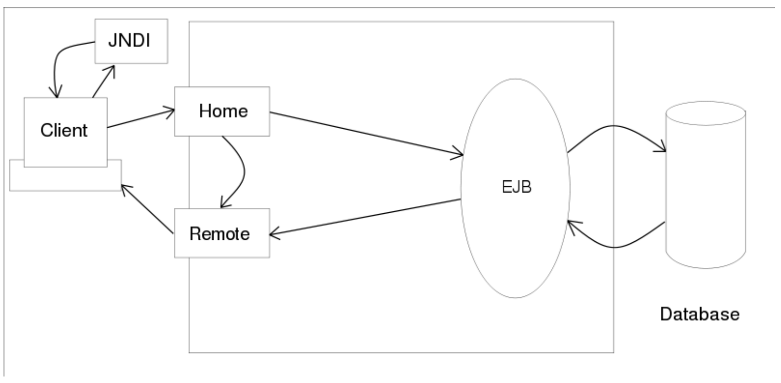
Enterprise JavaBean



Simple EJB2 Architecture

Enterprise JavaBeans (**EJB**) is a managed, server-side component architecture for modular construction of enterprise applications.

The EJB specification is one of several Java APIs in the Java EE specification. EJB is a server-side model that encapsulates the business logic of an application. The EJB specification was originally developed in 1997 by IBM and later adopted by Sun Microsystems (EJB 1.0 and 1.1) in 1999[1] and enhanced under the Java Community Process as JSR 19 [2] (EJB 2.0), JSR 153 [3] (EJB 2.1), JSR 220 [4] (EJB 3.0) and JSR 318 [5] (EJB 3.1).

The EJB specification intends to provide a standard way to implement the back-end 'business' code typically found in enterprise applications (as opposed to 'front-end' interface code). Such code was frequently found to address the same types of problems, and it was found that solutions to these problems are often repeatedly re-implemented by programmers. Enterprise JavaBeans were intended to handle such common concerns as persistence, transactional integrity, and security in a standard way, leaving programmers free to concentrate on the particular problem at hand.

Accordingly, the EJB specification details how an application server provides:

- Transaction processing
- Integration with the Persistence services offered by the Java Persistence API (JPA)
- Concurrency control
- Events using Java Message Service
- Naming and directory services (JNDI)
- Security (Java Cryptography Extension (JCE) [6] and JAAS)
- Deployment of software components in an application server
- Remote procedure calls using RMI-IIOP.
- Exposing business methods as Web Services.

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 specification does not detail anymore 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.

Rapid adoption followed by criticism

TEJA vision was persuasively presented by EJB advocates such as IBM and Sun Microsystems, and Enterprise JavaBeans were quickly adopted by large companies. Problems were quick to appear, however, and the reputation of EJBs began to suffer as a result. Some developers felt that the APIs of the EJB standard were far more complex than those developers were used to. An abundance of checked exceptions, required interfaces, and the implementation of the bean class as an abstract class were all unusual and counter-intuitive for many programmers. Granted, the problems that the EJB standard was attempting to address, such as object-relational mapping and transactional integrity, are complex. However many programmers found the APIs to be just as difficult if not more so, leading to a widespread perception that EJBs introduced complexity without delivering real benefits.

In addition, businesses found that using EJBs to encapsulate business logic brought a performance penalty. This is because the original specification only allowed for remote method invocation through CORBA (and optionally other protocols), even though the large majority of business applications actually do not require this 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. It was introduced largely to address the performance problems that existed with EJB 1.0.[7]

The complexity issue, however, continued to hinder EJB's acceptance. Although high-quality developer tools made it easy to create and use EJBs by automating most of the repetitive tasks, these tools did not make it any easier to learn how to use the technology. Moreover, a counter-movement had grown up on the grass-roots level among programmers. The main products of this movement were the so-called 'lightweight' (i. e. in comparison to EJB) technologies of Hibernate (for persistence and object-relational mapping) and Spring Framework (which provided an alternate and far less verbose way to encode business logic). Despite lacking the support of big businesses, these technologies grew in popularity and were adopted more and more by businesses who had become disillusioned with EJBs.

