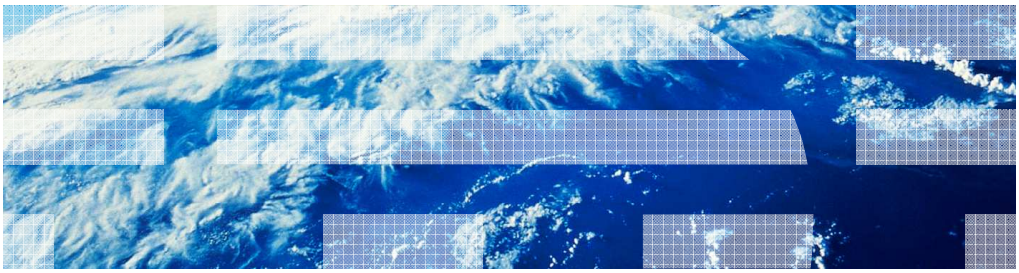
IBM WebSphere Application Server V8

Enterprise Java Beans (EJB) 3.1



WebSphere software

This presentation describes support for Enterprise Java Beans 3.1 in IBM WebSphere Application Server V8.

Table of contents

An overview of the enhancements to the EJB specification is provided, along with references.

Section

EJB 3.1 overview

An overview of the following enhancements to the EJB specification is provided, as supported in WebSphere Application Server V8:

The No-Interface local view, asynchronous session bean invocations, singleton session beans, calendar based timers, automatic timer creation, non-persistent timers, an embeddable EJB container, and packaging of EJBs in WAR files.

No-interface local view (1 of 2)

- Session beans can now be exposed to clients through a no-interface view
- Supported when:
  - The bean does not expose any other client views (Local, Remote, 2.x Remote Home, 2.x Local Home, web service) and its implements clause is empty
  - The bean exposes at least one other client view and it has @LocalBean annotation on the bean class or in the deployment descriptor

The No-Interface local view is a continuation of the ease of use of enhancements that were made in the EJB 3.0 specification. In the last release, the requirement that session beans extend framework classes was dropped, and now the requirement to provide an interface has also been dropped. The No-Interface view allows the session bean implementation class to be exposed to local clients, similar to a local business interface. All public methods of the session bean implementation class are exposed as local business methods.

A session bean automatically receives a No-Interface view when it has not been configured to have any other client view and the implements clause is empty.

In addition, a session bean may be explicitly configured to have a No-Interface view by either specifying the LocalBean annotation or the equivalent in the deployment descriptor.

With this feature, the EJB programming model now allows a “plain old Java object” or POJO to be configured as an enterprise bean, with all of the qualities of service provided by the container.

No-interface local view (2 of 2)

- Session bean with No-Interface view because no declared interfaces

```
@Stateless
Public class CartBean
```

- Session bean with No-Interface view using @LocalBean

```
@Stateless
@LocalBean
@Remote (cart.class)
public class CartBean implements Cart
```


Provided are examples that demonstrate the two techniques of configuring a No-Interface view.

In the first example, the stateless session bean does not implement any interfaces, nor does it provide any business or component interfaces. Therefore the CartBean class is exposed as a No-Interface view.

For the second example, the LocalBean annotation has been specified. The stateless session bean, CartBean, will have two client views: the remote business interface, Cart, and a No-Interface view.

Also note, that although the No-Interface view is effectively exposing the bean implementation class as a local business interface, it does not prevent the session bean from using both features together. A session bean may be configured with both a No-Interface view and local business interfaces.

Asynchronous session bean invocations

- Allows EJB methods to run asynchronously
- Improves performance and increases scalability
- Has two modes:
  - Fire and forget
  - Fire and return results

The EJB specification has now defined a mechanism to declare that a business method is to run asynchronously to the client, whether the client is local or remote. By default, business methods will continue to run synchronously.

This new feature is aimed at improving performance and scalability by allowing a client request to be processed on multiple threads. The client may call multiple asynchronous EJB methods, to process work concurrently on multiple threads, and continue processing in the client thread.

There are two forms of asynchronous EJB method invocations:

First, “fire and forget”. When using “fire and forget”, the asynchronous method is declared to have a void return type, and the application client has no way of knowing when or even if the asynchronous method completes. “Fire and forget” is useful for scenarios where there is some ‘nice to have’ function to be performed, but it is unimportant if it actually completes. By using an asynchronous request in this scenario, the performance of the important processing can be improved on the main thread, while the “nice to have” function runs concurrently on a separate thread.

