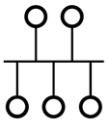

Orbix®

CORBA Tutorial Java

Version 6.3, December 2005

Making Software Work Together™

Updated: 23-Jan-2009

Contents

CHAPTER 1

Getting Started with Orbix

You can use the CORBA Code Generation Toolkit to develop an Orbix application quickly.

Given a user-defined IDL interface, the toolkit generates the bulk of the client and server application code, including build files. You then complete the distributed application by filling in the missing business logic.

In this chapter

This chapter contains the following sections:

Creating a Configuration Domain

Overview

This section describes how to create a simple configuration domain, `simple`, which is required for running basic demonstrations. This domain deploys a minimal set of Orbix services.

Prerequisites

Before creating a configuration domain, the following prerequisites must be satisfied:

- Orbix is installed.
- Some basic system variables are set up (in particular, the `IT_PRODUCT_DIR`, `IT_LICENSE_FILE`, and `PATH` variables).

Fore more details, please consult the *Installation Guide*.

Licensing

The location of the license file, `licenses.txt`, is specified by the `IT_LICENSE_FILE` system variable. If this system variable is not already set in your environment, you can set it now.

Steps

To create a configuration domain, `simple`, perform the following steps:

1. Run itconfigure.
2. Choose the domain type.
3. Specify service startup options.
4. Specify security settings.
5. Specify fault tolerance settings.
6. Select services.
7. Confirm choices.
8. Finish configuration.

Run itconfigure

To begin creating a new configuration domain, enter `itconfigure` at a command prompt. An **Orbix Configuration Welcome** dialog box appears, as shown in Figure 1.

Select **Create a new domain** and click **OK**.

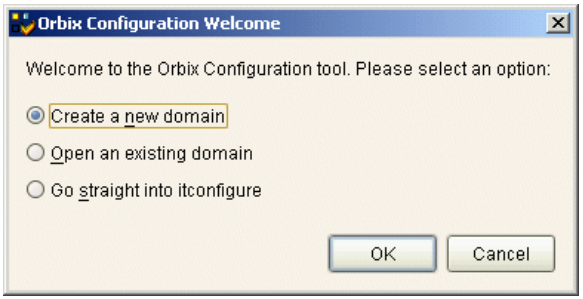

Figure 1: *The Orbix Configuration Welcome Dialog Box*

Choose the domain type

A **Domain Type** window appears, as shown in Figure 2.

In the **Configuration Domain Name** text field, type `simple`. Under **Configuration Domain Type**, click the **Select Services** radiobutton.

Click **Next>** to continue.

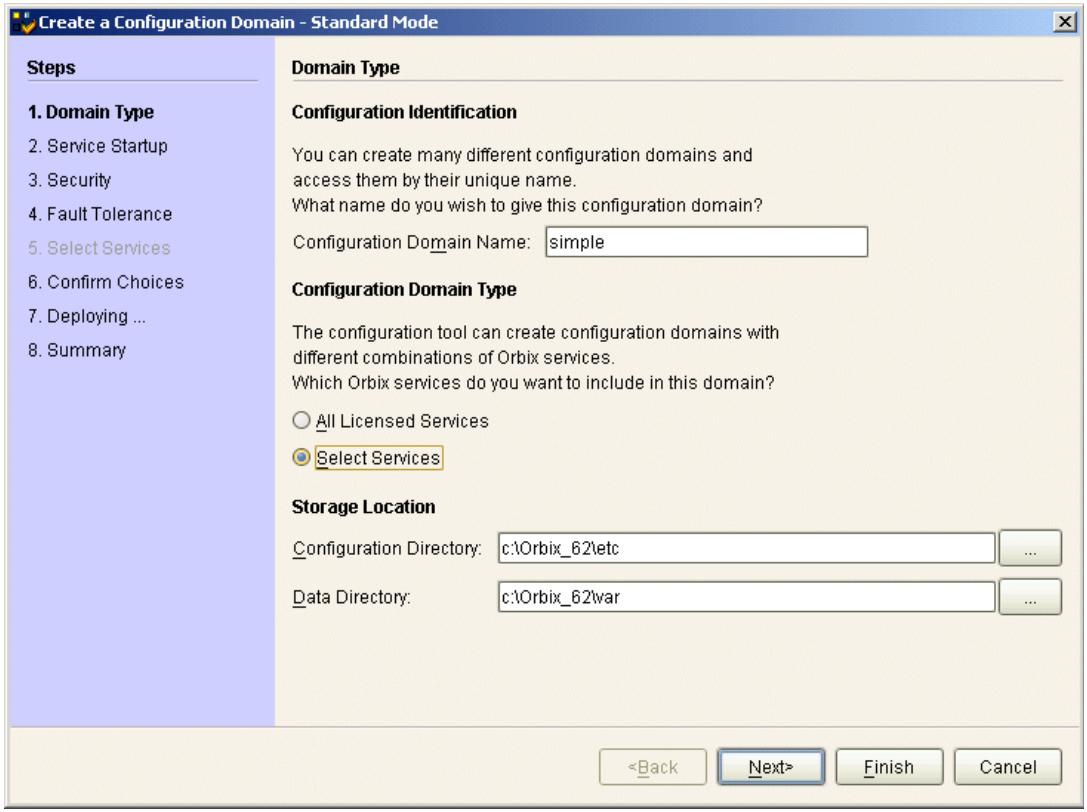

Figure 2: *The Domain Type Window*

Specify service startup options

A **Service Startup** window appears, as shown in Figure 3.

