

Tutorial: Using EJB Local Interfaces

This tutorial describes how the Financial Brokerage Service (FBS 10g) sample application uses *local interfaces* to improve application performance.

This tutorial assumes that you are familiar with the FBS 10g and have installed and configured the required software as described in [About the Financial Brokerage Service](#).

Contents

1. [Concepts](#)
2. [Design](#)
3. [Required Software](#)
4. [Setup](#)
5. [Implementation](#)
6. [Resources](#)
7. [Feedback](#)





Concepts

One goal of the original Enterprise JavaBeans (EJB) specification was to expand upon Java's write-once, run-anywhere philosophy: developers could build applications from server-side components and focus on business logic and other features without worrying about where the components would be deployed. To achieve this, early EJB specifications defined a robust yet fairly complex infrastructure—including remote interfaces, the Java Remote Method Invocation (RMI) protocol, and RMI over IIOP (Internet Inter-ORB Protocol)—that enables distributed components to find each other and communicate across the Internet or an intranet.

As developers worked with EJBs, two things became apparent. First, flexibility came at the expense of performance. Routing method calls and responses through remote interfaces and RMI gobbled up processing resources, and applications simply ran slower. Second, in many cases, EJBs were deployed to the same container—they didn't need the remote communication infrastructure but had to use it anyway.

Responding to this kind of feedback, the EJB 2.0 expert group introduced support for *local interfaces*. This mechanism, defined in the EJB 2.0 specification, enables components in the same container to bypass RMI and call each other's methods directly. In general, direct local method calls are faster than remote method calls. The downside is a loss of flexibility: because bean and client must run in the same container, the location of the bean is not transparent to the client (as it is with remote interfaces).

The [Design](#) section describes how OTN developers decided when and how to use local interfaces in the FBS 10g. Coding details are discussed in the [Implementation](#) section.





Design

As described in the [Concepts](#) section, local interfaces can improve the performance of an EJB-based application, but there are trade-offs. Up-front planning is essential, taking into account deployment scenarios, evaluating the need for component reuse, deciding which interfaces will be local, which will be remote. Also, local and remote interfaces use different calling semantics: remote interfaces pass parameters by value, while local interfaces pass them by reference. Clients must be designed and built accordingly.

OTN developers identified several areas where local interfaces could be used in the FBS 10g. In this application, AlertsBean, PreferencesBean, TradeDetails, and PortfolioBean are targets of a container-managed relationship with UserAccount. UserAccount is a local entity bean that has 1:N relationships with Alerts, Preferences, TradeDetails, and Portfolio records. (For more information about container-managed relationships, see the [Using Container-Managed Relationships](#) tutorial.) Therefore, per the EJB specification, the beans must expose local interfaces. They must also reside in the same EJB JAR file. This design, in addition to meeting the requirements for container-managed relationships, improves the performance of the application because local method calls are generally faster than remote calls.

Other factors can influence the choice of local or remote interfaces. In addition to container-managed relationships, beans that are tightly-coupled logically are good candidates for local interfaces. In the FBS 10g, a Preference bean is always associated with a User bean, so it makes sense to co-locate them and have them enable access via local interfaces. Another factor is the type of client. If a bean is only going to be accessed by Web components or other beans, consider local interfaces; otherwise, implement remote interfaces because they enable access from anywhere. Similarly, if the application design calls for beans distributed across multiple servers, go with remote interfaces. If you're not certain about where a bean will be deployed or how it will be accessed, choose remote interfaces to keep your options open.

Finally, a bean *can* expose both remote and local interfaces, but such implementations are rare. To watch a viewlet that shows how the FBS 10g uses local interfaces, browse to:

- http://otn.oracle.com/sample_code/tutorials/fbs/files/viewlets/ibfbs_ejb2_viewlet.html





Required Software



This tutorial presents several code examples. If you want to study them in context, download and install the [FBS 10g source code](#). If you also want to build and run the FBS 10g, you will need the software listed in the [Required Software](#) section of *About the Financial Brokerage Service*.





