



pointing the way

# White Papers

EJB

Components



## Introduction

This paper describes Enterprise JavaBeans 3.0 components.

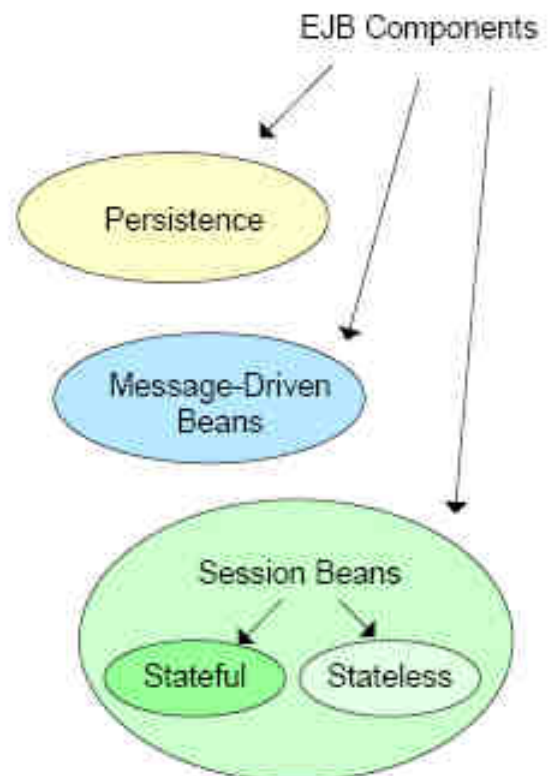
## Background

Enterprise applications process information and execute business logic using many different mechanisms. Depending on actual step taken in application following actions occur:

- ✓ Request/command application to take some actions
- ✓ Manage persistence data,
- ✓ Communicate with other systems.

To handle all types of enterprise application actions EJB standard introduced three major types of components: persistence, session and message driven.

Session beans divide into stateful and stateless giving even four JavaBeans types that have own lifecycle and behavior.



# EJB – Components



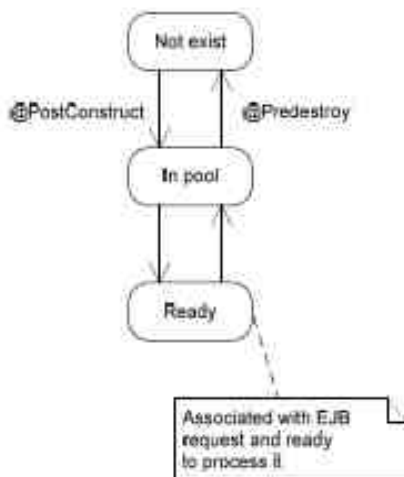
## EJB Components

### Stateless Session Beans

Stateless session beans are managed by EJB Container that instantiates beans and manage a pool.

EJB Container is a main part of EJB architecture. It defines an executable environment for EJB components; it integrates enterprise services and mechanisms.

All stateless session beans are equal – they are thread by EJB Container the same. Bean instances are randomly chosen to process business method calls. One application client can invoke different bean every method call.



Life cycle of stateless session bean. There are two callback events that can be intercepted in bean by listener's methods.

Stateless beans are used to process transaction in one procedure call/client request.

### Stateful Session Beans

Stateful session beans have a conversation state that represents a conversation with client application. This type of beans is managed by EJB

Container but instances are not stored in pool. To improve performance EJB Container implements an activation mechanism that can serialize stateful bean to save computer resources and later activate session bean creating new instance based on previously saved state.

Passivation is a process of detaching stateful bean and storing its state in cache. During passivation EJB specification requires keeping references to:

- ✓ other EJB components
- ✓ JNDI context
- ✓ EntityManager and EntityManagerFactory
- ✓ UserTransaction
- ✓ SessionContext
- ✓ Manageable object's factories (e.g. DataSource)

Activation is a process of restoring a stateful bean from cache and attaching it to client's session that is tracked in EJB Container.

Session stateful bean A can have injected another stateful bean B (using @EJB annotation). In this case EJB Container creates a new session for injected bean B which is owned by bean A.

### Interfaces for Session Beans

Session beans can have three types of interfaces:

- ✓ Remote, for methods that can be accessible outside EJB Container (by client's application)
- ✓ Local, for methods accessible inside EJB Container,
- ✓ Endpoint, for methods that map to Web Service interface created for given session bean.

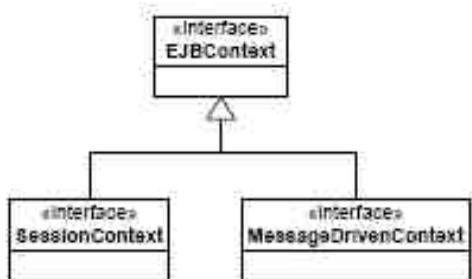
Session beans can include local/remote interfaces in three different ways:

# EJB – Components



- ✓ Implementing interfaces annotated with @Local, @Remote,
- ✓ Using annotations @Local(LocalInterface.class) and/or @Remote(RemoteInterface1.class) before session bean class declaration
- ✓ Writing XML descriptor with elements <remote> and/or <local>.

EJB Container provides implementation for javax.ejb.EJBContext interface. Every session bean and MDB has to declare EJBContext interface to be manageable by EJB Container.



