**Jakarta Enterprise Beans** (**EJB**; formerly Enterprise JavaBeans) is one of several [Java APIs](https://en.wikipedia.org/wiki/List_of_Java_APIs) for modular construction of [enterprise software](https://en.wikipedia.org/wiki/Enterprise_software). EJB is a [server-side](https://en.wikipedia.org/wiki/Server-side) [software component](https://en.wikipedia.org/wiki/Component_(software)) that [encapsulates](https://en.wikipedia.org/wiki/Encapsulation_(object-oriented_programming)) [business logic](https://en.wikipedia.org/wiki/Business_logic) of an application. An EJB [web container](https://en.wikipedia.org/wiki/Web_container) provides a [runtime environment](https://en.wikipedia.org/wiki/Runtime_environment) for web related software components, including [computer security](https://en.wikipedia.org/wiki/Computer_security), [Java servlet lifecycle management](https://en.wikipedia.org/wiki/Java_Servlet#Life_cycle_of_a_servlet), [transaction processing](https://en.wikipedia.org/wiki/Transaction_processing), and other [web services](https://en.wikipedia.org/wiki/Web_services). The EJB specification is a subset of the [Java EE](https://en.wikipedia.org/wiki/Java_EE) specification

Specification[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=1)]

The EJB specification was originally developed in 1997 by [IBM](https://en.wikipedia.org/wiki/International_Business_Machines) and later adopted by [Sun Microsystems](https://en.wikipedia.org/wiki/Sun_Microsystems) (EJB 1.0 and 1.1) in 1999[[2]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-2) and enhanced under the [Java Community Process](https://en.wikipedia.org/wiki/Java_Community_Process) as [JSR 19](http://www.jcp.org/en/jsr/detail?id=19) (EJB 2.0), [JSR 153](http://www.jcp.org/en/jsr/detail?id=153) (EJB 2.1), [JSR 220](http://www.jcp.org/en/jsr/detail?id=220) (EJB 3.0), [JSR 318](http://jcp.org/en/jsr/detail?id=318) (EJB 3.1) and [JSR 345](http://jcp.org/en/jsr/detail?id=345) (EJB 3.2).

The EJB specification provides a standard way to implement the server-side (also called "[back-end](https://en.wikipedia.org/wiki/Front_and_back_ends)") 'business' software typically found in enterprise applications (as opposed to 'front-end' [user interface](https://en.wikipedia.org/wiki/User_interface) software). Such software addresses the same types of problem, and solutions to these problems are often repeatedly re-implemented by programmers. Jakarta Enterprise Beans is intended to handle such common concerns as [persistence](https://en.wikipedia.org/wiki/Persistence_(computer_science)), [transactional integrity](https://en.wikipedia.org/wiki/Data_integrity) and [security](https://en.wikipedia.org/wiki/Computer_security) in a standard way, leaving programmers free to concentrate on the particular parts of the enterprise software at hand

## General responsibilities[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=2)]

The EJB specification details how an [application server](https://en.wikipedia.org/wiki/Application_server) provides the following responsibilities:

