Sun Microsystems

Enterprise JavaBeans^{TM} to CORBA Mapping

This is the specification of the standard mapping of the Enterprise JavaBeans^{TM} architecture to CORBA.

Please send technical comments on this specification to:

ejb-spec-comments@sun.com

Please send product and business questions to:

ejb-marketing@sun.com

Version 1.1

Sanjeev Krishnan

August 11, 1999

Contents

1 Introduction 6

2 Goals 7

3 Mapping of Distribution 9

4 Mapping of Naming 12

5 Mapping of Transactions 14

6 Mapping of Security 16

Appendix A: References 17

Appendix B: Enterprise JavaBeans IDL 18

1 Introduction

Enterprise JavaBeans^{TM} (EJB)[1] is a component architecture for development and deployment of object-oriented distributed enterprise-level Java applications. Applications written using Enterprise JavaBeans are scalable, transactional, and multi-user secure. These applications can be written once, and then deployed on any EJB-enabled server platform.

We expect that many EJB servers will be based on the CORBA/IIOP [2] industry standards. To ensure interoperability among CORBA-based EJB server implementations from multiple-vendors, we have defined a standard mapping of EJB to CORBA. This document corresponds to the EJB specification version 1.1.

This mapping must be used in conjunction with the relevant CORBA standards to ensure full on-the-wire interoperability.

The use of the EJB to CORBA mapping by an EJB Server is not a requirement for EJB 1.1 compliance. A later release of the Java 2 Platform, Enterprise Edition (J2EE) [11] is likely to require that a J2EE platform vendor implement the EJB to CORBA mapping.

1.1 Target Audience

The target audience for this specification are vendors of transaction processing platforms, vendors of enterprise application tools, and other vendors who want to use the CORBA/IIOP standard to provide support for Enterprise JavaBeans in their products.

1.2 Mapping Overview

The EJB to CORBA mapping is divided into four areas:

- *Mapping of Distribution* - defines the relationship between an Enterprise JavaBean and a CORBA object, and the mapping of the Java RMI remote interfaces defined in the EJB specification to OMG IDL.
- *Mapping of Naming* - specifies how CORBA's COS Naming service is used to locate EJBHome objects.
- *Mapping of Transactions* - defines the mapping of EJB transaction features to the CORBA Object Transaction Service.
- *Mapping of Security* - defines the mapping of the security features in EJB to CORBA security.

1.3 Acknowledgments

Rohit Garg authored version 1.0 of the EJB-to-CORBA mapping specification. Vlada Matena, Ken Cavanaugh and Vivek Nagar helped to define version 1.1 of the mapping. The input from several reviewers helped to improve this document.

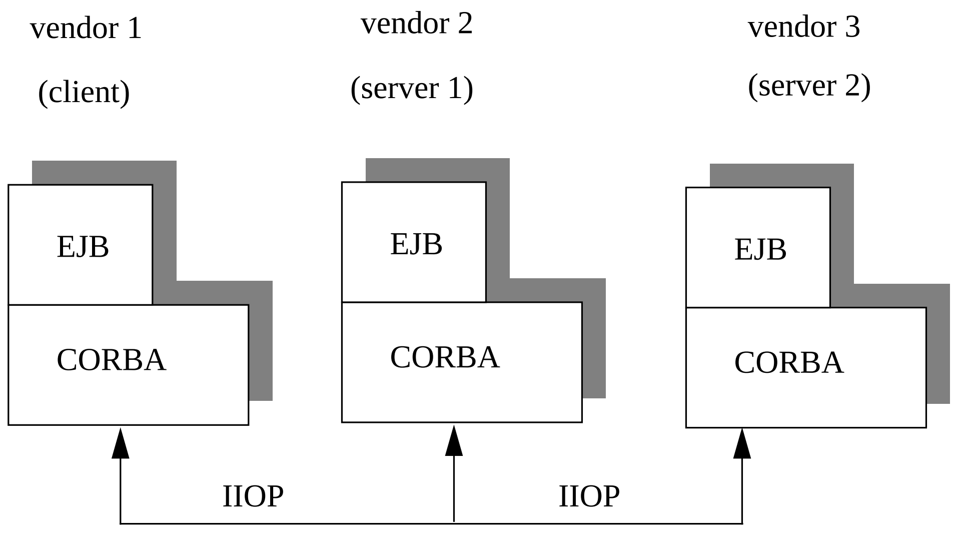
2 Goals

The primary goals of this specification are:

- define "on-the-wire" interoperability so that multiple CORBA based implementations of EJB containers can interoperate over a network. This includes sharing information between the naming, transactions and security services from different vendors.
- allow CORBA clients (written in any language supported by CORBA) to access enterprise beans deployed in a CORBA based EJB server through standard CORBA APIs.

For example, a CORBA client program can mix calls to CORBA objects and enterprise beans within the scope of a single transaction, even if the enterprise beans are located on multiple CORBA-based EJB servers provided by different vendors.

This mapping must be used in conjunction with the relevant CORBA standards to ensure full on-the-wire interoperability.



2.1 Types of CORBA Clients

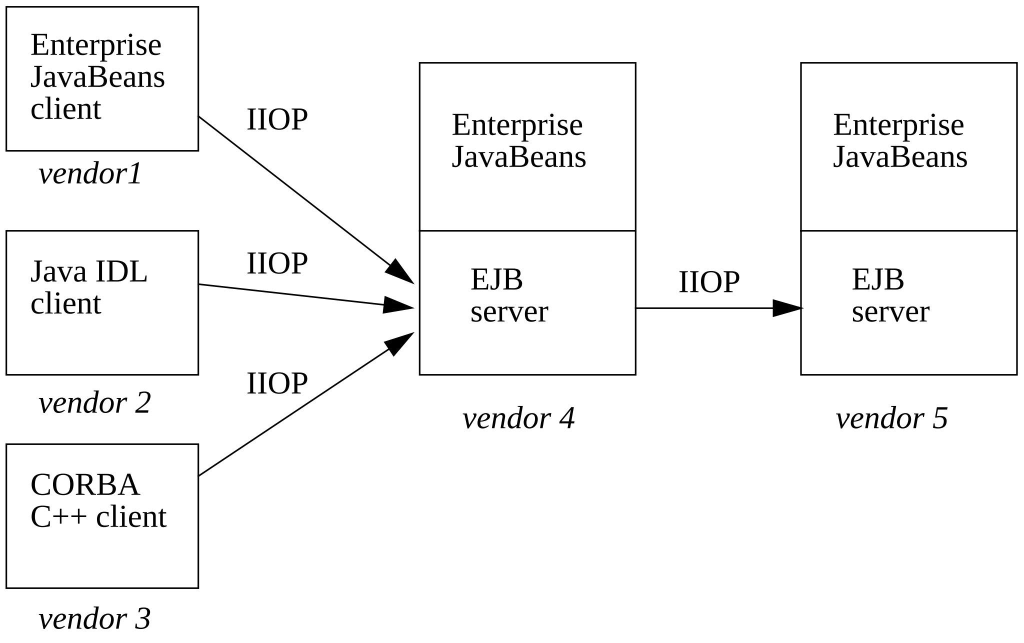
There are two types of CORBA clients for an EJB server:

- **EJB/CORBA Client** - A Java client that uses the EJB client-view APIs. This type of client uses the Java Naming and Directory Interface (JNDI) to locate objects, Java RMI over IIOP to invoke remote methods, and the *javax.transaction.UserTransaction* interface of the Java Transaction API (JTA) to demarcate transaction boundaries. The use of CORBA IDL is implicit (i.e.

the programmer writes only Java code and the corresponding CORBA IDL is used implicitly by the runtime).
An enterprise bean running in a CORBA based EJB server is also an EJB/CORBA client to other enterprise beans.