EJBs were promoted by Sun's Java Pet Store demo Java BluePrints. The use of EJBs was controversial and influential Java EE programmers such as Rod Johnson took positions in response to Java Pet Store that sought to deemphasize EJB use. Sun itself produced an alternative called Java Data Objects. Later, EJBs, Java Data Forms, and many of the ideas underlying Hibernate were combined to form EJB 3.0 which included the Java Persistence API and Plain Old Java Objects (POJOs). EJB 3.0 was less heavy weight than EJB 2.0 and provided more choices to developers.

Reinventing EJBs

Gradually an industry consensus emerged that the original EJB specification's primary virtue — enabling transactional integrity over distributed applications — was of limited use to most enterprise applications, and the functionality delivered by simpler frameworks like Spring and Hibernate was more useful. Accordingly, the EJB 3.0 specification (JSR 220) was a radical departure from its predecessors, following this new paradigm. It shows a clear influence from Spring in its use of POJOs, and its support for 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, the replacement for entity beans in EJB 3.0. The EJB 3.0 specification relies heavily on the use of annotations, a feature added to the Java language with its 5.0 release, to enable a much less verbose coding style.

Accordingly, in practical terms EJB 3.0 is very nearly a completely new API, bearing little resemblance to the previous EJB specifications.

Example

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

```
@Local
public interface CustomerServiceLocal {
   void addCustomer(Customer customer);
}

@Stateless
public class CustomerService implements CustomerServiceLocal {

  @PersistenceContext
  private EntityManager entityManager;

  public void addCustomer(Customer customer) {
    entityManager.persist(customer);
  }
}
```

The above defines a local interface and the implementation of a service class for persisting a Customer object (via O/R 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 CustomerServiceLocal customerService;