* [Transaction processing](https://en.wikipedia.org/wiki/Transaction_processing)
* Integration with the [persistence](https://en.wikipedia.org/wiki/Persistence_(computer_science)) services offered by the [Java Persistence API (JPA)](https://en.wikipedia.org/wiki/Java_Persistence_API)
* [Concurrency control](https://en.wikipedia.org/wiki/Concurrency_control)
* [Event-driven programming](https://en.wikipedia.org/wiki/Event-driven_programming) using [Java Message Service](https://en.wikipedia.org/wiki/Java_Message_Service) and [Java EE Connector Architecture](https://en.wikipedia.org/wiki/Java_EE_Connector_Architecture)
* [Asynchronous method invocation](https://en.wikipedia.org/wiki/Asynchronous_method_invocation)
* [Job scheduling](https://en.wikipedia.org/wiki/Job_scheduler)
* Naming and [directory services](https://en.wikipedia.org/wiki/Directory_service) ([JNDI](https://en.wikipedia.org/wiki/Java_Naming_and_Directory_Interface))
* [Interprocess Communication](https://en.wikipedia.org/wiki/Remote_procedure_call) using [RMI-IIOP](https://en.wikipedia.org/wiki/RMI-IIOP) and [Web services](https://en.wikipedia.org/wiki/Web_service)
* [Security](https://en.wikipedia.org/wiki/Computer_security) ([JCE](https://en.wikipedia.org/wiki/Java_Cryptography_Extension) and [JAAS](https://en.wikipedia.org/wiki/Java_Authentication_and_Authorization_Service))
* [Deployment](https://en.wikipedia.org/wiki/Software_deployment) of [software components](https://en.wikipedia.org/wiki/Software_component) in an application server

Additionally, the Enterprise JavaBean specification defines the roles played by the EJB container and the EJBs as well as how to deploy the EJBs in a container. Note that the current EJB 3.2 specification does not detail how an application server provides persistence (a task delegated to the JPA specification), but instead details how business logic can easily integrate with the persistence services offered by the application server.

## History[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=3)]

Businesses found that using EJBs to encapsulate business logic brought a performance penalty. This is because the original specification allowed only for remote method invocation through [CORBA](https://en.wikipedia.org/wiki/CORBA) (and optionally other protocols), even though the large majority of business applications actually do not require this [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing) functionality. The EJB 2.0 specification addressed this concern by adding the concept of local interfaces which could be called directly without performance penalties by applications that were not distributed over multiple servers.[[3]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-3)

The EJB 3.0 specification ([JSR](https://en.wikipedia.org/wiki/Java_Community_Process) 220) was a departure from its predecessors, following a new light-weight paradigm. EJB 3.0 shows an influence from [Spring](https://en.wikipedia.org/wiki/Spring_Framework) in its use of plain Java objects, and its support for [dependency injection](https://en.wikipedia.org/wiki/Dependency_injection) to simplify configuration and integration of heterogeneous systems. Gavin King, the creator of Hibernate, participated in the EJB 3.0 process and is an outspoken advocate of the technology. Many features originally in Hibernate were incorporated in the [Java Persistence API](https://en.wikipedia.org/wiki/Java_Persistence_API), the replacement for [entity beans](https://en.wikipedia.org/wiki/Entity_Bean) in EJB 3.0. The EJB 3.0 specification relies heavily on the use of [annotations](https://en.wikipedia.org/wiki/Java_annotation) (a feature added to the Java language with its 5.0 release) and [convention over configuration](https://en.wikipedia.org/wiki/Convention_over_configuration) to enable a much less verbose coding style. Accordingly, in practical terms EJB 3.0 is much more lightweight and nearly a completely new API, bearing little resemblance to the previous EJB specifications

## Example[[**edit**](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=4)**]**

The following shows a basic example of what an EJB looks like in code:

@Stateless

public class CustomerService {

private EntityManager entityManager;

public void addCustomer(Customer customer) {

entityManager.persist(customer);

}

}

The above defines a service class for persisting a Customer object (via [O/R mapping](https://en.wikipedia.org/wiki/Object-relational_mapping)). The EJB takes care of managing the persistence context and the addCustomer() method is transactional and thread-safe by default. As demonstrated, the EJB focuses only on business logic and persistence and knows nothing about any particular presentation.

Such an EJB can be used by a class in e.g. the web layer as follows:

@Named

@RequestScoped

public class CustomerBacking {

@EJB

private CustomerService customerService;

public String addCustomer(Customer customer) {

customerService.addCustomer(customer);

context.addMessage(...); *// abbreviated for brevity*

return "customer\_overview";

}

}

The above defines a [JavaServer Faces](https://en.wikipedia.org/wiki/JavaServer_Faces) (JSF) backing bean in which the EJB is injected by means of the @EJB annotation. Its addCustomer method is typically bound to some UI component, such as a button. Contrary to the EJB, the backing bean does not contain any business logic or persistence code, but delegates such concerns to the EJB. The backing bean does know about a particular presentation, of which the EJB had no knowledge.