You can leave the settings in this Window at their defaults.

Click **Next>** to continue.

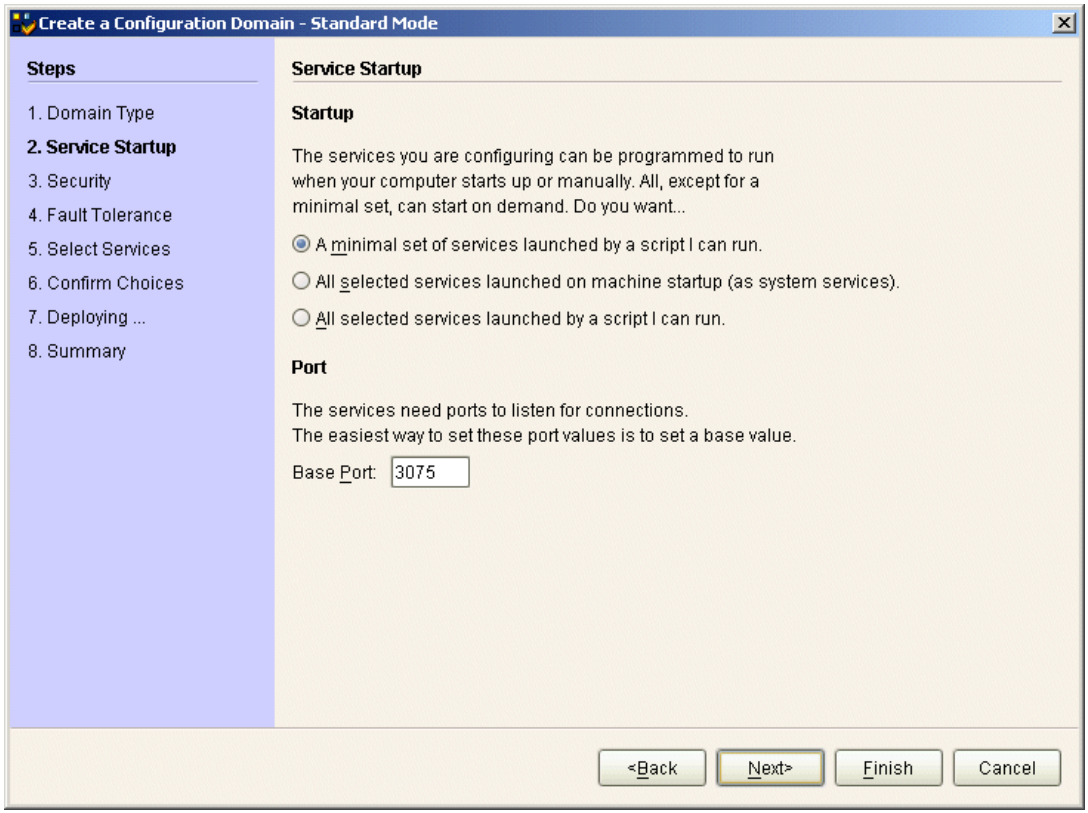

Figure 3: *The Service Startup Window*

Specify security settings

A **Security** window appears, as shown in Figure 4.

You can leave the settings in this Window at their defaults (no security).

Click **Next>** to continue.