Session bean declares descendant interface javax.ejb.SessionContext. Access to this interface is possible by following annotation: @Resource SessionContext ctx;

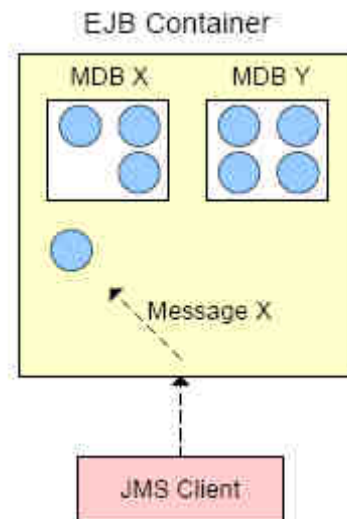
SessionContext provides information and references to:

- ✓ current user accessing given bean and its roles
- ✓ searching entities in ENC
- ✓ reference to itself (that is required to pass given object to other beans)
- ✓ interface used to invoke given method (remote/local)
- ✓ reference to TimeService.

## Message-Driven Beans

MDB are stateless and are managed by EJB Container using pool. MDB component register type of messages and addresses that they are

interested with. When asynchronous message is sent by JMS client it goes to EJB Container that decides to which JMS-MDB component should handle request.



MDB related to message type is pulled from pool to handle request.

## Persistence/Entities

Previous EJB specifications define a Container-Managed Persistence (CMP) and Bean-Managed Persistence (BMP). EJB 3.0 specification uses more independent persistence mechanism Java Persistence API (JPA). This new API is designed to work inside and outside Java Enterprise Edition (Java EE) container.

### Entity

Entity is a representation of an atomic data in Relational World. This term refers also to Plain, Old Java Object (POJO) with annotations. Entities are managed in EJB Container by Entity Manager.

# EJB – Components

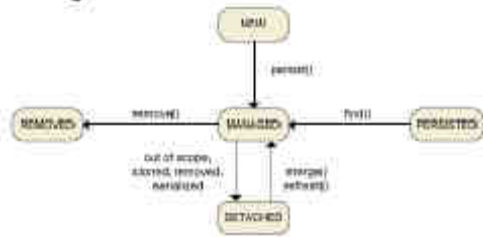


Entity Manager is a core part of Java Persistence API and it is responsible for managing entities including:

- ✓ Mapping between OO and Relational World,
- ✓ Entity Lifecycle (persist, merge, retrieve),
- ✓ Interacting with persistence store (database).

## Life-cycle

Entity Manager synchronized entities state with database. It means that if application creates new entity or removes one, Manager insert or delete row in given table.



**Entity Manager API methods can change actual entity state.**

Changes to the entity's data are reflected in database by periodically check for data freshness as long as entity is managed. Detached entities are not managed, so they do not reflect to data in DB.

## Annotations

There are only two annotations needed to transform a POJO into an entity:

- ✓ `@Entity`, before class definition,
- ✓ `@Id`, before method or attribute definition that will represent a primary key.

EntityManager will assume that class attributes (including types) names and class name map to table and table's columns in database.

There are two others annotations that with their properties can be used to define specific mapping: `@Table`, `@Column`.

## Primary Key

Primary key is annotated with `@Id`. It should be unique, so to provide key generation EJB serves `@GeneratedValue` annotation. There are four strategies of generating process:

- ✓ **IDENTITY**, it forces using column type designed for primary keys (and which is available in given database implementation)
- ✓ **TABLE**, points to database generator table with two columns (primary key column, value column); this table stores a sequence number and mapping to it should be defined using `@TableGenerator` annotation
- ✓ **SEQUENCE**, points to sequence defined in database; mapping to sequence should be defined using `@SequenceGenerator`,
- ✓ **AUTO**, strategy is chosen by persistence mechanism.

## Callback Events

It is possible to make stateful session bean aware of life-cycle processes by using annotations. For example method annotated with `@javax.ejb.PostActivate` is invoked after successful process of bean activation. Method annotated with `@javax.ejb.PrePassivate` is invoked before passivation process.

For entities exist a bunch of callback events annotations from `javax.persistence` package such as: `PrePersist`, `PostLoad`, `PostRemove`, etc.

## Interceptors

Interceptors are designed to implement objects behavior in more efficient way. They provide separation of behavior code and business logic. Interceptor is a typical java class that has methods annotated with `@javax.interceptor.AroundInvoke`. Interceptor's method has a full control of what happens before and after business method invocation or if given method is even invoked or

# EJB – Components



not. Business method parameters and result can be freely modified by interceptor.

Interceptors can be injected into beans by using annotations or in XML descriptor. The easiest way is to add:

`@javax.interceptor.Interceptors(Intercl.class)`  
annotation:

- ✓ before business class definition,
- ✓ or before class definition (then interceptor will apply to all methods in given class).

Additionally according to EJB specification interceptors allow intercepting life-cycle events' methods such as PrePersist, PostActivation, etc.

## Conclusion

EJB 3.0 brings a simpler model of creating Java Enterprise Applications than previous EJB releases. Creating session beans with their interfaces is now much easier than ever before.

Moving persistence to separated specification (JPA) that can work inside and outside container

and where entities can be detached and sent to client side makes persistence development simpler and intuitive. EJB 3.0 supports Java 5 annotations feature that let developers writing code faster with better understanding. There is no reason to stick with old EJB releases when version 3.0 is available!

## Links

Bill Burke, Richard Monson-Haefel – “Enterprise JavaBeans 3.0”, O'Reilly 2007

Enterprise JavaBeans Technology, EJB 3.0 Specification

<http://java.sun.com/products/ejb/>

Simplifying EJB Development with EJB 3.0

[http://www.oracle.com/technology/tech/java/newsletter/articles/simplifying\\_ejb3.html](http://www.oracle.com/technology/tech/java/newsletter/articles/simplifying_ejb3.html)

EJB 3 Session Beans

<http://www.developer.com/java/ejb/article.php/3650661>