## **Types of Enterprise Beans[**[**edit**](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=5)**]**

An EJB container holds two major types of beans:

* Session Beans[[4]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-4) that can be either "Stateful", "Stateless" or "Singleton" and can be accessed via either a **Local** (same JVM) or **Remote** (different JVM) interface or directly without an interface,[[5]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-5) in which case local semantics apply. All session beans support asynchronous execution[[6]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-6) for all views (local/remote/no-interface).
* Message Driven Beans (MDBs, also known as Message Beans). MDBs also support asynchronous execution, but via a messaging paradigm.

### Session beans[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=6)]

#### Stateful Session Beans[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=7)]

*See also:*[*Conversational state (Java EE)*](https://en.wikipedia.org/wiki/Conversational_state_(Java_EE))

Stateful Session Beans[[7]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-7) are business objects having [state](https://en.wikipedia.org/wiki/State_(computer_science)): that is, they keep track of which calling client they are dealing with throughout a session and thus access to the bean instance is strictly limited to only one client at a time.[[8]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-autogenerated1-8) If concurrent access to a single bean is attempted anyway the container serializes those requests, but via the @AccessTimeout annotation the container can instead throw an exception.[[9]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-9) Stateful session beans' state may be persisted (passivated) automatically by the container to free up memory after the client hasn't accessed the bean for some time. The JPA extended persistence context is explicitly supported by Stateful Session Beans.[[10]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-10)

***Examples***

* Checking out in a web store might be handled by a stateful session bean that would use its state to keep track of where the customer is in the checkout process, possibly holding locks on the items the customer is purchasing (from a system architecture's point of view, it would be less ideal to have the client manage those locks).

#### Stateless Session Beans[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=8)]

Stateless Session Beans[[11]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-11) are business objects that do not have state associated with them. However, access to a single bean instance is still limited to only one client at a time, [concurrent](https://en.wikipedia.org/wiki/Concurrent_computing) access to the bean is prohibited.[[8]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-autogenerated1-8) If concurrent access to a single bean is attempted, the container simply routes each request to a different instance.[[12]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-12) This makes a stateless session bean automatically thread-safe. Instance variables can be used during a single method call from a client to the bean, but the contents of those instance variables are not guaranteed to be preserved across different client [method](https://en.wikipedia.org/wiki/Method_(computer_science)) calls. Instances of Stateless Session beans are typically pooled. If a second client accesses a specific bean right after a method call on it made by a first client has finished, it might get the same instance. The lack of overhead to maintain a conversation with the calling client makes them less resource-intensive than stateful beans.

***Examples***

* Sending an e-mail to customer support might be handled by a stateless bean, since this is a one-off operation and not part of a multi-step process.
* A user of a website clicking on a "keep me informed of future updates" box may trigger a call to an asynchronous method of the session bean to add the user to a list in the company's database (this call is asynchronous because the user does not need to wait to be informed of its success or failure).
* Fetching multiple independent pieces of data for a website, like a list of products and the history of the current user might be handled by asynchronous methods of a session bean as well (these calls are asynchronous because they can execute in [parallel](https://en.wikipedia.org/wiki/Parallel_computing) that way, which potentially increases performance). In this case, the asynchronous method will return a [Future](http://java.sun.com/javase/6/docs/api/java/util/concurrent/Future.html) instance.

#### Singleton Session Beans[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=9)]

Singleton Session Beans[[13]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-13)[[14]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-14) are business objects having a global shared state within a JVM. Concurrent access to the one and only bean instance can be controlled by the container (Container-managed concurrency, CMC) or by the bean itself (Bean-managed concurrency, BMC). CMC can be tuned using the @Lock annotation, that designates whether a read lock or a write lock will be used for a method call. Additionally, Singleton Session Beans can explicitly request to be instantiated when the EJB container starts up, using the @Startup annotation.