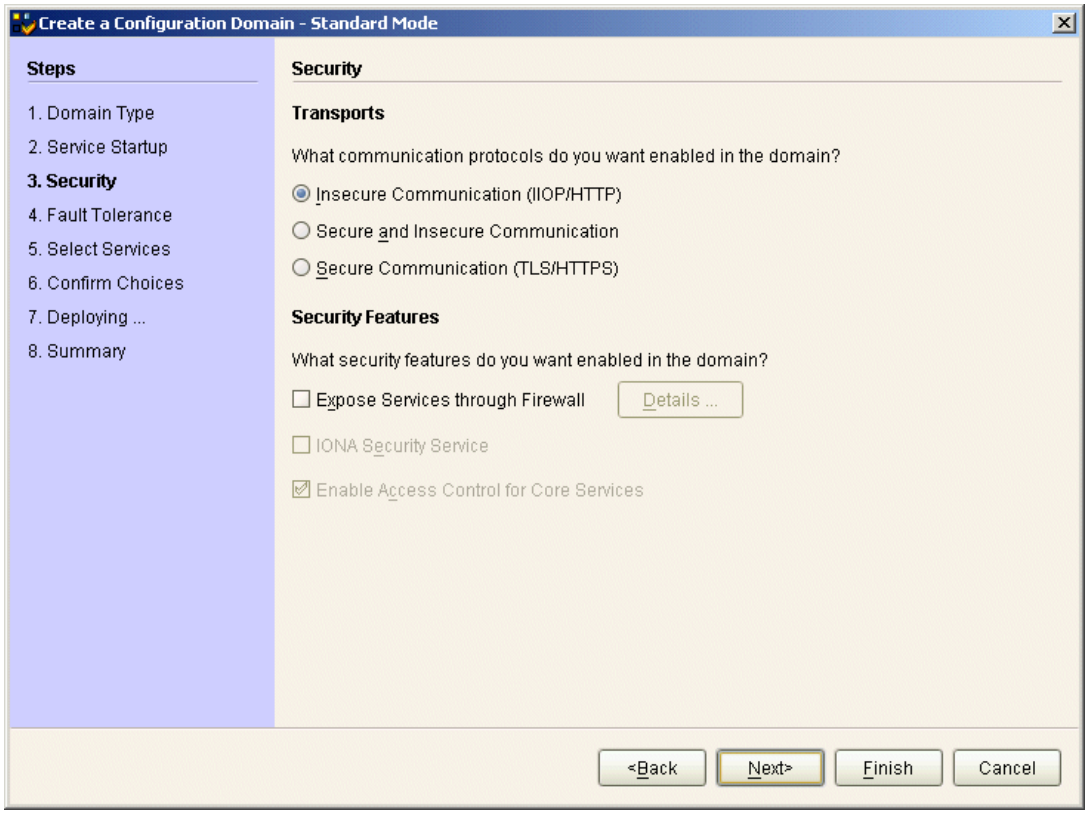

Figure 4: *The Security Window*

Specify fault tolerance settings

A **Fault Tolerance** window appears, as shown in Figure 5.

You can leave the settings in this Window at their defaults.

Click **Next>** to continue.

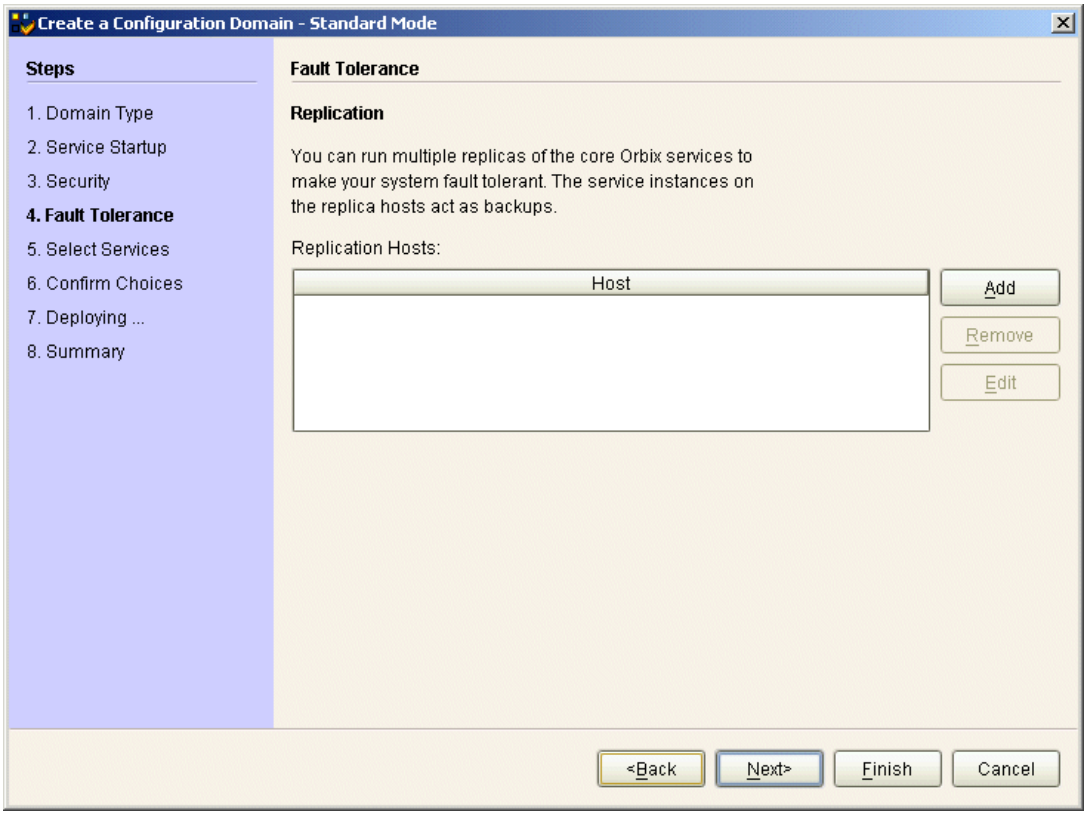

Figure 5: *The Fault Tolerance Window*

Select services

A **Select Services** window appears, as shown in Figure 6.

In the Select Services window, select the following services and components for inclusion in the configuration domain: **Location**, **Node daemon**, **Management**, **CORBA Interface Repository**, **CORBA Naming**, and **demos**.

Click **Next>** to continue.

