EJB Client

This topic explains how to configure and apply the EJB Client tool that invokes methods of EJB remote objects deployed on a J2EE application server. Sections include:

Understanding the EJB Client Tool

The EJB Client obtains an EJB Remote Object via a JNDI query or from the Object Data Bank. It can then be used to invoke methods on that object. The return values of those methods can then be passed to a chained tool, such as a Diff or an Object Data Bank.

In order to invoke an EJB Client tool on a certain method of an EJB remote object, you must tell the tool the source of the remote object. An EJB remote object can either be obtained from a remote directory via a JNDI query, or from SOAtest's local Object Data Bank. In the latter case, the EJB remote object should be put into the Object Data Bank as a return value from a previous EJB Client tool invocation.

The EJB Client tool must obtain the EJB Remote Object via JNDI query at least once. Subsequently, you can obtain the same EJB Remote Object via the Object Data Bank. This avoids multiple JNDI queries for the same EJB Remote Object.

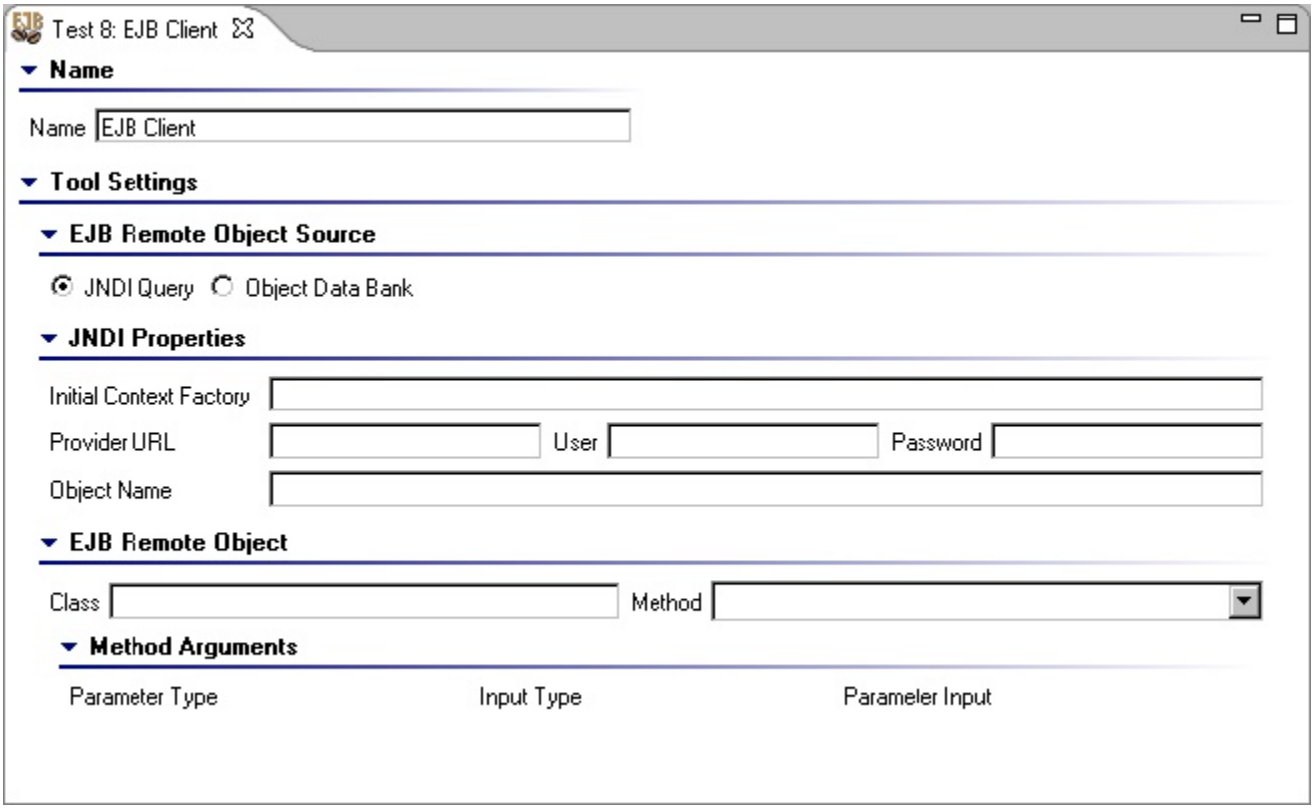
For stateful EJBs in EJB 2.0, use chained Object Data Bank tools to store the returned EJB objects. This enables their methods to be invoked in subsequent tests.

Configuring the EJB Client Tool

The options of the EJB Client tool will vary depending on where the EJB remote object is obtained from (from a **JNDI Query** or from an **Object Data bank**).