***Examples***

* Loading a global daily price list that will be the same for every user might be done with a singleton session bean, since this will prevent the application having to do the same query to a database over and over again...

### Message driven beans[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=10)]

**Message Driven Beans**[[15]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-15) are business objects whose execution is triggered by messages instead of by method calls. The Message Driven Bean is used among others to provide a high level ease-of-use abstraction for the lower level JMS ([Java Message Service](https://en.wikipedia.org/wiki/Java_Message_Service)) specification. It may subscribe to JMS message queues or message topics, which typically happens via the activationConfig attribute of the @MessageDriven annotation. They were added in EJB to allow event-driven processing. Unlike session beans, an MDB does not have a client view (Local/Remote/No-interface), i. e. clients cannot look-up an MDB instance. An MDB just listens for any incoming message on, for example, a JMS queue or topic and processes them automatically. Only JMS support is required by the Java EE spec,[[16]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-16) but Message Driven Beans can support other messaging protocols.[[17]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-17)[[18]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-18) Such protocols may be asynchronous but can also be synchronous. Since session beans can also be synchronous or asynchronous, the prime difference between session- and message driven beans is not the synchronicity, but the difference between (object oriented) [method](https://en.wikipedia.org/wiki/Method_(computer_science)) [calling](https://en.wikipedia.org/wiki/Calling_convention) and [messaging](https://en.wikipedia.org/wiki/Message_passing).

***Examples***

* Sending a configuration update to multiple nodes might be done by sending a JMS message to a 'message topic' and could be handled by a Message Driven Bean listening to this topic (the message paradigm is used here since the sender does not need to know the number of consumers, their location, or even their exact type).
* Submitting a job to a work cluster might be done by sending a JMS message to a 'message queue' and could also be handled by a Message Driven Bean, but this time listening to a queue (the message paradigm and the queue is used, since the sender doesn't have to care which worker executes the job, but it does need assurance that a job is only executed once).
* Processing timing events from the [Quartz scheduler](http://www.quartz-scheduler.org/) can be handled by a Message Driven Bean; when a Quartz *trigger* fires, the MDB is automatically invoked. Since Java EE doesn't know about Quartz by default, a [JCA](https://en.wikipedia.org/wiki/Java_EE_Connector_Architecture) resource adapter would be needed and the MDB would be annotated with a reference to this.[[19]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-19)

## **Execution[**[**edit**](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=11)**]**

EJBs are deployed in an EJB container, typically within an [application server](https://en.wikipedia.org/wiki/Application_server). The specification describes how an EJB interacts with its container and how client code interacts with the container/EJB combination. The EJB classes used by applications are included in the [javax.ejb](https://javaee.github.io/javaee-spec/javadocs/javax/ejb/package-summary.html) package. (The [javax.ejb.spi](https://javaee.github.io/javaee-spec/javadocs/javax/ejb/spi/package-summary.html) package is a [service provider interface](https://en.wikipedia.org/wiki/Service_provider_interface) used only by EJB container implementations.)

Clients of EJBs do not instantiate those beans directly via Java's new operator, but instead have to obtain a reference via the EJB container. This reference is usually not a reference to the implementation bean itself, but to a [proxy](https://en.wikipedia.org/wiki/Proxy_pattern), which either dynamically implements the local or remote business interface that the client requested or dynamically implements a sub-type of the actual bean. The proxy can then be directly cast to the interface or bean. A client is said to have a 'view' on the EJB, and the local interface, remote interface and bean type itself respectively correspond with the local view, remote view and no-interface view.