Second, “fire and return results”. When using “fire and return results”, the asynchronous method is declared to have a Future return type. An instance of Future is returned to the application client immediately, before the asynchronous method runs, and the client can then use the Future to check on the status of the asynchronous request and eventually obtain the result.

Note: For Asynchronous methods, only transaction required, transaction requires new, or transaction not supported are supported. The asynchronous method will always run under a different transaction context than that of the client.

The transaction service context and the activity session service context are not propagated from the client to the asynchronous thread. The security context, and all of the WebSphere Application Server extension contexts, such as work area, internationalization, and so on, are propagated for use on the execution thread.

Asynchronous session bean invocations – fire and forget

```
public interface Email
  {
   public void sendEmail (String name, String message);
  }

  @Stateless
  @Local(Email.class)
  public class CalculatorBean {
  @Asynchronous
public void sendEmail (String name, String message);
  {
    // ... Send email.;
}
```


Here is an example of an asynchronous local business method using the “fire and forget” semantics.

Anytime a client calls the ‘sendEmail’ method on the CalculatorBean, control is returned immediately to the client, and the ‘sendEmail’ method is submitted to run concurrently on another thread.

Asynchronous session bean invocations – fire and return results (1 of 2)

Here is an example of an asynchronous local business method using the “fire and return results” semantics.

When a client calls the ‘performCalculation’ method on the CalculatorBean, an instance of Future is returned immediately to the client, and the ‘performCalculation’ method is submitted to run concurrently on another thread. The client can then use methods on the Future object to check on the status of the asynchronous method, and eventually obtain the Integer result value.

Note: The javax.ejb.AsyncResult object is a convenience implementation of the Future interface used only to pass the object to the container. The object is never passed to the client.

Asynchronous session bean invocations – fire and return results (2 of 2)

```
Import javax.ejb.AsyncResult;
....
  @Stateless
  @Local(Calculator.class)
  @LocalBean
  public class CalculatorBean {
   public someMethod(int a, int b);

 @Asynchronous
  public Future<Integer> performCalculation(int a, int b)
  {
    // ... do calculation
    Integer result = ...;
    return new AsynchResult(result);
  }
```


This is basically the same “fire and return results” example, but for a bean with both a local business interface and a No-Interface view.

The ‘performCalculation’ method will run asynchronously whether it is called through the local Calculator interface or the No-Interface view.

Asynchronous session bean invocations – Future object

Method Summary	
boolean	cancel(boolean mayInterruptIfRunning) Attempts to cancel execution of this task.
V	get() Waits if necessary for the computation to complete, and then retrieves its result.
V	get(long timeout, TimeUnit unit) Waits if necessary for at most the given time for the computation to complete, and then retrieves its result, if available.
boolean	isCancelled() Returns true if this task was cancelled before it completed normally.
boolean	isDone() Returns true if this task completed.

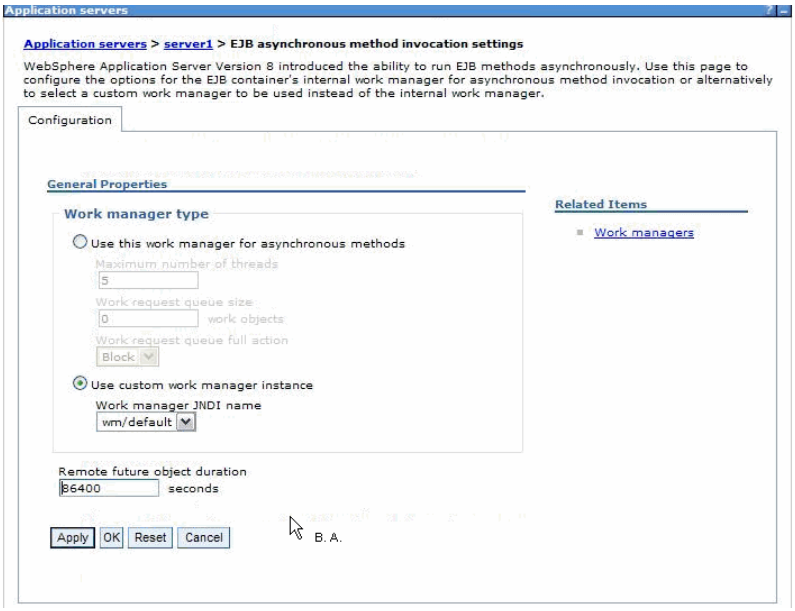
For reference, here are the methods provided on the returned Future object.