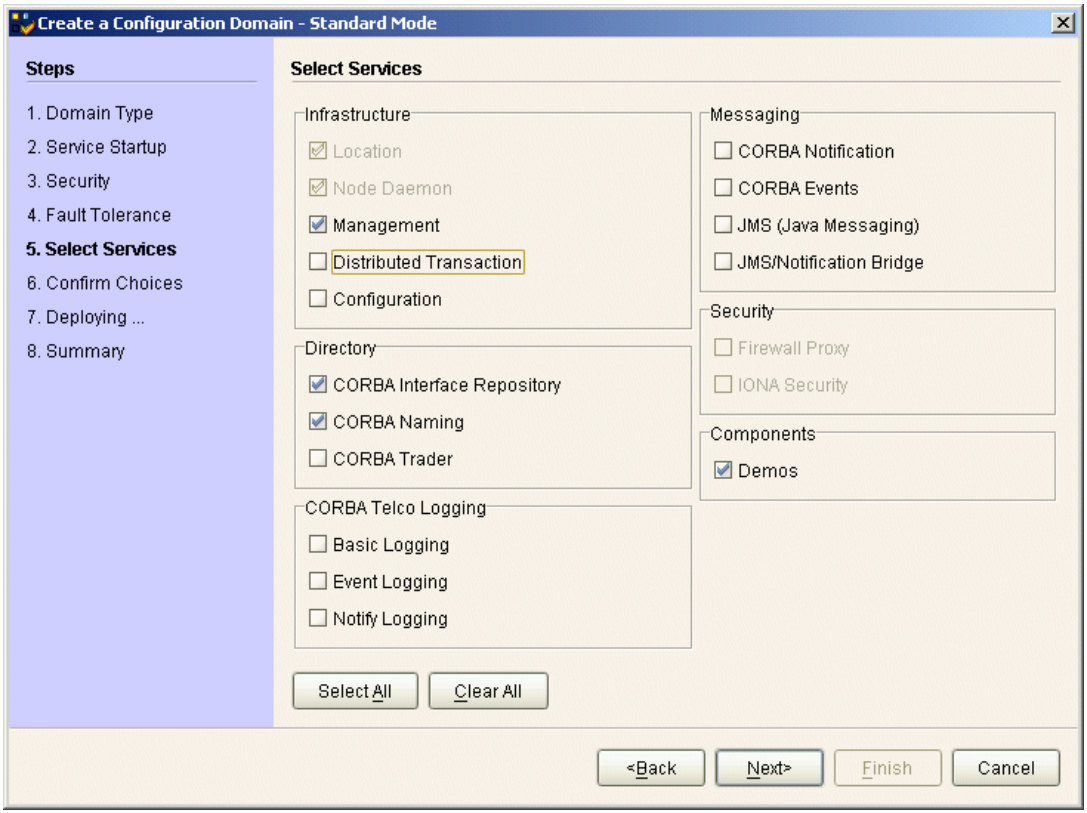

Figure 6: *The Select Services Window*

Confirm choices

You now have the opportunity to review the configuration settings in the **Confirm Choices** window, Figure 7. If necessary, you can use the **<Back** button to make corrections.

Click **Next>** to create the configuration domain and progress to the next window.