This proxy is needed in order to give the EJB container the opportunity to transparently provide cross-cutting ([AOP](https://en.wikipedia.org/wiki/Aspect-oriented_programming)-like) services to a bean like transactions, security, interceptions, injections, and remoting. As an example, a client invokes a method on a proxy, which will first start a transaction with the help of the EJB container and then call the actual bean method. When the bean method returns, the proxy ends the transaction (i.e. by committing it or doing a rollback) and transfers control back to the client.

The EJB Container is responsible for ensuring the client code has sufficient access rights to an EJB.[[20]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-20) Security aspects can be declaratively applied to an EJB via annotations.[[21]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-21)

### Transactions[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=12)]

EJB containers must support both container managed [ACID](https://en.wikipedia.org/wiki/ACID) transactions and bean managed transactions.[[22]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-22)

Container-managed transactions (CMT) are by default active for calls to session beans. That is, no explicit configuration is needed. This behavior may be declaratively tuned by the bean via annotations and if needed such configuration can later be overridden in the deployment descriptor. Tuning includes switching off transactions for the whole bean or specific methods, or requesting alternative strategies for transaction propagation and starting or joining a transaction. Such strategies mainly deal with what should happen if a transaction is or isn't already in progress at the time the bean is called. The following variations are supported:[[23]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-23)[[24]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-24)

|  |  |
| --- | --- |
| **Declarative Transactions Management Types** | |
| **Type** | **Explanation** |
| MANDATORY | If the client has not started a transaction, an exception is thrown. Otherwise the client's transaction is used. |
| REQUIRED | If the client has started a transaction, it is used. Otherwise a new transaction is started. (this is the default when no explicit type has been specified) |
| REQUIRES\_NEW | If the client has started a transaction, it is suspended. A new transaction is always started. |
| SUPPORTS | If the client has started a transaction, it is used. Otherwise, no transaction is used. |
| NOT\_SUPPORTED | If the client has started a transaction, it is suspended. No new transaction is started. |
| NEVER | If the client has started a transaction, an exception is thrown. No new transaction is started. |

Alternatively, the bean can also declare via an annotation that it wants to handle transactions programmatically via the [JTA](https://en.wikipedia.org/wiki/Java_Transaction_API) API. This mode of operation is called Bean Managed Transactions (BMT), since the bean itself handles the transaction instead of the container.[[25]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-25)

### Events[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=13)]

JMS ([Java Message Service](https://en.wikipedia.org/wiki/Java_Message_Service)) is used to send messages from beans to clients, to let clients receive asynchronous messages from these beans. MDBs can be used to receive messages from clients asynchronously using either a **JMS** Queue or a Topic.

### Naming and directory services[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=14)]

As an alternative to injection, clients of an EJB can obtain a reference to the session bean's proxy object (the EJB stub) using [Java Naming and Directory Interface (JNDI)](https://en.wikipedia.org/wiki/JNDI). This alternative can be used in cases where injection is not available, such as in non-managed code or standalone remote Java SE clients, or when it's necessary to programmatically determine which bean to obtain.

JNDI names for EJB session beans are assigned by the EJB container via the following scheme:[[26]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-26)[[27]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-27)[[28]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-28)

|  |  |
| --- | --- |
| **JNDI names** | |
| **Scope** | **Name pattern** |
| Global | java:global[/<app-name>]/<module-name>/<bean-name>[!<fully-qualified-interface-name>] |
| Application | java:app/<module-name>/<bean-name>[!<fully-qualified-interface-name>] |
| Module | java:module/<bean-name>[!<fully-qualified-interface-name>] |

*(entries in square brackets denote optional parts)*

A single bean can be obtained by any name matching the above patterns, depending on the 'location' of the client. Clients in the same module as the required bean can use the module scope and larger scopes, clients in the same application as the required bean can use the app scope and higher, etc.

