transaction and avoiding the restrictions of using commit option A.

An EJB activation can be scoped to the ActivitySession, rather than being scoped to the transaction boundaries, offering additional control of the timing for EJB activation and passivation, with potential performance benefits.

This feature can highly improve performance by avoiding unnecessary passivations. EJBs remain active and ready.

Configuration

New EJB Bean cache policies have been added to existing IBM extensions. The Bean cache policies are governed by two attributes:

- Activate At

  Defines when an EJB is activated and placed in cache as well as when it is removed from cache and passivated.

  - Once

    The EJB is activated and put in cache the first time it is accessed. Removal and passivation depend on the container.

  - Transaction

    Activation and passivation as well as cache lifetime are governed by transaction boundaries. The EJB is activated and put in cache when the transaction begins and passivated and removed from cache when the transaction ends.

  - ActivitySession

    Activation and passivation as well as cache lifetime are governed by ActivitySession boundaries. This is a new policy.

- Load At

  Specifies when the container should synchronize the bean with the persistence layer. This defines whether the container has exclusive or shared access to the database.

  – Activation

    The EJB is loaded from the persistent storage when it is activated, regardless of the Activate At policy. The container will use exclusive access to the persistence layer.

  – Transaction

    The EJB is loaded at the start of the transaction. The container will use shared access to the persistence layer.

The Bean cache policy governs which commit option will be used for the specific bean where it is configured.

Load At \ Activate At	Once	Transaction	Activity Session
Activation	A	Unsupported	Unsupported
Transaction	B	C	C+

Figure 15 Bean cache policies and commit options

Figure 15 shows which commit options will be used, depending on the Load At and Activate At values of the Bean cache policy.

According to the EJB 2.1 specification, supported commit options are A, B and C. Commit option C+ is an IBM extension to support an EJB life cycle bound to the Activity Session boundaries. With commit option C+, the EJB is activated only once per ActivitySession, but data may be loaded or stored multiple times within that session at checkpoints.

- Option A

  When the transaction ends, the container caches the instance and it remains valid across transactions. The container invokes ejbStore to synchronize the instance. The instance remains both ready and valid, so the container will not load it again from the persistent storage when the next transaction begins. Exclusive access to the instance is ensured by the container.

- Option B

  The container invokes ejbStore, caches the instance but marks it as not valid. When a new transaction begins, the container invokes ejbLoad to synchronize the bean, since the container does not ensure exclusive access.

- Option C

  The container does not cache the instance. At the end of the transaction, the container invokes ejbStore and ejbPassivate and returns the instance to the pool. At the beginning of the next transaction, ejbActivate and ejbLoad are invoked on the retrieved instance to synchronize the state of the EJB.

- Option C+

  This option is different from the others because instead of referring to transaction boundaries, it refers to ActivitySession boundaries. An ActivitySession may contain multiple transactions, global or local. With commit option C+, an EJB is stored at the transaction end and loaded at the beginning of the next transaction. No activation or passivation is needed. The same object is reused between transactions.

Figure 16 summarizes these properties. The Write column specifies whether the EJB is written to persistent storage at the transaction end. The Ready column specifies whether the instance remains ready for the next transaction. If ready, the instance is not returned to the pool. The Valid column specifies whether the instance remains valid for the next transaction. If valid, the instance does not need to be synchronized with the persistent storage at the beginning of the next transaction.

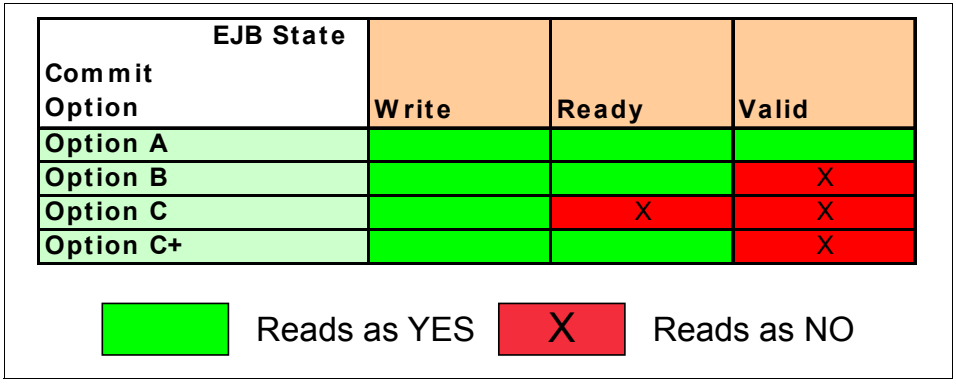


Figure 16 Commit options

The EJB life cycle is influenced by the commit options. Depending on the commit option used by the container, some EJB callback methods may be obviated to achieve better performance.

Figure 17 on page 27 shows which methods are invoked in a scenario where an Entity EJB is invoked in two consecutive transactions. The figure illustrates whether the selected method is invoked or not at the end of the first transaction and at the beginning of the second transaction.