The ‘get’ methods allow the client to obtain the result of the asynchronous method call, either waiting until completion or until the specified time expires.

The ‘isCancelled’ and ‘isDone’ methods allow the client to check on the status of the asynchronous request.

And, the ‘cancel’ method allows the client to attempt to cancel an asynchronous request. The ‘cancel’ method will only return true if the asynchronous method could be cancelled before it was ever started; that is, while it was still in the queued state.

Asynchronous session bean invocations – configuration (1 of 2)



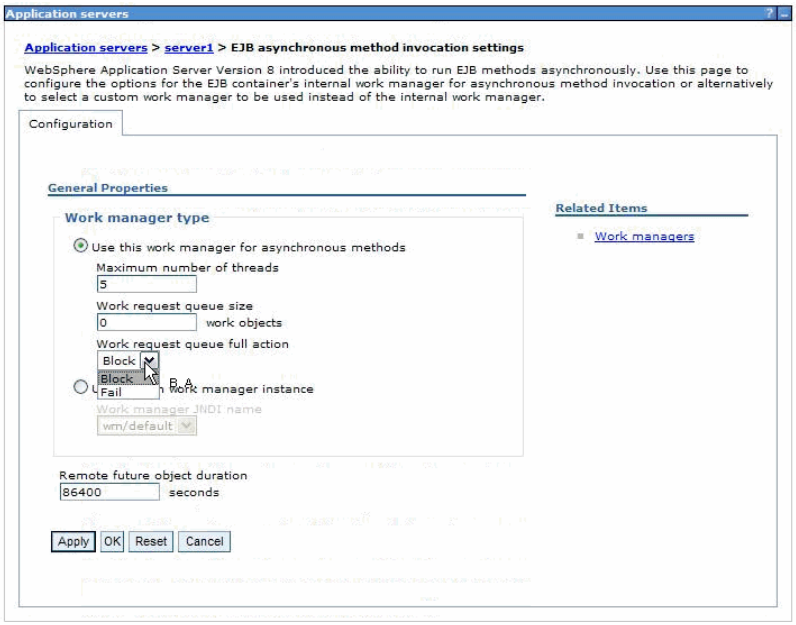
WebSphere Application Sever supports asynchronous session bean invocations by submitting each request to a work manager. A work manager controls a thread pool created for Java EE applications. A work manager will place all submitted work on a queue until a thread becomes available to run the request.

Although a default work manager configuration is provided, it is also possible to configure a custom work manager as shown in this screen capture.

When configuring a custom work manager, be aware that not all options available are compliant with the EJB specification behavior.

More information about work managers may be found in the WebSphere Application Server Information Center.

Asynchronous session bean invocations – configuration (2 of 2)



In addition to configuring the size of the thread pool and queue, another important configuration option is the queue full action.

By default, when all threads are active, and the queue is full, additional asynchronous requests will block. This means that control is not returned to the client immediately, but instead, the client thread will block until the request can be added to the queue.

The alternative is to specify the value ‘Fail’. When fail is specified, an exception is returned to the client indicating that the asynchronous request could not be submitted.

Singleton session beans (1 of 2)

- New session bean type
- Guaranteed single instance per JVM
- Supports eager initialization during application startup
- Allows for sharing of data across all the apps in the Server
  – avoid extra database trips
- Concurrency management
  @ConcurrencyManagement(BEAN)
  For example: public synchronized setProductInfo….

  @ConcurrencyManagement(CONTAINER)
  @Lock(LockType.*READ*)

  @Lock(LockType.*WRITE*)

Also new in EJB 3.1 are singleton session beans. These are a new type of session bean, in addition to stateless and stateful.

Singleton beans have many of the same characteristics of other session beans, such as declarative transaction management, security, remote interfaces, dependency injection, component life-cycle callbacks, and interceptors.

However, singleton beans introduce several new behaviors:

First, as the name suggests, it is guaranteed that only a single instance will exist per JVM.

Next, singleton beans support initialization during application startup and destruction during application shutdown, similar to the Startup beans that existed in prior versions of WebSphere Application Server. This feature allows a singleton to fully initialize application state before clients are allowed to access other EJBs.