- **Plain CORBA Client** - A client written in any language that uses a language specific binding of the CORBA IDL. Such a client uses COS Naming APIs to locate objects, CORBA IDL to invoke remote methods, and the CORBA Object Transaction Service APIs to demarcate transactions. The use of CORBA IDL is explicit (i.e. the programmer creates an IDL file and runs an IDL compiler to generate stubs for a given language).

This mapping ensures that both types of clients interoperate with a CORBA-based EJB server by producing the same bits on the wire.



3 Mapping of Distribution

This chapter describes the mapping of interfaces and classes defined by the EJB1.1 specification to CORBA IDL. Appendix B contains the complete IDL for Enterprise JavaBeans.

Even though CORBA does not mandate using IIOP as the wire protocol for remote invocations, IIOP is the de-facto standard wire protocol for most CORBA products. This specification requires that EJB/CORBA compliant implementations use IIOP[1] as the communication protocol.

3.1 Mapping Java Remote Interfaces to IDL

For each Enterprise JavaBean that is deployed in the EJB Server, there are two Java RMI remote interfaces - the bean’s EJBHome interface, and the bean’s EJBObject interface. The Java Language to IDL Mapping [7] specification describes precisely how these remote interfaces are mapped to IDL. This mapping to IDL is typically implicit when Java RMI over IIOP is used to invoke on enterprise beans.

3.1.1 Marking of transaction-enabled enterprise bean interfaces

For enterprise beans that may execute within the scope of a transaction started by the client, the client’s transaction context must be implicitly propagated to the EJB server. The IDL interface for such enterprise beans must inherit from *CosTransactions::TransactionalObject*. The exact rules for this are specified in Section 5.1.

3.2 Mapping value objects to IDL

The EJB1.1 specification describes three Java interfaces that are passed by value during remote invocations: *javax.ejb.Handle, javax.ejb.HomeHandle* and *javax.ejb.EJBMetaData*. In addition, the Enumeration or Collection objects returned by EntityBean find methods are also value types. These interfaces are mapped to IDL abstract valuetypes or abstract interfaces, using the rules in the Java Language to IDL Mapping.

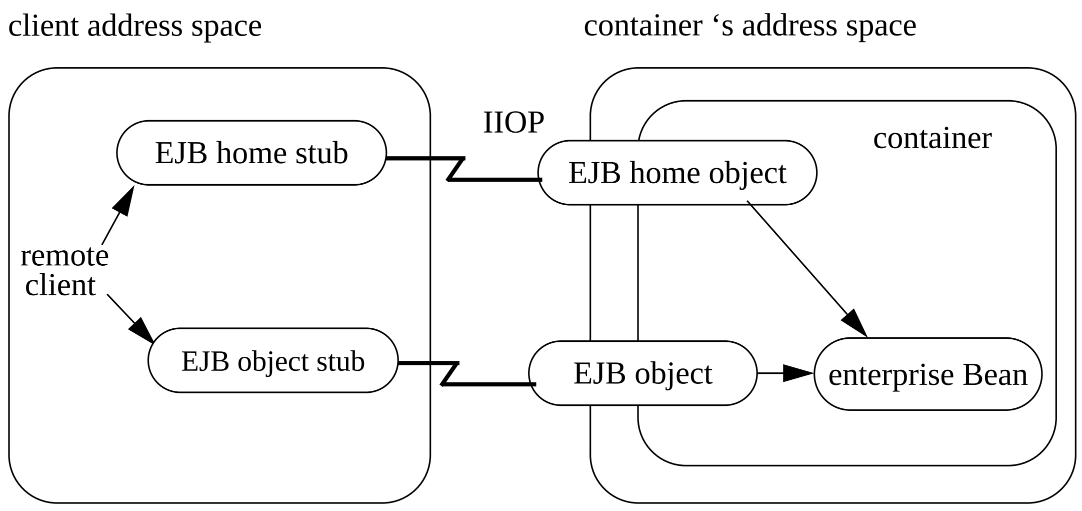
Concrete implementations of these value types as well as application-specific value types are required to be available to clients (either pre-installed in the client’s environment or by downloading from the server) and can be potentially instantiated in a different vendor’s CORBA runtime environment.