E.g. code running in the same module as the CustomerService bean *(as given by the example shown earlier in this article)* could use the following code to obtain a (local) reference to it:

CustomerServiceLocal customerService =

(CustomerServiceLocal) new InitialContext().lookup("java:module/CustomerService");

### Remoting/distributed execution[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=15)]

For communication with a client that's written in the Java programming language a session bean can expose a remote-view via an @Remote annotated interface.[[29]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-29) This allows those beans to be called from clients in other [JVMs](https://en.wikipedia.org/wiki/JVM) which themselves may be located on other (remote) systems. From the point of view of the EJB container, any code in another JVM is remote.

Stateless- and Singleton session beans may also expose a "web service client view" for remote communication via [WSDL](https://en.wikipedia.org/wiki/WSDL) and [SOAP](https://en.wikipedia.org/wiki/SOAP) or plain XML.[[30]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-30)[[31]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-31)[[32]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-32) This follows the [JAX-RPC](https://en.wikipedia.org/wiki/JAX-RPC) and [JAX-WS](https://en.wikipedia.org/wiki/JAX-WS) specifications. JAX-RPC support however is proposed for future removal.[[33]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-autogenerated2-33) To support JAX-WS, the session bean is annotated with the @WebService annotation, and methods that are to be exposed remotely with the @WebMethod annotation..

Although the EJB specification does not mention exposure as RESTful web services in any way and has no explicit support for this form of communication, the [JAX-RS](https://en.wikipedia.org/wiki/JAX-RS) specification does explicitly support EJB.[[34]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-34) Following the JAX-RS spec, Stateless- and Singleton session beans can be root resources via the @Path annotation and EJB business methods can be mapped to resource methods via the @GET, @PUT, @POST and @DELETE annotations. This however does not count as a "web service client view", which is used exclusively for JAX-WS and JAX-RPC.

Communication via web services is typical for clients not written in the Java programming language, but is also convenient for Java clients who have trouble reaching the EJB server via a firewall. Additionally, web service based communication can be used by Java clients to circumvent the arcane and ill-defined requirements for the so-called "client-libraries"; a set of jar files that a Java client must have on its class-path in order to communicate with the remote EJB server. These client-libraries potentially conflict with libraries the client may already have (for instance, if the client itself is also a full Java EE server) and such a conflict is deemed to be very hard or impossible to resolve.[[35]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-35)

### Legacy[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=16)]

#### Home interfaces and required business interface[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=17)]

*"Home interface" redirects here. For other uses, see*[*Home interface (disambiguation)*](https://en.wikipedia.org/w/index.php?title=Home_interface_(disambiguation)&action=edit&redlink=1)*.*

With EJB 2.1 and earlier, each EJB had to provide a Java implementation [class](https://en.wikipedia.org/wiki/Class_(computer_science)) and two Java interfaces. The EJB container created instances of the Java implementation class to provide the EJB implementation. The Java interfaces were used by client code of the EJB.

#### Required deployment descriptor[[edit](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=18)]

With EJB 2.1 and earlier, the EJB specification required a deployment descriptor to be present. This was needed to implement a mechanism that allowed EJBs to be [deployed](https://en.wikipedia.org/wiki/Software_deployment) in a consistent manner regardless of the specific EJB platform that was chosen. Information about how the bean should be deployed (such as the name of the home or remote interfaces, whether and how to store the bean in a database, etc.) had to be specified in the deployment descriptor.

The [deployment descriptor](https://en.wikipedia.org/wiki/Deployment_descriptor) is an [XML](https://en.wikipedia.org/wiki/XML) document having an entry for each EJB to be deployed. This XML document specifies the following information for each EJB:

* Name of the Home interface
* Java class for the Bean (business object)
* Java interface for the Home interface
* Java interface for the business object
* Persistent store (only for Entity Beans)
* Security roles and permissions
* Stateful or Stateless (for Session Beans)

Old EJB containers from many vendors required more deployment information than that in the EJB specification. They would require the additional information as separate XML files, or some other configuration file format. An EJB platform vendor generally provided their own tools that would read this deployment descriptor, and possibly generated a set of classes that would implement the now deprecated Home and Remote interfaces.