And finally, singletons introduce the ability to configure concurrency management for EJB methods. Since there is only a single instance in each JVM, it is important that the bean implementation handle concurrent method calls. The bean implementation can rely on the EJB Container to provide this support by using container managed concurrency, and properly indicating which methods are read only, and which methods will change the state of the singleton instance.

Singleton session beans (2 of 2)

```
@Singleton
@LocalBean
@Startup
public class InventoryBean
{
   @Lock(LockType.READ)
    public int[] getInventory()  {…}

    @Lock(LockType.WRITE)
     public void setInventory()  {…}
}
```


Here is an example of how to configure a singleton session bean using annotations.

This singleton bean will have a No-Interface view that exposes two methods. One of the methods is read only, and may be entered concurrently on multiple threads, whereas the other method has been configured for write access, and will block until all other threads have exited methods on the bean instance.

Also note the use of the startup annotation. This indicates that the bean instance will be created, and the PostConstruct life cycle methods called at the end of module start.

Calendar based timer expressions

```
@Stateless
  public class mybean{

  @Resource
  private TimerService ts;

  public void doSomeTimerWork(String message, String minute)
  {
     TimerConfig tc = new TimerConfig(message, false);
   ScheduleExpression se = new ScheduleExpression();
   se.minute(minute);
  ts.createCalendarTimer(se, tc);
  }

  @Timeout
  private void doSomeTimeOutWork(timer timer)

    .. Stuff
  }
```


Calendar based EJB timer expressions is the first of several new enhancements to the EJB Timer Service.

The new calendar-based syntax is modeled after the UNIX cron facility and may be used for both programmatically and automatically created timers. And also both persistent and non-persistent timers.

This is an example of how to programmatically create a calendar based timer. In the example, a timer is created that runs at the specified minute, or minutes, depending on the contents of the ‘minute’ string.

Examples that demonstrate using calendar based timer expressions are provided later, when covering the new automatically created timers feature.

Automatic timer creation (1 of 2)

- Created automatically
- Can be created using annotation or XML
- Created or started when application first started
- Removed when application is uninstalled

Since EJB timers were first introduced in the EJB specification, the only way to create them was programmatically through the Timer Service API. Now, an EJB may be configured so that the timers associated with it are automatically created during application install.

These new automatically created timers may be configured through either annotations or in the EJB deployment descriptor. Once created, the timers will exist until the application is uninstalled.

Automatic timer creation (2 of 2)

```
// Generate account statements at 1 a.m. on the 1st of every month
@Schedule (hour="1", dayOfMonth="1", info="AccountStatementTimer")
public void generateMonthlyAccountStatements(Timer t) {
String timerInfo = t.getInfo();
```


The Schedule annotation is used to configure automatically created timers.

In this example, the schedule annotation has been used to configure a timer that will run the 'generateMonthlyAccountStatements' method at 1 a.m. on the 1st of every month. This is also another example of a calendar based timer.

Non-persistent Timers

- Ability to declare non persistent timers.
- Applies to automatically and programmatically created timers

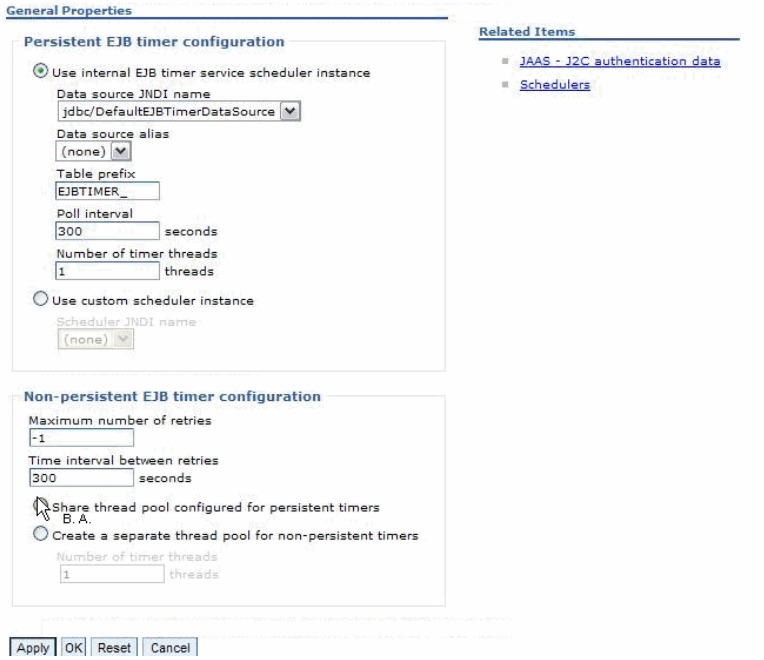
```
@Singleton
public class CacheBean {
Cache cache;
// Setup an automatic timer to refresh
// the Singleton instance cache every 10 minutes
@Schedule(minute="*/10", hour="*", persistent=false)
public void refresh()
{
   // ... Code to refresh the cache.
}
}
```