In addition, several Java exception classes that are thrown by remote methods also result in concrete IDL value types.

3.3 Client Side Stubs

The following figure illustrates the runtime objects used in a typical distributed EJB-enabled CORBA environment.

1.plain insecure IIOP, SECIOP, or IIOP over SSL



Depending on the client type, the client stubs are either RMI-IIOP stubs, or stubs generated from IDL as defined by the language-specific CORBA mappings [6].

This specification does not impose any additional requirements on the type of client-side stubs created by ORBs while unmarshaling remote object references. Thus the stub received by the client application may not correspond to the most-derived type of the EJB home/object. A CORBA client should use language-specific APIs for narrowing remote object references to the desired type. An EJB/CORBA client should use the *javax.rmi.PortableRemoteObject.narrow* API for obtaining the desired type.

3.4 Mapping of system exceptions

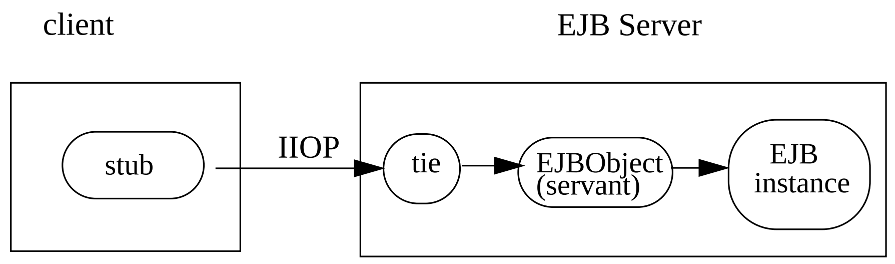
Java system exceptions including *java.rmi.RemoteException* and its subclasses may be thrown by the EJB container, as specified in EJB1.1. The EJB server is required to map these exceptions to CORBA system exceptions as specified in the following table:

System exception thrown by EJB container	**CORBA system exception received by client**
javax.transaction. TransactionRolledbackException	TRANSACTION_ROLLEDBACK
javax.transaction. TransactionRequiredException	TRANSACTION_REQUIRED
javax.transaction. InvalidTransactionException	INVALID_TRANSACTION
java.rmi.NoSuchObjectException	OBJECT_NOT_EXIST
java.rmi.RemoteException	UNKNOWN

For EJB/CORBA clients, the ORB's unmarshaling machinery will map CORBA system exceptions received in the IIOP reply message to the appropriate Java exception as specified in the Java Language to IDL mapping.

3.5 CORBA Object and Enterprise JavaBean Relationship

As a server-side implementation technique, the CORBA runtime may use an RMI-IIOP servant implementing the enterprise bean's EJBObject interface which receives a method invocation and delegates it to the appropriate enterprise bean instance. One way to achieve this is to use Tie based skeletons (as defined in the Java Language to IDL Mapping [7]). Since the architecture of stubs and skeletons does not relate to on-the-wire interoperability, it is not specified in this document. This document also does not require usage of the Portable Object Adapter or any other CORBA based server-side architecture.