   public String addCustomer() {
      customerService.addCustomer(customer);
      context.addMessage(...); // abbreviated for brevity
      return "customer_overview";
   }
}
```

The above defines a 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, like 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

An EJB container holds two major types of beans:

- Session Beans 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[8] , in which case local semantics apply. All session beans support asynchronous execution for all views (local/remote/no-interface).
- Message Driven Beans (also known as MDBs or Message Beans). MDBs also support asynchronous execution, but via a messaging paradigm.

Session beans

Stateful Session Beans

Stateful Session Beans are business objects having state: 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. 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 [9] .

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

Stateless Session Beans 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 and thus concurrent access to the bean is prohibited. This makes a stateless session bean automatically thread-safe. Instance variables can be used during a single method call to the bean, but the contents of those instance variables are not guaranteed to be preserved across method 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 that way, which potentially increases performance). In this case, the asynchronous method will return a Future [10] instance.

Singleton Session Beans

Singleton Session Beans are business objects having a global shared state. 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

Message Driven Beans 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) specification. It may subscribe to JMS message queues or message topics, which are typically injected into the bean. 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 can not look-up an MDB instance. It just listens for any incoming message on e.g. a JMS queue or topic and processes them automatically. Only JMS support is required by the Java EE spec, but Message Driven Beans can support other messaging protocols. 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 calling and messaging.

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 have knowledge about the amount of consumers or 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 [11] 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 resource adapter would be needed and the MDB would be annotated with a reference to this[12] .

Legacy (deprecated) beans

Previous versions of EJB also used a type of bean known as an Entity Bean. These were distributed objects having persistent state. Beans in which their container managed the persistent state were said to be using *Container-Managed Persistence* (CMP), whereas beans that managed their own state were said to be using *Bean-Managed Persistence* (BMP). Entity Beans were replaced by the Java Persistence API in EJB 3.0, though as of 2007, CMP 2.x style Entity beans are still available for backward compatibility. As of EJB 3.1, JPA has been completely separated to its own spec and EJB will focus only on the "*core session bean and message-driven bean component models and their client API*" [13] .

Other types of Enterprise Beans have been proposed. For instance, *Enterprise Media Beans* (JSR 86 [14]) address the integration of multimedia objects in Java EE applications.

Execution

EJBs are deployed in an EJB container within the 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 [15] package. (The javax.ejb.spi [16] package is a service provider interface used only by EJB container implementations.)

Clients of EJB beans 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 then not a reference to the implementation bean itself, but to a proxy, that 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-like) services to a bean like transactions, security, interceptions, injections, remoting, etc.

E.g. a client invokes a method on a proxy, which will then first start a transaction with the help of the EJB container and then call the actual bean method. When the actual bean method returns, the proxy ends the transaction (i.e. by committing it or doing a rollback) and transfers control back to the client.

Transactions

EJB containers must support both container managed ACID transactions and bean managed transactions.

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.

Alternatively, the bean can also declare via an annotation that it wants to handle transactions programmatically via the JTA API. This mode of operation is called Bean Managed Transactions (BMT), since the bean itself handles the transaction instead of the container.

Events

JMS (Java Message Service) is used to send messages from the beans to client objects, to let clients receive asynchronous messages from these beans. MDB can be used to receive messages from client applications asynchronously using either a **JMS** Queue or a Topic.

Naming and directory services

As an alternative to injection, clients of an EJB can obtain a reference to the session bean's proxy object (the EJB stub) using JNDI. This alternative can be used in cases where injection is not available, such as non-managed beans 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 [17] [18] :

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)* would use the following code to obtain a (local) reference to it:

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

Security

The EJB Container is responsible for ensuring the client code has sufficient access rights to an EJB.

Legacy

Home interfaces and required business interface