Since EJB 3.0 ([JSR 220](http://jcp.org/en/jsr/detail?id=220)), the XML descriptor is replaced by [Java annotations](https://en.wikipedia.org/wiki/Java_annotation) set in the Enterprise Bean implementation (at source level), although it is still possible to use an XML descriptor instead of (or in addition to) the annotations. If an XML descriptor and annotations are both applied to the same attribute within an Enterprise Bean, the XML definition overrides the corresponding source-level annotation, although some XML elements can also be additive (e.g., an activation-config-property in XML with a different name than already defined via an @ActivationConfigProperty annotation will be added instead of replacing all existing properties).

## **Container variations[**[**edit**](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=19)**]**

Starting with EJB 3.1, the EJB specification defines two variants of the EJB container; a full version and a limited version. The limited version adheres to a [proper subset](https://en.wikipedia.org/wiki/Proper_subset) of the specification called EJB 3.1 Lite [[36]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-36)[[37]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-37) and is part of [Java EE 6's web profile](https://en.wikipedia.org/wiki/Java_EE#Web_Profile) (which is itself a subset of the full Java EE 6 specification).

EJB 3.1 Lite excludes support for the following features:[[38]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-38)

* Remote interfaces
* RMI-IIOP Interoperability
* JAX-WS Web Service Endpoints
* EJB Timer Service (@Schedule, @Timeout)
* Asynchronous session bean invocations (@Asynchronous)
* Message-driven beans

EJB 3.2 Lite excludes less features. Particularly it no longer excludes @Asynchronous and @Schedule/@Timeout, but for @Schedule it does not support the "persistent" attribute that full EJB 3.2 does support. The complete excluded list for EJB 3.2 Lite is:

* Remote interfaces
* RMI-IIOP Interoperability
* JAX-WS Web Service Endpoints
* Persistent timers ("persistent" attribute on @Schedule)
* Message-driven beans

## **Version history[**[**edit**](https://en.wikipedia.org/w/index.php?title=Jakarta_Enterprise_Beans&action=edit&section=20)**]**

**EJB 3.2.6, final release (2019-08-23)**

Despite still using "EJB" the [Eclipse Foundation](https://en.wikipedia.org/wiki/Eclipse_Foundation) has official renamed these as Jakarta Enterprise Beans so as not to tread on the "Java" trademark.

**EJB 3.2, final release (2013-05-28)**

[JSR 345](http://jcp.org/en/jsr/detail?id=345). Enterprise JavaBeans 3.2 was a relatively minor release that mainly contained specification clarifications and lifted some restrictions that were imposed by the spec but over time appeared to serve no real purpose. A few existing full EJB features were also demanded to be in EJB 3 lite and functionality that was proposed to be pruned in EJB 3.1 was indeed pruned (made optional).[[39]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-39)[[40]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-40)

The following features were added:

* Passivation of a stateful session bean can be deactivated via attribute on @Stateful annotation (passivationCapable = false)
* TimerService can retrieve all active timers in the same EJB module (could previously only retrieve timers for the bean in which the TimerService was called)
* Lifecycle methods (e.g. @PostConstruct) can be transactional for stateful session beans using the existing @TransactionAttribute annotation
* Autocloseable interface implemented by embeddable container

**EJB 3.1, final release (2009-12-10)**

[JSR 318](http://jcp.org/en/jsr/detail?id=318). The purpose of the Enterprise JavaBeans 3.1 specification is to further simplify the EJB architecture by reducing its complexity from the developer's point of view, while also adding new functionality in response to the needs of the community:

* Local view without interface (No-interface view)
* [.war](https://en.wikipedia.org/wiki/WAR_file_format_(Sun)) packaging of EJB components
* EJB Lite: definition of a subset of EJB
* Portable EJB Global [JNDI](https://en.wikipedia.org/wiki/JNDI) Names
* [Singletons](https://en.wikipedia.org/wiki/Singleton_pattern) (Singleton Session Beans)
* Application Initialization and Shutdown Events
* EJB Timer Service Enhancements
* Simple [Asynchrony](https://en.wiktionary.org/wiki/asynchrony) (@Asynchronous for session beans)