4 Mapping of Naming

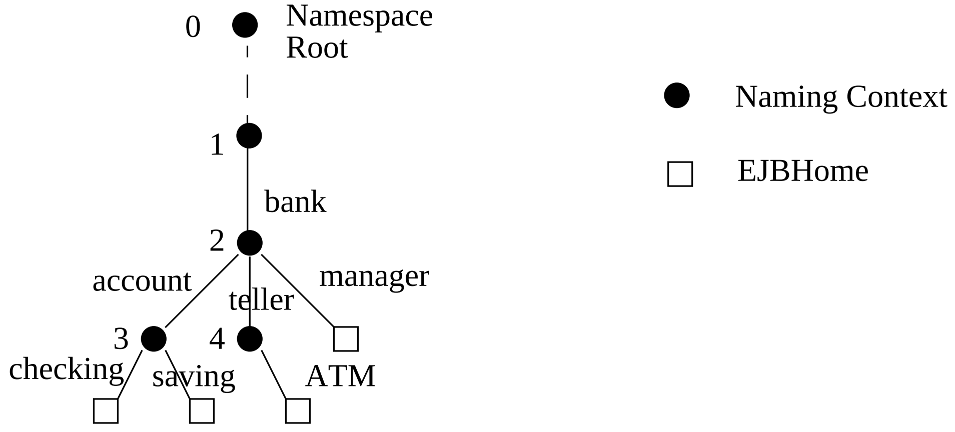
A CORBA based EJB runtime is required to use the OMG COS NameService [4] for publishing and resolving EJBHome objects. EJB/CORBA clients access the COS NameService by using the Java Naming and Directory Interface API with the standard COS Naming service provider [10]. Plain CORBA clients access the COS NameService by using the IDL for the CosNaming module specified in [4] .

The CORBA Interoperable Naming Service specification [9] should also be implemented by CORBA-based EJB servers to allow clients to lookup the root naming context in an interoperable manner.

4.1 COS Namespace Layout

When an EJB1.1 jar is deployed into an EJB/CORBA server environment, the deployer chooses a pathname in the COSNaming namespace at which the EJBHome object reference is bound. The EJB1.1 specification does not prescribe the format of this pathname.

For example, if an ejb-jar contains four enterprise beans whose EJBHome objects are deployed with names: *bank/account/checking, bank/account/saving, bank/teller/ATM*, and *bank/manager* then the COSNaming namespace would be organized as shown in the following figure:



In the figure above, there are 5 naming contexts:

- naming context 0 is the root of the COSNaming name space. One possible way of obtaining it is the ORB’s resolve_initial_references method with “NameService” as the argument.
- naming context 1 is the subcontext under which EJBHome objects for enterprise beans in this ejb-jar are “installed”. Note that this specification does not mandate any relationship between naming contexts 0 and 1. If they are not

the same, then the client doing the resolve operation has to be configured with the path between them. There is also no requirement that all EJBHome objects in an ejb-jar should be installed under one subcontext.

- naming context 2 is bound to context1 with name `bank`; naming contexts 3 and 4 are bound to context 2 with names `account` and `teller` respectively.

5 Mapping of Transactions

A CORBA based EJB runtime is required to use an implementation of the CORBA Object Transaction Service (OTS) version 1.1 [5] for transaction support. The following sections describe the mapping of the transaction concepts in EJB to OTS.

5.1 Transaction Propagation

For enterprise beans that may execute within the scope of a transaction started by the client, the client's transaction context must be implicitly propagated to the EJB server. The OTS requires client ORBs to propagate transaction context to the server if the server CORBA object inherits from the *CosTransactions::TransactionalObject* IDL interface. This subsection specifies the rules which allow transaction context propagation to occur for enterprise beans, based on the OTS.

EJB allows transaction attributes to be specified per method, while OTS only allows an entire IDL interface to be marked transactional. The rules below ensure that transaction context will be propagated if any method of an enterprise bean needs to execute in the client's transaction context. However, in some cases there may be extra performance overhead of propagating the client's transaction context even if it will not be used by the enterprise bean method.