Another new timer enhancement is the ability to define non-persistent timers. Unlike persistent timers, non-persistent timers exist only in memory and are removed when the application is stopped.

Both automatically and programmatically created timers can be created as non-persistent timers.

This example shows how a calendar based automatically created timer is defined as non-persistent.

Timers configuration



The configuration options for persistent timers remain the same as in the past. A scheduler is used to manage the timer tasks, backed by a database, where both the scheduler and database can be configured.

Non-persistent timers require a simpler configuration. A database is not needed, and a scheduler is not used. Instead, all that is needed is a basic work manager and thread pool. Optionally, both persistent and non-persistent timers can be configured to use the same thread pool, and will thus run with the same priority, and potentially consume fewer resources.

In place of the scheduler, all that is needed are the two settings that control how frequently to retry failed timers, and how many times to retry a failed timer before discarding. The defaults are 300 seconds between retries, and to retry until successful. By default, a non-persistent timer is not discarded until it has either succeeded or the application is stopped.

Another ease of use enhancement for the EJB programming model in general, is the introduction of an ‘Embeddable EJB Container’.

This new feature is targeted at developers, to allow for an easy way to unit test EJB business logic in a J2SE environment. Client code is able to instantiate an EJB container that runs within the same class loader and JVM.

In WebSphere Application Server V8, the provided embeddable EJB container supports only the subset of EJB features defined as ‘EJB Lite’.

Embeddable EJB container – EJB Lite (1 of 2)

Feature	EJB Lite	EJB
Stateless beans	✓	✓
Stateful beans	✓	✓
Singleton beans	✓	✓
Message driven beans		✓
No interfaces	✓	✓
Local interfaces	✓	✓
Remote interfaces		✓
Web service interfaces		✓
Asynchronous invocation		✓

The embeddable EJB Container does not support all feature of the EJB specification, but rather a subset that is called ‘EJB Lite’. EJB Lite is a small, yet powerful subset of the full EJB API set, suitable for writing many portable applications.

For reference, here are the main differences between the full EJB feature set and EJB Lite.

Embeddable EJB container – EJB Lite (2 of 2)

	EJB Lite	EJB
Interceptors	✓	✓
Programmatic transactions	✓	✓
Declarative transactions	✓	✓
Declarative security	✓	✓
Timer service		✓
CORBA interoperability		✓
EJB 2.x support		✓

Here are the remaining differences.

Embeddable EJB container (1 of 3)


```
public static EJBContainer getEJBContainer()
{
    EJBContainer ec = EJBContainer.createEJBContainer(getEmbeddableEJBContainerProps());
    return ec;
}

public static Map<String,Object> getEmbeddableEJBContainerProps()
{
    Map<String,Object> properties = new HashMap<String,Object>();
    properties.put(EJBContainer.PROVIDER, "com.ibm.websphere.ejbcontainer.EmbeddableContainerProvider")
    properties.put("DataSource.ds1.name", "jdbc/orderds");
    properties.put("DataSource.ds1.className","com.ibm.db2.jcc.DB2DataSource");
    properties.put("DataSource.ds1.password", "wasledu2");
    properties.put("DataSource.ds1.user", "db2inst1");
    properties.put("DataSource.ds1.driverType", "4");
    properties.put("DataSource.ds1.serverName", "localhost");
    properties.put("DataSource.ds1.portNumber", "60000");
    properties.put("DataSource.ds1.databaseName", "ORDERDB");
    properties.put("role.SecureShopper", "rbarcia, kbrown");
    properties.put("role.runAs.manager", "rbarcia");
    properties.put("user.invocation", "rbarcia");
    properties.put("com.ibm.websphere.securityEnabled", "true");
    return properties;
}
```


Here is an example demonstrating how a test environment might create an instance of an embeddable EJB Container using a set of properties to configure a data source and a security role.