Obtaining EJB Remote Object from JNDI Quer

In order to obtain an EJB Remote Object from a JNDI Query, select the **JNDI Query** radio button from the **EJB Remote Object Source** area and configure the following parameters:



- **JNDI Properties**: Allows you to configure the properties for the remote directory from a JNDI Query. The following options are available:

- **Initial Context Factory**: Specifies the context factory, such as `org.jnp.interfaces.NamingContextFactory`.
- **Provider URL**: Specifies the location of the JNDI query, such as `cheetah.parasoft.com`. Also specify the **User** and **Password**.
- **Object Name**: Specifies the name of the object in the JNDI directory, such as `ejb/CartHomeRemote`. This object should be an instance of the class that you enter in the Class field of the EJB Remote Object panel. You can find the JNDI names of the EJB objects bound to the JNDI directory in your J2EE server configuration.

> ⚠ **The Initial Context Factory Class Should be Included in SOAtest's Classpath**
>
> For more information on adding class files to SOAtest, see System Properties Settings.

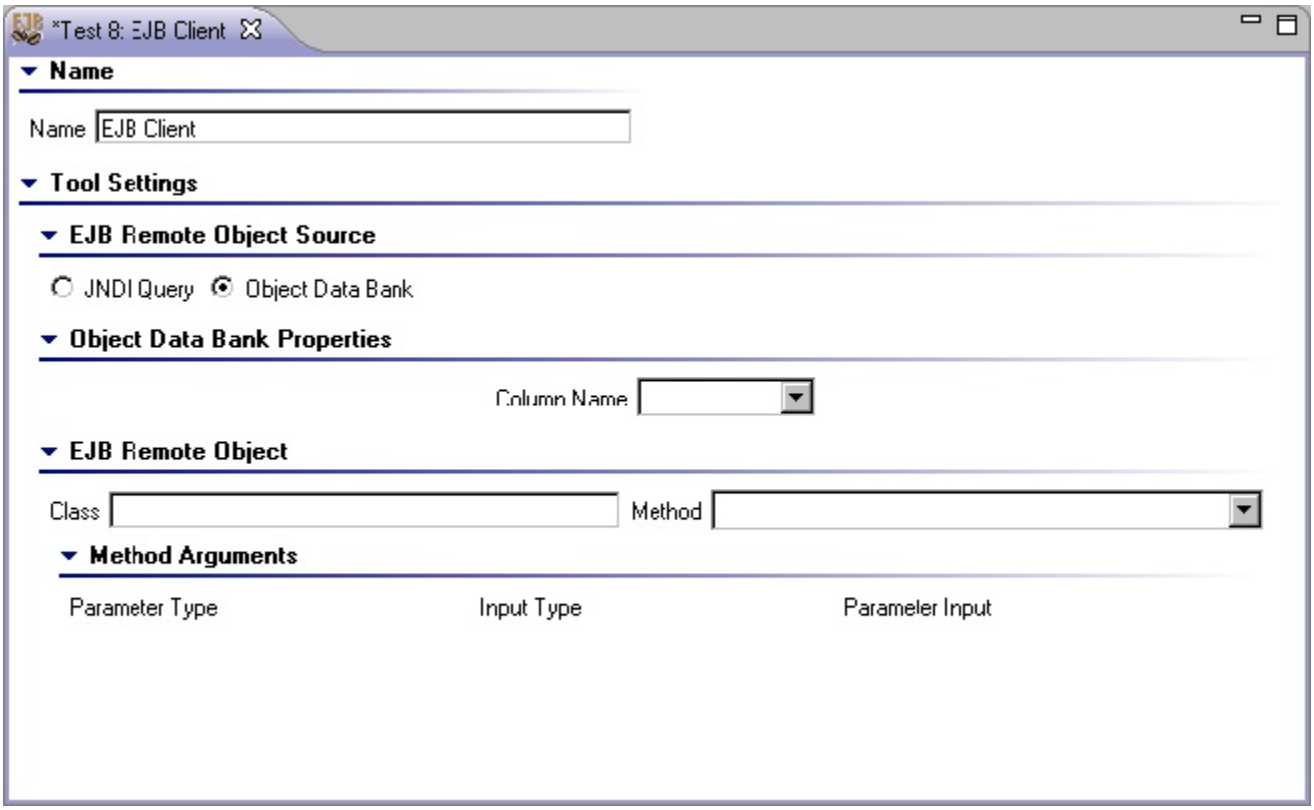
- **EJB Remote Object**: Allows you to configure the properties for invoking a remote method for the EJB Remote Object. The following options are available:
  - **Class**: Specifies the class of the Object that is expected to be returned from either the JNDI query or the Object Data Bank. For instance: `com.parasoft.soatest.bookstore.cart.CartRemote.` If the class is in SOAtest's classpath, the Method selection box will be automatically populated with public methods of this class.
  - **Method**: Specifies the method you would like to invoke. If the class specified in the Class field is found on SOAtest's classpath, the **Method** combo box will be automatically filled with available methods.
  - **Method Arguments**: If the selected method takes arguments, the Method Arguments sub panel will prompt you to specify the **Input Type** and **Parameter Input** for each of the arguments of the remote method.
    - For Java primitive types (java.lang.String and java.util.Date), the following input types are available:
      - **Literal**: Specifies the input value as a string. If for example the **Parameter Type** is Java primitive type `int`, the following will be a valid input `10`, `12345`. For Java float type: `7.62`, `105.3`, etc.
      - **Parameterized**: This input will be available if there is at least one Data Source "visible" to this tool. Select the desired data source column as input.
      - **Scripted**: Specifies the parameter as a return value of a custom method.
    - For the rest of the Java types, the following input types are available
      - **Interpreted**: Allows use of tabular input (CSV, Excel, etc.) to instantiate a Java Object that will be used as a remote method parameter. For more information, see Using Interpreted Data Sources.
      - **Scripted**: Specifies the parameter as a return value of a custom method.