The following rules list the types of EJB interfaces having remote methods that execute in the client's transaction context:

- SessionBean remote interfaces satisfying both the following rules:
  - the SessionBean's transaction demarcation type is set to "Container", i.e. SessionBeans with container-managed transactions, and
  - at least one of the SessionBean's remote interface methods has one of the transaction attributes "Supports", "Required" or "Mandatory".
- EntityBean remote interfaces satisfying the following rule:
  - at least one of the EntityBean's remote interface methods has one of the transaction attributes "Supports", "Required" or "Mandatory".
- EntityBean home interfaces satisfying the following rule:
  - at least one of the EntityBean's home interface methods has one of the transaction attributes "Supports", "Required" or "Mandatory".

The IDL mapping of an EJB interface satisfying the above rules must inherit from the *CosTransactions::TransactionalObject* IDL interface. This ensures that if the client makes an invocation to the EJB home/object within the scope of a transaction, the transaction context will be implicitly propagated to the server.

For EJB/CORBA servers which use implicit IDL, the is_a operation (defined on *CORBA::Object*) with the repository-id parameter "*IDL:omg.org/CosTransactions/TransactionalObject:1.0*" should return TRUE for EJB interfaces satisfying the above rules.

5.2 Client-side Demarcation

A CORBA client will typically use the *CosTransactions::Current* OTS interface to demarcate transaction boundaries. An EJB/CORBA client will use the *javax.transaction .UserTransaction* interface in the Java Transaction API to demarcate transaction boundaries. In both cases the ORB must propagate transaction context implicitly to the server as described in section 5.1.

6 Mapping of Security

The main security concern in the EJB1.1 specification is access control, which requires the EJB server to determine the client's security identity.

This specification does not require usage of a specific CORBA security protocol for propagating security information. Vendors should use compatible CORBA security protocols to ensure on-the-wire interoperability. This specification will track the Common Secure Interoperability RFP at OMG [8].

EJB/CORBA servers should determine the client identity based on the actual security and communication protocols used by the ORB, as specified in the CORBA Security Service [3]:

- *plain IIOP* - since the Security Service does not specify the mechanisms for propagating identity information in plain IIOP messages, the determination of client identity is vendor specific.
- *Common Secure IIOP* (CSI) - The client identity is defined by the specific mechanism (GSSKerberos, SPKM, CSI-ECMA) used with SECIOP (Secure IIOP).
- *IIOP over SSL* - The client's identity is the X.500 distinguished name of the subject obtained from the X.509 certificate during SSL client authentication. *Note: since SSL does not support delegation, an EJB/CORBA server using IIOP over SSL may not be able to propagate client principals from caller to callee in an interoperable manner.*

Appendix A: References

[1] Enterprise JavaBeans Specification, Version 1.1 (*http://java.sun.com/products/ejb/docs.html*)

[2] CORBA 2.3 Specification (*http://www.omg.org/cgi-bin/doc?formal/98-12-01*)

[3] CORBA Security Service (*http://www.omg.org/corba/sectrans.html#sec*)

[4] CORBA Naming Service (*http://www.omg.org/corba/sectrans.html#nam*)

[5] CORBA Transaction Service (*http://www.omg.org/corba/sectrans.html#trans*)

[6] Mapping of OMG IDL to Java (*http://www.omg.org/cgi-bin/doc?formal/99-07-53*)

[7] Java Language to IDL Mapping (*http://www.omg.org/cgi-bin/doc?formal/99-07-59*)

[8] Common Secure Interoperability RFP (*http://www.omg.org/docs/orbos/99-01-10.pdf)*

[9] Interoperable Name Service (*http://www.omg.org/docs/orbos/98-10-11.pdf)*

[10] Java Naming and Directory Service Providers (*http://java.sun.com/products/jndi/serviceproviders.html*)

[11] Java 2 Platform, Enterprise Edition Specification (*http://java.sun.com/j2ee/docs.html*)