**EJB 3.0, final release (2006-05-11)**

[JSR 220](http://www.jcp.org/en/jsr/detail?id=220) - *Major changes*: This release made it much easier to write EJBs, using 'annotations' rather than the complex 'deployment descriptors' used in version 2.x. The use of home and remote interfaces and the ejb-jar.xml file were also no longer required in this release, having been replaced with a business interface and a bean that implements the interface.

**EJB 2.1, final release (2003-11-24)**

[JSR 153](http://www.jcp.org/en/jsr/detail?id=153) - *Major changes*:

* [Web service](https://en.wikipedia.org/wiki/Web_service) support (new): stateless session beans can be invoked over [SOAP](https://en.wikipedia.org/wiki/SOAP)/[HTTP](https://en.wikipedia.org/wiki/HTTP). Also, an EJB can easily access a Web service using the new service reference.
* EJB timer service (new): Event-based mechanism for invoking EJBs at specific times.
* Message-driven beans accepts messages from sources other than [JMS](https://en.wikipedia.org/wiki/Java_Message_Service).
* Message destinations (the same idea as EJB references, resource references, etc.) has been added.
* EJB query language (EJB-QL) additions: ORDER BY, AVG, MIN, MAX, SUM, COUNT, and MOD.
* [XML schema](https://en.wikipedia.org/wiki/XML_Schema_(W3C)) is used to specify deployment descriptors, replaces [DTDs](https://en.wikipedia.org/wiki/Document_Type_Definition)

**EJB 2.0, final release (2001-08-22)**

[JSR 19](http://www.jcp.org/en/jsr/detail?id=19) - *Major changes*: *Overall goals*:

* The standard component architecture for building *distributed* object-oriented business applications in [Java](https://en.wikipedia.org/wiki/Java_(programming_language)).
* Make it possible to build distributed applications by combining components developed using tools from *different vendors*.
* Make it easy to write (enterprise) applications: Application developers will not have to understand low-level transaction and state management details, multi-threading, connection pooling, and other complex low-level APIs.
* Will follow the "Write Once, Run Anywhere" philosophy of [Java](https://en.wikipedia.org/wiki/Java_(programming_language)). An enterprise Bean can be developed once, and then deployed on multiple platforms without recompilation or source code modification.
* Address the development, deployment, and runtime aspects of an enterprise application’s life cycle.
* Define the contracts that enable tools from multiple vendors to develop and deploy components that can interoperate at runtime.
* Be compatible with existing server platforms. Vendors will be able to extend their existing products to support EJBs.
* Be compatible with other [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) APIs.
* Provide interoperability between enterprise Beans and Java EE components as well as non-Java programming language applications.
* Be compatible with the CORBA protocols (RMI-IIOP).

**EJB 1.1, final release (1999-12-17)**

*Major changes*:

* XML deployment descriptors
* Default JNDI contexts
* RMI over IIOP
* Security - role driven, not method driven
* Entity Bean support - mandatory, not optional

*Goals* for Release 1.1:

* Provide better support for application assembly and deployment.
* Specify in greater detail the responsibilities of the individual EJB roles.

**EJB 1.0 (1998-03-24)**

Announced at [JavaOne 1998](https://en.wikipedia.org/wiki/JavaOne),[[41]](https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans#cite_note-41) Sun's third Java developers conference (March 24 through 27) *Goals* for Release 1.0:

* Defined the distinct "EJB Roles" that are assumed by the component architecture.
* Defined the client view of enterprise Beans.
* Defined the enterprise Bean developer’s view.
* Defined the responsibilities of an EJB Container provider and server provider; together these make up a system that supports the deployment and execution of enterprise Beans.