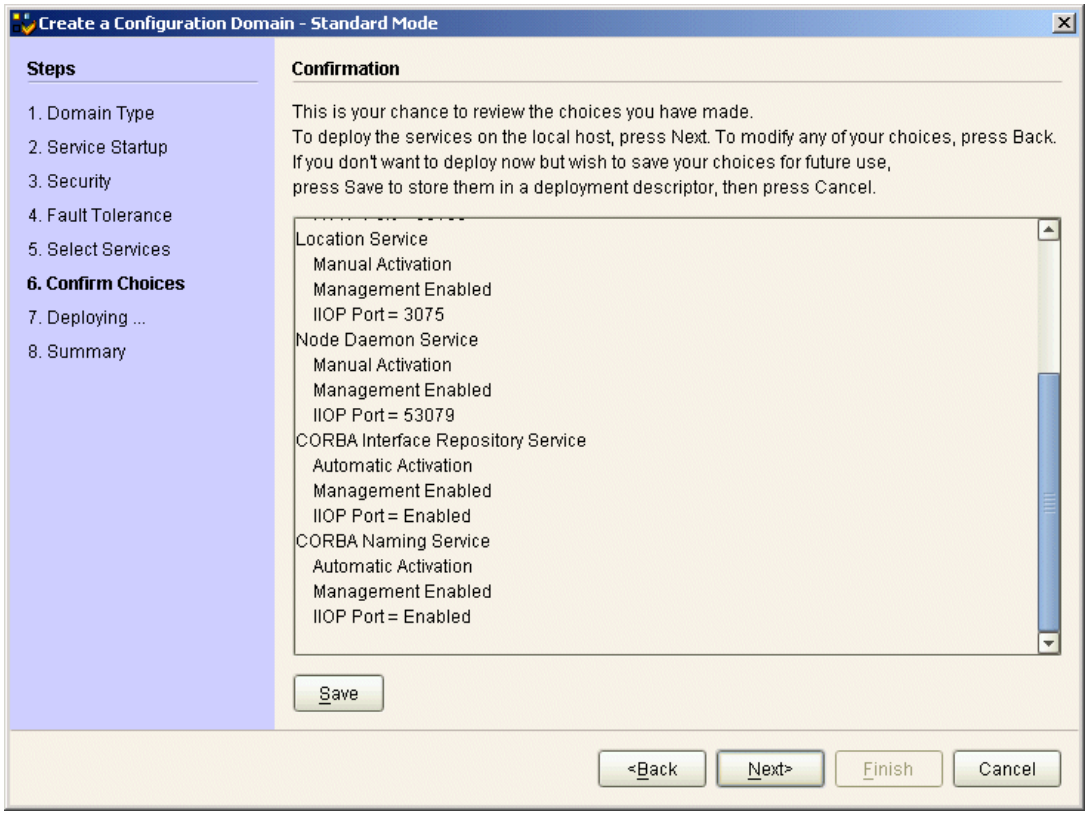

Figure 7: *The Confirm Choices Window*

Finish configuration

The `itconfigure` utility now creates and deploys the `simple` configuration domain, writing files into the *OrbixInstallDir*`/etc/bin`, *OrbixInstallDir*`/etc/domain`, *OrbixInstallDir*`/etc/log`, and *OrbixInstallDir*`/var` directories.

If the configuration domain is created successfully, you should see a **Summary** window with a message similar to that shown in Figure 8.

Click **Finish** to quit the `itconfigure` utility.

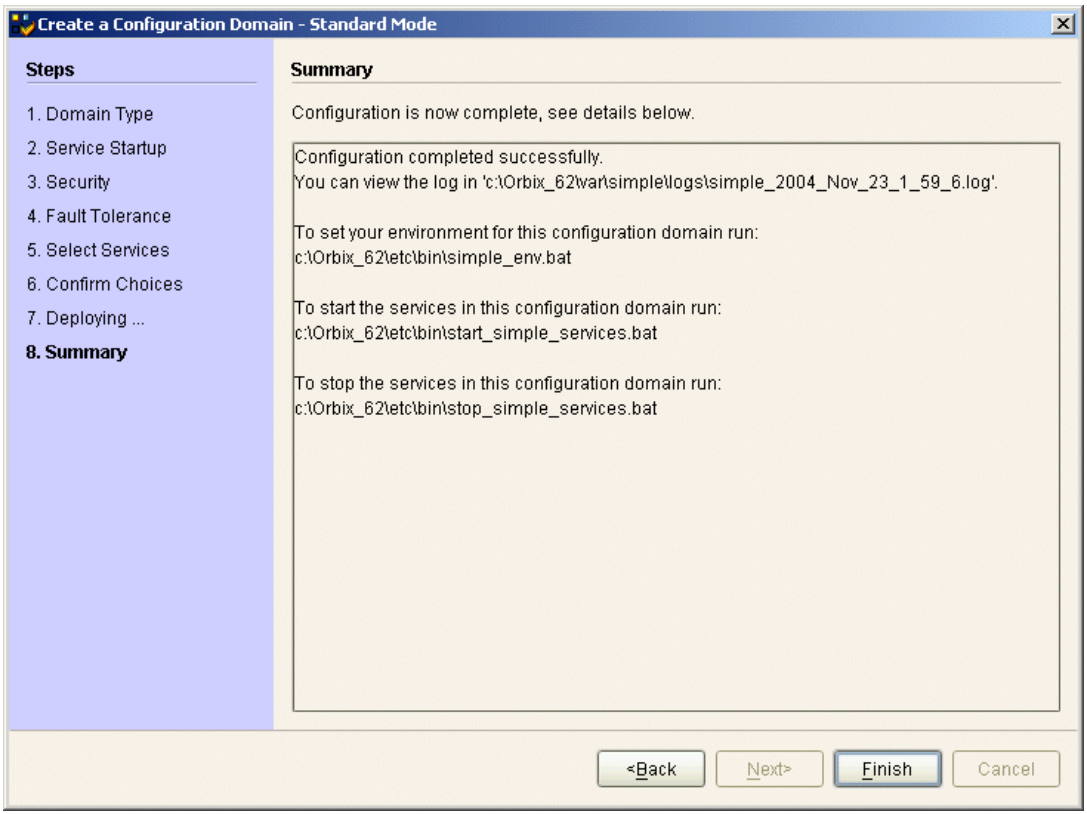

Figure 8: *Configuration Summary*

Setting the Orbix Environment

Prerequisites

Before proceeding with the demonstration in this chapter you need to ensure:

- The CORBA developer's kit is installed on your host.
- Orbix is configured to run on your host platform.
- Your Java development kit (JDK) is configured to use the Orbix ORB runtime (see "Setting ORB Properties for the Orbix ORB" on page 12).
- Your configuration domain is set (see "Setting the domain").

The *Administrator's Guide* contains more information on Orbix configuration, and details of Orbix command line utilities.

Setting the domain

The scripts that set the Orbix environment are associated with a particular *domain*, which is the basic unit of Orbix configuration. See the *Installation Guide*, and the *Administrator's Guide* for further details on configuring your environment.