Setup

Each bean that exposes a local interface requires the following files, where *<BeanName>* represents the name of the bean, for example, `PreferencesHomeLocal`.

File	Description
<code><BeanName>HomeLocal</code>	Local Home interface, extends <code>javax.ejb.EJBLocalHome</code> .
<code><BeanName>Local</code>	Local interface, extends <code>javax.ejb.EJBLocalObject</code> .
<code><BeanName>Bean</code>	Bean implementation class, implements <code>javax.ejb.EntityBean</code> .

Clients typically import the following classes:

- `java.rmi.RemoteException`
- `javax.naming.InitialContext`
- `javax.ejb.CreateException`
- `javax.ejb.FinderException`

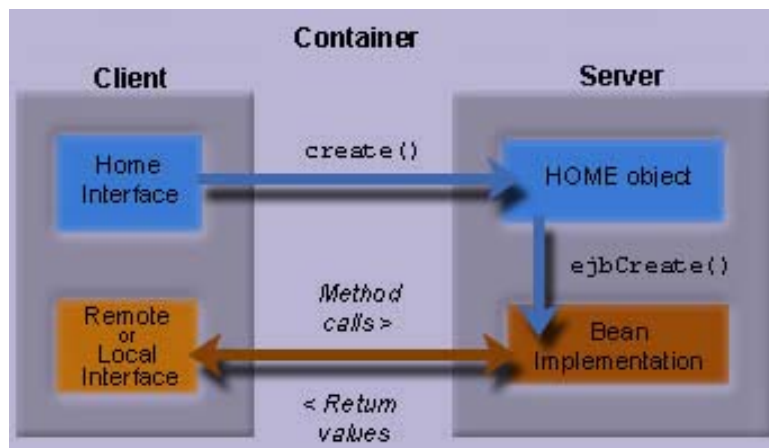
To configure your system to build and run the FBS 10g sample application, see the [Setup](#) section of *About the Financial Brokerage Service*.



Implementation

This section describes how local interfaces are implemented in the FBS 10g. The [Design](#) section describes how OTN developers decided where to use local interfaces in the FBS 10g.

The figure below shows how various elements interact when a client creates an EJB instance and calls methods through a local interface. When a client invokes `create` (which returns an interface—in this case, a local interface) on the Home Interface, the EJB Container calls `ejbCreate` to instantiate the bean. Then the client can access the bean through the interface returned by `create`.



In the FBS 10g, AlertsBean, PreferencesBean, TradeDetails, and PortfolioBean are local EJBs with corresponding local interfaces. UserAccount is a local Entity Object which has 1:N relationships with Alerts, Preferences, TradeDetails, and Portfolio records. This section focuses on the PreferencesBean implementation, but the same principles apply to the others.

FBS 10g users can set preferences to indicate what information (news, a price quote, or both) they want for specified stock ticker symbol. For example, a person who is thinking of buying Oracle stock might want a quote for the symbol ORCL, while someone who already owns Oracle stock might want to see news items to keep abreast of developments at the company. This functionality is implemented in the following files:

File	Description
PreferencesHomeLocal	Local Home interface, extends <code>javax.ejb.EJBLocalHome</code> , declares two methods: <code>create</code> and <code>findByPrimarykey</code> .

PreferencesLocal	Local interface, extends javax.ejb.EJBLocalObject, declares methods to get and set account number, ticker symbol, and preference.
PreferencesBean	Bean implementation class, implements javax.ejb.EntityBean.
PreferencesInfo	Encapsulates preference data for a particular user account, implements java.io.Serializable.