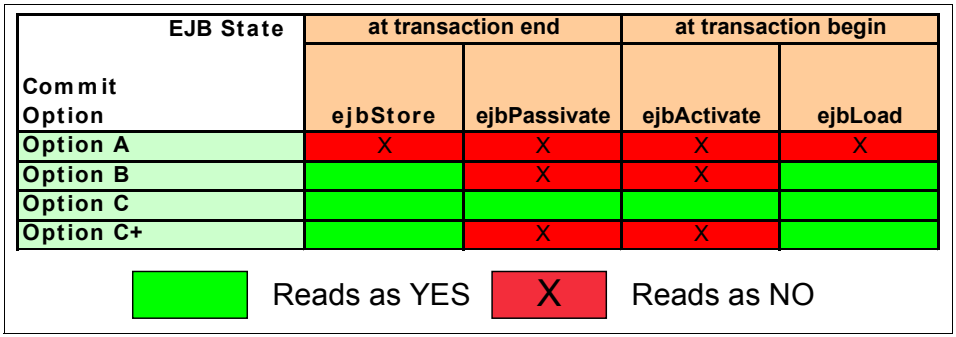


Figure 17 EJB callback methods invocation depending on commit options

Important: Both Figure 16 and in Figure 17 show different views of the same data. Notice that Option C+ in an ActivitySession context behaves in a way similar to Option B in a transaction context. Option C+ is the only possible option when using ActivitySession service for scoped activation.

Usage scenarios

Using the ActivitySession service in association with a Web container lets you extend and define transactions of EJBs during the client's life cycle.

Coordination of one-phase commit resources is not supported by the J2EE specification. ActivitySession adds more flexibility to the constrained J2EE model, enabling the coordination of one-phase commit resources under a unit of work.

Client scoped life cycle

ActivitySessions used within a Web container are automatically associated to the HTTP Session. An EJB can be activated and passivated in ActivitySession boundaries, as was seen in "Extended EJB life cycle" on page 24. Therefore, it is possible to span the life of an EJB and map it to the client's life cycle.

Client side demarcation

The association between the ActivitySession and the HTTP Session also rewards you with the possibility of client-side defined transactions. ActivitySession lets you coordinate a number of one-phase commit resources within the boundaries of a single unit of work. You can span this unit of work through multiple client invocations.

Coordination of one-phase commit resources

ActivitySession service allows the coordination of multiple one-phase commit resources within the boundaries of a single unit of work. It is not a substitute for the two-commit phase protocol and introduces a heuristic hazard.

Runtime

Although ActivitySession service usage can lead to a mixed outcome when coordinating several one-phase commit resources, there is no need to accept the heuristic hazard.

Enabling the ActivitySession service

The only runtime configuration needed is the enablement of the ActivitySession service on the server to which you want to deploy your application. To enable the ActivitySession service, follow these steps:

1. In the left pane of the Administrative Console, select **Servers -> Application Servers**.
2. In the right pane, select the server where you want to enable the ActivitySession service. The default server name is server1.
3. Select the **Configuration** tab, if not already selected.
4. Scroll down the Additional Properties table and click **ActivitySession Service**.
5. Select the **Startup** check box and specify a default timeout value. The default value is five seconds. This means that if the ActivitySession is not resolved after five seconds, it will be rolled back.

Troubleshooting

This section describes how to use WebSphere tracing facilities to recover from errors derived from mixed outcomes. It also shows how to deal with exceptions when using a bean-managed ActivitySession.

Exception handling

When using a bean-managed ActivitySession, several exceptions must be taken into account, depending on the invoked method.

- begin()::UserActivitySession
  - ActivitySessionAlreadyActiveException

    You tried to start an ActivitySession while another one was already present. ActivitySession boundaries cannot overlap one another.
  - SystemException

    Unknown error starting a new ActivitySession.
  - NotSupportedException

    ActivitySession is not supported by a nested call within this ActivitySession context.
  - TransactionPendingException

    You tried to start a new ActivitySession while a transaction was still pending. ActivitySessions and transactions cannot overlap their boundaries.
- endSession()::UserActivitySession
  - MixedOutcomeException

    A heuristic condition has been reached. Some of the resources were able to commit, whereas others were rolled back.
  - NotOriginatorException

    You tried to end the ActivitySession from a component other than the one that started the ActivitySession. ActivitySessions must be started and resolved within the same component.
  - SystemException

    Unknown error when resolving the ActivitySession.
  - NotSupportedException

    ActivitySession is not supported by a nested call within this ActivitySession context.
  - ActivitySessionResetException

    Unknown error when trying to reset the current ActivitySession.
  - NoActivitySessionException

    You tried to end an ActivitySession context but no ActivitySession context was present.

- ContextPendingException
- ActivitySessionPendingException

Heuristic reporting

You can also use the heuristic reporting facility when using the ActivitySession service. Refer to “Heuristic reporting” on page 11.

JTA extensions

Programming Model Extensions have also been introduced to JTA. These extensions are available both to Web and J2EE containers. Two new mechanisms have been added:

- Callback registration

  Provides a mechanism for applications to be notified when a transaction completes.

- Transaction identity

  Provides a mechanism to determine the transaction identity.

For more information on JTA Extensions, refer to the WebSphere InfoCenter.

The team that wrote this Redpaper

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

Peter Kovari is a WebSphere Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively about all areas of WebSphere. His areas of expertise include e-business, e-commerce, security, Internet technologies and mobile computing. Before joining the ITSO, he worked as an IT Specialist for IBM in Hungary.

Daniel Cerecedo Diaz is an IT Architect working in IBM Global Services in Madrid. He has four years of experience in J2EE development and Systems Integration. He holds a Master’s degree in Computer Science and is a Sun Certified Programmer for the Java 2 Platform. His areas of expertise include J2EE architecture, object-oriented technologies and application design.

Notices

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

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

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

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

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

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

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

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

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

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

Trademarks

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

@server™	Rational®	@server™
CICS®	Redbooks(logo) ™	WebSphere®
DB2®	ibm.com®	ibm.com®
IBM®	Redbooks™	

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

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

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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