Appendix B: Enterprise JavaBeans IDL

B.1 OMG IDL mapping for Enterprise Javabeans 1.1

```
module javax {
module ejb {

// Exceptions
valuetype CreateException: ::java::lang::_Exception {
   factory create__( );
   factory create__CORBA_WStringValue( in ::CORBA::WStringValue arg0 );
};
exception CreateEx {
   CreateException value;
};
#pragma ID CreateException "RMI:javax.ejb.CreateException:7C78AA9E9FB0D1B7:575FB6C03D49AD6A"


valuetype DuplicateKeyException: ::javax::ejb::CreateException {
   factory create__( );
   factory create__CORBA_WStringValue( in ::CORBA::WStringValue arg0 );
};
exception DuplicateKeyEx {
   DuplicateKeyException value;
};
#pragma ID DuplicateKeyException "RMI:javax.ejb.DuplicateKeyException:3112CAD2A6288A29:9ADDF450AB68AAC4"


valuetype FinderException: ::java::lang::_Exception {
   factory create__( );
   factory create__CORBA_WStringValue( in ::CORBA::WStringValue arg0 );
};
```

```
exception FinderEx {
   FinderException value;
};
#pragma ID FinderException "RMI:javax.ejb.FinderException:7C78AA9E9FB0D1B7:79EE1514C8B7CA15"


valuetype ObjectNotFoundException: ::javax::ejb::FinderException {
      factory create__( );
      factory create__CORBA_WStringValue( in ::CORBA::WStringValue arg0 );
};
exception ObjectNotFoundEx {
   ObjectNotFoundException value;
};
#pragma ID ObjectNotFoundException "RMI:javax.ejb.ObjectNotFoundException:3112CAD2A6288A29:00106DD5ADF01DDA"


valuetype RemoveException: ::java::lang::_Exception {
   factory create__( );
   factory create__CORBA_WStringValue( in ::CORBA::WStringValue arg0 );
};
exception RemoveEx {
   RemoveException value;
};
#pragma ID RemoveException "RMI:javax.ejb.RemoveException:7C78AA9E9FB0D1B7:C06A008FD05A462A"



// Forward references
interface EJBHome;
interface EJBObject;
```

```
abstract valuetype EJBMetaData;
abstract interface Handle;
abstract interface HomeHandle;



// Interfaces
interface EJBHome {
   readonly attribute ::javax::ejb::EJBMetaData EJBMetaData;
   readonly attribute ::javax::ejb::HomeHandle homeHandle;
   void remove__java_lang_Object( in ::java::lang::_Object arg0 )
raises ( ::javax::ejb::RemoveEx );
   void remove__javax_ejb_Handle( in ::javax::ejb::Handle arg0 )
raises ( ::javax::ejb::RemoveEx );
};
#pragma ID EJBHome “RMI:javax.ejb.EJBHome:0000000000000000”


interface EJBObject {
   readonly attribute ::javax::ejb::EJBHome EJBHome;
   readonly attribute ::javax::ejb::Handle handle;
   readonly attribute ::java::lang::_Object primaryKey;
   boolean isIdentical( in ::javax::ejb::EJBObject arg0 );
   void remove( )
raises ( ::javax::ejb::RemoveEx );
};
#pragma ID EJBObject “RMI:javax.ejb.EJBObject:0000000000000000”



// Abstract interfaces
abstract interface Handle {
    readonly attribute ::javax::ejb::EJBObject EJBObject;
};
#pragma ID Handle "RMI:javax.ejb.Handle:0000000000000000"


abstract interface HomeHandle {
    readonly attribute ::javax::ejb::EJBHome EJBHome;
};
#pragma ID HomeHandle "RMI:javax.ejb.HomeHandle:0000000000000000"



// Value types
abstract valuetype EJBMetaData {
    ::javax::ejb::EJBHome getEJBHome( );
    ::javax::rmi::CORBA::ClassDesc getHomeInterfaceClass( );
    ::javax::rmi::CORBA::ClassDesc getPrimaryKeyClass( );
    ::javax::rmi::CORBA::ClassDesc getRemoteInterfaceClass( );
    boolean isSession( );
};

}; // end module ejb
}; // end module javax
```