With EJB 2.1 and earlier, each EJB had to provide a Java implementation class 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.

The two interfaces, referred to as the *Home* and the *Remote* interface, specified the signatures of the EJB's remote methods. The methods were split into two groups:

Class methods

> Not tied to a specific instance, such as those used to create an EJB instance (factory method) or to find an existing entity EJB (see EJB Types, above). These were declared by the Home interface.

Instance methods

> These are methods tied to a specific instance. They are placed in the Remote interface.

Required deployment descriptor

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 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 is an 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 EJB3.0 (JSR 220 [19]), the XML descriptor is replaced by Java annotations 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.

Version history

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

JSR 318 [5]. 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 packaging of EJB components
- EJB Lite: definition of a subset of EJB
- Portable EJB Global JNDI Names
- Singletons (Singleton Session Beans)
- Application Initialization and Shutdown Events
- EJB Timer Service Enhancements
- Simple Asynchrony (@Asynchronous for session beans)

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

JSR 220 [4] - *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 [3] - *Major changes*:

- Web service support (new): stateless session beans can be invoked over SOAP/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.
- 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 is used to specify deployment descriptors, replaces DTDs

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

JSR 19 [2] - *Major changes*: *Overall goals*:

- The standard component architecture for building *distributed* object-oriented business applications in Java.
- 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. 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 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, 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.

External links

- Sun's EJB Product main page [20]
- Java EE 6 API Javadocs [21]
- EJB 3.0 API Javadocs [22]
- The EJB 3.0 Specification [23]
- Sun's EJB 3.0 Tutorial [24]
- EJB (3.0) Glossary [25]
- JSR 318 [26] (EJB 3.1)
- JSR 220 [4] (EJB 3.0)
- JSR 153 [3] (EJB 2.1)
- JSR 19 [2] (EJB 2.0)
- EJB 2 example based tutorial issa [27]

References

[1] *J2EE Design and Development*, © 2002 Wrox Press Ltd., p. 5.
[2] http://www.jcp.org/en/jsr/detail?id=19
[3] http://www.jcp.org/en/jsr/detail?id=153
[4] http://www.jcp.org/en/jsr/detail?id=220
[5] http://jcp.org/en/jsr/detail?id=318
[6] http://java.sun.com/products/jce/index-14.html
[7] *J2EE Design and Development*, © 2002 Wrox Press Ltd., p. 19.
[8] http://blogs.sun.com/kensaks/entry/optional_local_business_interfaces
[9] http://blogs.sun.com/enterprisetechtips/entry/extended_persistence_context_in_stateful
[10] http://java.sun.com/javase/6/docs/api/java/util/concurrent/Future.html
[11] http://www.quartz-scheduler.org/
[12] http://www.mastertheboss.com/en/jboss-server/192-developing-quartz-mdb.html
[13] http://jcp.org/en/jsr/detail?id=318
[14] http://jcp.org/en/jsr/detail?id=86
[15] http://java.sun.com/javaee/6/docs/api/javax/ejb/package-summary.html
[16] http://java.sun.com/javaee/6/docs/api/javax/ejb/spi/package-summary.html
[17] http://blogs.sun.com/MaheshKannan/entry/portable_global_jndi_names
[18] http://blogs.sun.com/kensaks/entry/portable_global_jndi_names
[19] http://jcp.org/en/jsr/detail?id=220
[20] http://java.sun.com/products/ejb/index.html
[21] http://java.sun.com/javaee/6/docs/api/
[22] http://java.sun.com/products/ejb/javadoc-3_0-fr/index.html
[23] http://java.sun.com/products/ejb/docs.html
[24] http://java.sun.com/javaee/5/docs/tutorial/doc/bnblr.html
[25] http://www.infoq.com/articles/EJB-Glossary
[26] http://www.jcp.org/en/jsr/detail?id=318
[27] http://javalessons.com/cgi-bin/fun/java-tutorials-main.cgi?sub=ejb&ses=ao789

Article Sources and Contributors

Enterprise JavaBean *Source*: http://en.wikipedia.org/w/index.php?oldid=400836678 *Contributors*: Ahoerstemeier, Alainr345, Anwar saadat, Arjant, Audriusa, Beeromatic, Beetstra, Bezenek, Bj96, Borgx, Bovineone, Bposert, Charles Matthews, Chnvskumar, Chowbok, Cícero, DMacks, DanMS, DavidLeeLambert, Diptanshu.D, Docboat, Donsez, Doug Bell, Dstrube, Edenphd, Ehn, Elsp, Enviroboy, Errandir, Explanator, Fastfilm, Femto, Findepi, Finlay McWalter, Fred Bradstadt, Gcm, Gepree, Gherson2, Gmoore19, Gondooley, Gosub, Gracefool, Hadal, HairyFotr, Hertzsprung, Howthatdo, Hu12, IP Singh, Ideoplex, Ijevans, Ivan Pozdeev, J Di, James.sutherland, Jarsyl, Jems421, Jgold03, Jh51681, Jonik, JoseREMY, Jstplace, Julesd, KevekKerinth, Khalid hassani, Kks krishna, Kku, Kolhebhushan, Korg, Kutekrish, Lfstevens, LiDaobing, Liujiang, M-le-mot-dit, MER-C, MJKazin, Majedkh, Marlow4, Materialscientist, Mathiastck, Matijahan, Matt Crypto, Mazi, Mboverload, MiNombreDeGuerra, Mindaugas, MrOllie, MrStalker, Music Sorter, Muzzamo, Nixdorf, Oberiko, Ohnoitsjamie, Olekva, Penzes.tamas, Pi.r, Poc0123, Radagast83, Randy549, RedWolf, Rettetast, RichardVeryard, Richmd, Rookkey, SAE1962, Schastain, Schenkeli, Sfmontyo, Slappstah, Steve802, Tangotango, Technobadger, Texture, The Silent Contributor, Theone256, Uzume, Vanessaezekowitz, Virgiltrasca, Waggers, Woohookitty, Wwwwolf, Yafai, Zoicon5, 345 anonymous edits

Image Sources, Licenses and Contributors

Image:Ejb.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Ejb.svg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: Vanessaezekowitz, 1 anonymous edits

License