To set the Orbix environment associated with the *domain-name* domain, enter:

Windows

```
> set JAVA_HOME=YourJdkDir
> config-dir\etc\bin\domain-name_env.bat
```

UNIX

```
% JAVA_HOME=YourJdkDir ; export JAVA_HOME
% . config-dir/etc/bin/domain-name_env
```

YourJdkDir is the root directory of the Java development kit that you want to use with Orbix. See the *Installation Guide* for details of supported Java platforms.

config-dir is the root directory where the Appliation Server Platform stores its configuration information. You specify this directory while configuring your domain. *domain-name* is the name of a configuration domain.

Setting ORB Properties for the Orbix ORB

SUN's Java development kit (JDK) comes with a built-in ORB runtime that is used by default. However, you cannot use SUN's ORB runtime with Orbix applications. You must configure the JDK to use the Orbix ORB runtime instead by setting system properties `org.omg.CORBA.ORBClass` and `org.omg.CORBA.ORBSingletonClass` to the appropriate values. You can set the ORB properties in one of the following ways:

- Using the iona.properties file
- Using Java interpreter arguments

Using the iona.properties file

Setting system properties `org.omg.CORBA.ORBClass` and `org.omg.CORBA.ORBSingletonClass` in the `iona.properties` file is the preferred way to configure your JDK to use the Orbix ORB runtime.

Location of the iona.properties file

The `iona.properties` file is located in the *`JDKHome`*`/jre/lib` directory, where *`JDKHome`* is the JDK root directory.

Contents of the iona.properties file

The `iona.properties` file should contain the following two lines of text:

```
org.omg.CORBA.ORBClass=com.iona.corba.art.artimpl.ORBImpl
org.omg.CORBA.ORBSingletonClass=
    com.iona.corba.art.artimpl.ORBSingleton
```

The first line sets `org.omg.CORBA.ORBClass` to the name of a class that implements `org.omg.CORBA.ORB`.

The second line sets `org.omg.CORBA.ORBSingletonClass` to the name of a class that implements the static ORB instance returned from `org.omg.CORBA.ORB.init()` (taking no arguments).

WARNING: By setting system properties `org.omg.CORBA.ORBClass` and `org.omg.CORBA.ORBSingletonClass` in the `iona.properties` file, as detailed above, you effectively specify the Orbix ORB classes as the ORB runtime for the JDK. This might affect other applications that use the same JDK but want to use different ORB classes—if this is the case, you should consider using one of the alternative mechanisms for setting ORB properties, given in the following sub-sections.

Using Java interpreter arguments

You can use the `-D`*`property_name`*`=property_value` option on the Java Interpreter to specify the `org.omg.CORBA.ORBClass` and `org.omg.CORBA.ORBSingletonClass` properties. For example, to set the ORB properties for an `orbix_app` Orbix application:

```
java -Dorg.omg.CORBA.ORBClass=com.iona.corba.art.artimpl.ORBImpl\
     -Dorg.omg.CORBA.ORBSingletonClass=\
     com.iona.corba.art.artimpl.ORBSingleton orbix_app
```

Setting Your Classpath

Overview

Before building any Orbix Java server or client application, you must ensure that your classpath is configured appropriately for the Orbix features that you wish to use.

Basic Orbix classpath settings

The basic Orbix JAR files that must be included on you classpath are as follows:

```
OrbixInstallDir/lib/art/omg/1.3/omg.jar
OrbixInstallDir/lib/art/art/1.3/art.jar
```

Windows

For example, on Windows, the following command adds these JAR files to your classpath:

```
set CLASSPATH=%CLASSPATH%;%IT_PRODUCT_DIR%/lib/art/omg/1.3/omg.jar;
%IT_PRODUCT_DIR%/lib/art/omg/1.3/art.jar;
```

UNIX

For example, on UNIX, the following command adds these JAR files to your classpath:

```
export CLASSPATH=$CLASSPATH:$IT_PRODUCT_DIR/lib/art/omg/1.3/omg.jar:
   $IT_PRODUCT_DIR/lib/art/art/1.3/art.jar
```

Classpath settings for Orbix features

Other Orbix JAR files might also need to be included on your classpath, depending on which Orbix features your application is using (for example, the naming service or notification service). The following list of JAR files shows typical Orbix features that you may wish to include on your classpath:

```
OrbixInstallDir/lib/platform/java_poa/1.3/poa.jar
OrbixInstallDir/lib/corba/idlgen/5.3/it_genie.jar
OrbixInstallDir/lib/platform/naming_service/1.3/naming.jar
OrbixInstallDir/lib/platform/lease/1.3/lease.jar
```

```
OrbixInstallDir/lib/corba/event_service/5.3/event.jar
OrbixInstallDir/lib/common/ifc/1.3/ifc.jar
OrbixInstallDir/lib/corba/event_service/5.3/event_psk.jar
OrbixInstallDir/lib/corba/messaging_utils/5.3/messaging.jar
OrbixInstallDir/lib/platform/ots/1.3/ots.jar
OrbixInstallDir/lib/corba/notification_service/5.3/notification.jar
OrbixInstallDir/lib/corba/notification_service/5.3/notification_psk.jar
OrbixInstallDir/lib/corba/event_service/5.3/event.jar
OrbixInstallDir/lib/corba/trading_service/5.3/trading.jar
OrbixInstallDir/lib/corba/trading_service/5.3/trading_psk.jar
OrbixInstallDir/lib/corba/basic_log_service/5.3/basic_log.jar
OrbixInstallDir/lib/corba/event_log_service/5.3/event_log.jar
OrbixInstallDir/lib/corba/notification_log_service/5.3/notify_log.jar
OrbixInstallDir/lib/platform/fps/1.3/fps_agent.jar
OrbixInstallDir/lib/platform/java_secure_transports/1.3/tls.jar
OrbixInstallDir/lib/platform/java_transports/1.3/iiop.jar
```