The following listings show how a client class works with local interfaces. The process begins in the `ejbCreate` method with a call to `InitialContext.lookup`. This is a standard EJB call—the `InitialContext` class provides a context for performing naming operations. However, with local interfaces, the client and the bean are in the same container, so the client knows where to find the bean. As a result, the location can be hard-coded into the `lookup` call.

```
public class UserManagementSessionFacadeBean implements javax.ejb.SessionBean {
...
    /** Reference to Local Home Interface of UserAccountBean */
    UserAccountHomeLocal userAccountHomeLocal = null;
    /** Reference to Local Home Interface of PreferencesBean */
    PreferencesHomeLocal preferencesHomeLocal = null;
...
    public void ejbCreate() {
        try {
            InitialContext ic = new InitialContext();
            ...
            preferencesHomeLocal =
                (PreferencesHomeLocal)ic.lookup("java:comp/env/ejb/PreferencesHomeLocal");
            ...
        }
    }
}
...
}
```

Once the client has a reference to the bean's local Home interface, it can call `create` to get a reference to the bean's local interface, and with that, can call the bean's methods.

```
public class UserManagementSessionFacadeBean implements javax.ejb.SessionBean {
...
    public String addPreferences(Integer accountNumber,
        PreferencesInfo preferencesInfo) throws RemoteException {
        ...
        if (!symbolPresent && validSymbol) {
            PreferencesLocal preferencesLocal =
                preferencesHomeLocal.create(this.getNextID("PREFERENCES_SEQ"),
```

```

        accountNumber,
        preferencesInfo.getSymbol(),
        preferencesInfo.getPreferenceType());
    allPrefIter.add(preferencesLocal); // allPrefIter is an ArrayList
}
...
}
...
}

```

The code below shows how a client can call methods from a bean's local interface. The call to `userAccountLocal.getPreferences` returns a collection of `PreferencesLocal` interfaces, one interface for each of the user's preferences. The code then iterates over this collection, each time retrieving a reference to a `PreferencesLocal` interface and using it to call `getSymbol` and `getPrefType`.

```

public class UserManagementSessionFacadeBean implements javax.ejb.SessionBean {
...
    public Collection getPreferences(Integer accountNumber) throws RemoteException {
        Collection allPreferences = new ArrayList();
        try {
            UserAccountLocal userAccountLocal =
                userAccountHomeLocal.findByPrimaryKey(accountNumber);
            // Get all the preferences and Loop through them
            Iterator prefLocalIter = (userAccountLocal.getPreferences()).iterator();
            while (prefLocalIter.hasNext()) {
                // Get the next handle to PreferencesBean Local EJB Object
                PreferencesLocal preferencesLocal = (PreferencesLocal) prefLocalIter.next();
                // Add a new instance of PreferencesInfo Value Object to final list
                allPreferences.add(new PreferencesInfo(preferencesLocal.getSymbol(),
                    preferencesLocal.getPrefType()));
            }
        } catch (FinderException ex) { // Trap Errors While Finding EJB Objects
            ...
        }
        return allPreferences; // Return All Preferences
    }
...
}

```





Resources

This tutorial is part of a series, *J2EE and EJB 2.1 Techniques*, based on the Financial Brokerage Service (FBS 10g) sample application. Following are links to resources that can help you understand and apply the concepts and techniques presented in this tutorial.

Resource	URL
How-To: Implement Local Interfaces	http://otn.oracle.com/tech/java/oc4j/htdocs/how-to-ejb-local-interfaces.html
Viewlet: Local Interfaces in the FBS 10g	http://otn.oracle.com/sample_code/tutorials/fbs/files/viewlets/ibfbs_ejb2_viewlet.html
EJB Overview	http://otn.oracle.com/tech/java/oc4j/doc_library/902/ejb/overview.htm
EJB Primer for OC4J	http://otn.oracle.com/tech/java/oc4j/doc_library/902/ejb/primer.htm
EJB Downloads and Specifications	http://java.sun.com/products/ejb/docs.html
Oracle Products	http://otn.oracle.com/products/





Feedback

If you have questions or comments about this tutorial, you can:

- Post a message in the [OTN Sample Code discussion forum](#). OTN developers and other experts monitor the forum.
- Send email to the author. <mailto:Robert.Hall@oracle.com>

If you have suggestions or ideas for future tutorials, please send email to:

- <mailto:Raghavan.Sarathy@oracle.com>