Embeddable EJB container (2 of 3)

```
private EJBContainer ec;

public void setUp() throws Exception
{
    super.setUp();
    ec = TestUtil.getEJBContainer();
    customerOrderServices = (CustomerOrderServices)
        ec.getContext().lookup("java:global/CustomerOrderServices/CustomerOrderServicesImpl");
}


protected void tearDown() throws Exception {
    super.tearDown();
    ec.close();
};
```


This is a continuation of the previous example, showing how the embeddable EJB container might be used in a test environment.

Embeddable EJB container (3 of 3)

```
public class EmbeddableContainerSample {

    public static void main(String[] args) throws Throwable
    {
        //Create a properties map to pass to the embeddable container:
        Map<String,String> properties = new HashMap<String,String>();
        // Specify that you want to use the WebSphere embeddable container:
        properties.put(EJBContainer.PROVIDER, "com.ibm.websphere.ejbcontainer.EmbeddableContainer");
        properties.put(EJBContainer.APP_NAME, "myappname");
        properties.put(EJBContainer.MODULES, "MyEJBModule");
          // Create the container instance, passing it our properties map:
        EJBContainer ec = EJBContainer.createEJBContainer(properties);
        MyBeanIface bean = (MyBeanIface) ec.getContext().lookup("java:global/MyEJBModule/MyBean!com.myCompany.MyBeanIface");
        // Invoke a method on the bean instance:
        bean.doStuff();
        //Close the embeddable container:
        ec.close();
    }
```


This is an example demonstrating use of the embeddable container from a class main method. This is useful for stand-alone testing.

See the WebSphere Application Server Information Center for additional information and examples of calling this sample from a command line. Note that the embeddable EJB container jar file and the EJB module need to be on the class path.

Embeddable EJB container – configuring a data source

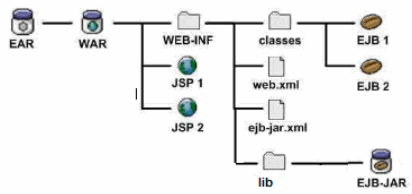
- The embeddable EJB container configuration properties are listed in the information center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.base.doc/info/aes/ae/rejb_emconproperties.html

- These properties cover configuring
  - A data source
  - Local transactions
  - Reference bindings
  - Security
  - XA

You can find a list of configuration properties in the information center link given here. The properties allow you to configure data sources, local transactions, bindings, security on the container, and XA distributed transactions.

Packaging of EJBs in WAR files



- Continue with simplicity concept.
- No longer required to package EJBs in jars.
- EJB can be defined using annotation in WEB-INF/classes.
- EJB can be defined in WEB-INF/ejb-jar.xml
- A jar can exist in a WAR under WEB-INF/lib (part of war module)
- Except for entity beans, EJB version 1 and version 2 content is supported in a WAR module.

The final EJB 3.1 enhancement being covered is the ability to package EJBs in WAR files.

This new feature continues to build on the EJB 3.0 enhancements for improved ease of use.

Since many applications only access EJBs from servlets, it only makes sense to allow the EJBs to be packaged with the servlet code that will be accessing them.

For convenience, the EJBs may be packaged in either the WEB-INF/classes directory or in a standard EJB jar file packaged in the WEB-INF/lib directory. Also, the EJB configuration information may be provided through either annotations or an EJB deployment descriptor placed in the WEB-INF directory.

Except for the older EJB version 1 and version 2 entity beans, all other EJB features are supported in a WAR module.

Following is a summary of the EJB 3.1 features covered.

Summary

- Overview of these new EJB 3.1 functions:
  - The no-interface local view
  - Embeddable EJB container
  - Asynchronous session bean invocation
  - Singleton session bean type
  - Calendar-based timer expressions
  - Automatic timer creation
  - Non-persistent timers
  - Packaging of EJBs in .WAR files

In summary, even though this was just a point revision to the EJB specification, many significant improvements and features have been added.

References

- EJB 3.1 specification

http://jcp.org/aboutJava/communityprocess/final/jsr318/index.html

- WebSphere Application Server information center

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.base.doc/info/aes/ae/welcome_base.html

For further information about these EJB 3.1 topics, see the EJB 3.1 Specification or the WebSphere Application Server Information Center.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASV8_EJB31.ppt

This module is also available in PDF format at: ../WASV8_EJB31.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information