Windows

For example, on Windows, the following command adds the naming service JAR file to your classpath:

```
set CLASSPATH=%CLASSPATH%;%IT_PRODUCT_DIR%/lib/platform/naming_service/1.3/naming.jar;
```

UNIX

For example, on UNIX, the following command adds the naming service JAR file to your classpath:

```
export CLASSPATH=$CLASSPATH:$IT_PRODUCT_DIR/lib/platform/naming_service/1.3/naming.jar
```

Note: The following Orbix JAR file should not be included in your build classpath: *OrbixInstallDir*`/asp/6.3/lib/asp-corba.jar`

Hello World Example

This chapter shows how to create, build, and run a complete client/server demonstration with the help of the CORBA code generation toolkit. The architecture of this example system is shown in Figure 9.

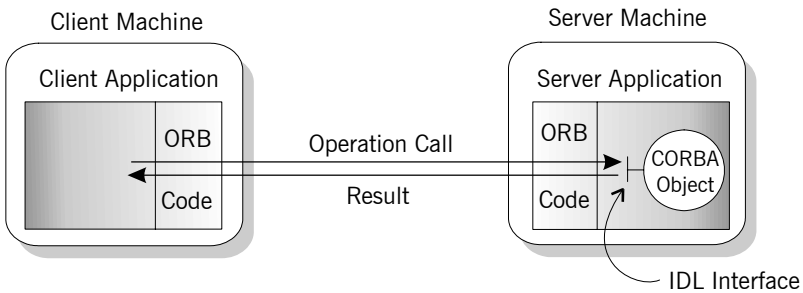

Figure 9: *Client makes a single operation call on a server*

The client and server applications communicate with each other using the Internet Inter-ORB Protocol (IIOP), which sits on top of TCP/IP. When a client invokes a remote operation, a request message is sent from the client to the server. When the operation returns, a reply message containing its return values is sent back to the client. This completes a single remote CORBA invocation.

All interaction between the client and server is mediated via a set of IDL declarations. The IDL for the Hello World! application is:

```
//IDL
interface Hello {
    string getGreeting();
};
```

The IDL declares a single `Hello` interface, which exposes a single operation `getGreeting()`. This declaration provides a language neutral interface to CORBA objects of type `Hello`.

The concrete implementation of the `Hello` CORBA object is written in Java and is provided by the server application. The server could create multiple instances of `Hello` objects if required. However, the generated code generates only one `Hello` object.

The client application has to locate the `Hello` object—it does this by reading a stringified object reference from the file `Hello.ref`. There is one operation `getGreeting()` defined on the `Hello` interface. The client invokes this operation and exits.

Development from the Command Line

Starting point code for CORBA client and server applications can be generated using the `idlgen` command line utility.

The `idlgen` utility can be used on Windows and UNIX platforms.

You implement the `Hello World!` application with the following steps:

1. Define the IDL interface, `Hello`.
2. Generate starting point code.
3. Complete the server program by implementing the single IDL `getGreeting()` operation.
4. Complete the client program by inserting a line of code to invoke the `getGreeting()` operation.
5. Build the demonstration.
6. Run the demonstration.

Define the IDL interface

Create the IDL file for the `Hello World!` application. First of all, make a directory to hold the example code:

Windows

```
> mkdir C:\OCGT\HelloExample
```

UNIX

```
% mkdir -p OCGT/HelloExample
```

Create an IDL file `C:\OCGT\HelloExample\hello.idl` (Windows) or `OCGT/HelloExample/hello.idl` (UNIX) using a text editor.

Enter the following text into the file `hello.idl`:

```
//IDL
interface Hello {
    string getGreeting();
};
```

This interface mediates the interaction between the client and the server halves of the distributed application.

Generate starting point code

Generate files for the server and client application using the CORBA Code Generation Toolkit.