Obtaining An EJB Remote Object from an Object Data Bank

Before you can obtain an EJB Remote Object from an Object Data Bank (so it can be used in subsequent tests), you must first store a return value in an Object Data Bank. To do this:

1. Right-click the EJB Client Tool node and select **Add Output> Object Output> New Output> Object Data Bank**.
2. In the Object Data Bank control panel, specify a unique **Column Name** by which this value can be later queried.

Once a value is stored in an Object Data Bank, select the **Object Data Bank** radio button from the **EJB Remote Object Source** sub panel of the EJB Client tool and configure the following parameters:



- **Object Data Bank Properties**: Allows you to configure the properties for the remote directory from an Object Data Bank. The following options are available:
  - **Column Name**: Select the appropriate column name for the value stored in the Object Data Bank.
- **EJB Remote Object**: Allows you to configure the properties for invoking a remote method for the EJB Remote Object. The following options are available:
  - **Class**: Specifies the class of the Object that is expected to be returned from either the JNDI query or the Object Data Bank. For instance: `com.parasoft.soatest.bookstore.cart.CartRemote`. If the class is in SOAtest's classpath, the Method selection box will be automatically populated with public methods of this class.
  - **Method**: Specifies the method you would like to invoke. If the class specified in the **Class** field is found on SOAtest's classpath, the **Method** combo box will be automatically filled with available methods.
  - **Method Arguments**: If the selected method takes arguments, the Method Arguments sub panel will prompt you to specify the **Input Type** and **Parameter Input** for each of the arguments of the remote method.
    - For Java primitive types (java.lang.String and java.util.Date), the following input types are available:
      - **Literal**: Specifies the input value as a string. If for example the **Parameter Type** is Java primitive type `int`, the following will be a valid input `10, 12345`. For Java float type: `7.62, 105.3`, etc.
      - **Parameterized**: This input will be available if there is at least one Data Source “visible” to this tool. Select the desired data source column as input.
      - **Scripted**: Specifies the parameter as a return value of a custom method.
    - For the rest of the Java types, the following input types are available:
      - **Interpreted**: Allows use of tabular input (CSV, Excel, etc.) to instantiate a Java Object that will be used as a remote method parameter.
      - **Scripted**: Specifies the parameter as a return value of a custom method.

Viewing the Data Bank Variables Used During Test Execution

You can configure the Console view (**Window> Show View> Console**) to display the data bank variables used during test execution. For details, see Console view.

Using Regression Controls

To create or update regression controls for a set of EJB Client tool tests, right-click on the test suite encompassing the tests and select **Update** or **Create Regression Controls** from the shortcut menu. Select **Multiple Controls** if you are using data sources.

Invoking EJBs Deployed on IBM WebSphere Application Server (WAS)

To invoke an EJB deployed on the WebSphere Application Server (WAS) using the Sun JRE shipped with SOAtest, some special configuration is necessary. All other configuration settings explained in this document apply. To configure SOAtest, complete the following:

1. Use the Sun Initial Context Factory (`com.sun.jndi.cosnaming.CNCtxFactory`): This setting should be configured in the EJB Client's JNDI properties. This setting requires that `Sun's j2ee.jar` is on the SOAtest classpath. This `.jar` file is available from the Oracle Java site. For more information on adding class files to SOAtest, see System Properties Settings.
2. In the EJB Clients's JNDI properties, use a provider URL in the form of `corbaloc:iiop:WAS:2809` where `WAS` is the location of your WebSphere Application Server.(for example, `corbaloc:iiop:bison:2809`)
3. In the EJB Client's JNDI properties, specify the Object Name in the following format: `cell/nodes/[node]/servers/[server] /YOUR_EJB_OBJECT_NAME` (for example, `cell/nodes/bisonNode01/servers/server1/ejb/BasicCalculator`)

Using XML Encoder/Decoder with the EJB Client

The XML Encoder and XML Decoder are complementary tools which allow you to transfer a Java Bean object graph into an XML representation (XML Encoder) or transfer a compatible XML output to a Java Bean (XML Decoder).

The XML Encoder and XML Decoder tools can be particularly useful when creating EJB Tests. An XML Encoder can be used to transform the Java Bean Object Output of an EJB Tool to an XML format. This output can be further manipulated by other SOAtest tools, such as XML Transformer, to extract or modify portions of the XML document that are of interest to you. The modified XML output can then be transformed back to Java Bean format with the help of the XML Decoder. The result can be chained to the Object Data Bank—thus making it possible for the subsequent tests to use the modified Java Bean object as a Tool input.

Please keep in mind that the EJB Tool conveniently allows you to add chained Tools to its XML or Object outputs. The XML Output of the EJB Tool is an Object Output transformed by an XML Encoder.