In the directory `C:\OCGT\HelloExample` (Windows) or `OCGT/HelloExample` (UNIX) enter the following command:

```
idlgen java_poa_genie.tcl -all -jP HelloExample hello.idl
```

This command logs the following output to the screen while it is generating the files:

```
hello.idl:
java_poa_genie.tcl: creating idlgen/RandomFuncs.java
java_poa_genie.tcl: creating
   idlgen/HelloExample/RandomHello.java
java_poa_genie.tcl: creating idlgen/RandomHelloExample.java
java_poa_genie.tcl: creating HelloExample/HelloCaller.java
java_poa_genie.tcl: creating HelloExample/client.java
java_poa_genie.tcl: creating HelloExample/HelloImpl.java
java_poa_genie.tcl: creating HelloExample/server.java
java_poa_genie.tcl: creating build.xml
```

You can edit the following files to customize client and server applications:

Client:

`HelloExample/client.java`

Server:

`HelloExample/server.java`
`HelloExample/HelloImpl.java`

Complete the server program

Complete the implementation class, `HelloImpl`, by providing the definition of the `HelloImpl.getGreeting() method`. This Java method provides the concrete realization of the `Hello::getGreeting()` IDL operation.

Edit the `HelloImpl.java` file, and delete most of the generated boilerplate code occupying the body of the `HelloImpl.getGreeting` method Replace it with the line of code highlighted in bold font below:

```
//Java
//File 'HelloImpl.java'
...
   public java.lang.String getGreeting()
   throws org.omg.CORBA.SystemException
   {
      java.lang.String                _result;

      _result = "Hello World!";

      return _result;
   }
...
```

Complete the client program

Complete the implementation of the client `main()` function in the `client.java` file. You must add a couple of lines of code to make a remote invocation of the `getGreeting()` operation on the `Hello` object.

Edit the `client.java` file and search for the line where the `HelloExample.HelloCaller.getGreeting()` method is called. Delete this line and replace it with the line of code highlighted in bold font below:

```
//Java
//File: 'client.java'
...
    try
    {
       ...
       // Exercise interface HelloExample.Hello.
       //
       tmp_ref = read_reference("Hello.ref");
       HelloExample.Hello Hello1 =
          HelloExample.HelloHelper.narrow(tmp_ref);
       System.out.println("Greeting is: " +
   Hello1.getGreeting());

    }
    catch(Exception ex)
    {
       System.out.println("Unexpected CORBA exception: " + ex);
    }
...
```

The object reference `Hello1` refers to an instance of a `Hello` object in the server application. It is already initialized for you.

A remote invocation is made by invoking `getGreeting()` on the `Hello1` object reference. The ORB automatically establishes a network connection and sends packets across the network to invoke the `HelloImpl.getGreeting()` method in the server application.

Build the demonstration

The `itant` utility—a Java-based build tool—is used to build the generated Java code. For more details about `itant`, see http://jakarta.apache.org/ant. The `itant` utility is bundled with Orbix.

The generated file `build.xml` is used to build this demonstration. This file contains the rules for building the Hello World! application in an XML format that is understood by the `itant` utility.

To build the client and server complete the following steps:

1. Open a command line window.
2. Go to the `../OCGT/HelloExample` directory.

3. Enter:

```
> itant
```

Run the demonstration

Run the application as follows:

1. Run the Orbix services (if required).

   If you have configured Orbix to use file-based configuration, no services need to run for this demonstration. Proceed to step 2.

   If you have configured Orbix to use configuration repository based configuration, start up the basic Orbix services.

   Open a DOS prompt in Windows, or `xterm` in UNIX. Enter:

   ```
   start_domain-name_services
   ```

   Where *domain-name* is the name of the configuration domain.

2. Set the Application Server Platform's environment.

   ```
   > domain-name_env
   ```

3. Run the server program.

   Open a DOS prompt, or `xterm` window (UNIX). Enter the following command:

   ```
   itant runserver
   ```

   The server outputs the following lines to the screen:

   ```
   Buildfile: build.xml

   runserver:
       [java] Initializing the ORB
       [java] Writing stringified object reference to Hello.ref
       [java] Waiting for requests...
   ```

   The server performs the following steps when it is launched:

   - It instantiates and activates a single `Hello` CORBA object.
   - The stringified object reference for the `Hello` object is written to the local `Hello.ref` file.

- The server opens an IP port and begins listening on the port for connection attempts by CORBA clients.

4. Run the client program.

   Open a new DOS prompt, or `xterm` window (UNIX). Enter the following command:

   ```
   itant runclient
   ```

   The client outputs the following lines to the screen:

   ```
   Buildfile: build.xml

   runclient:
     [java] Reading stringified object reference from Hello.ref
   Greeting is: Hello World!

   Total time: 3 seconds
   ```

   The client performs the following steps when it is run:

   - It reads the stringified object reference for the `Hello` object from the `Hello.ref` file.
   - It converts the stringified object reference into an object reference.
   - It calls the remote `Hello::getGreeting()` operation by invoking on the object reference. This causes a connection to be established with the server and the remote invocation to be performed.

5. When you are finished, terminate all processes.

   Shut down the server by typing **`Ctrl-C`** in the window where it is running.

6. Stop the Orbix services (if they are running).

   From a DOS prompt in Windows, or `xterm` in UNIX, enter:

   ```
   stop_domain-name_services
   ```

The passing of the object reference from the server to the client in this way is suitable only for simple demonstrations. Realistic server applications use the CORBA naming service to export their object references instead.

Index