Orbix 6.3.7

CORBA Tutorial: C++

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK

http://www.microfocus.com

2014-06-10

Contents

Getting Started with Orbix 1

Creating a Configuration Domain 1

Setting the Orbix Environment 9

Hello World Example 10

Development from the Command Line 11

Index 17

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 makefiles. You then complete the distributed application by filling in the missing business logic.

Creating a Configuration Domain

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**.

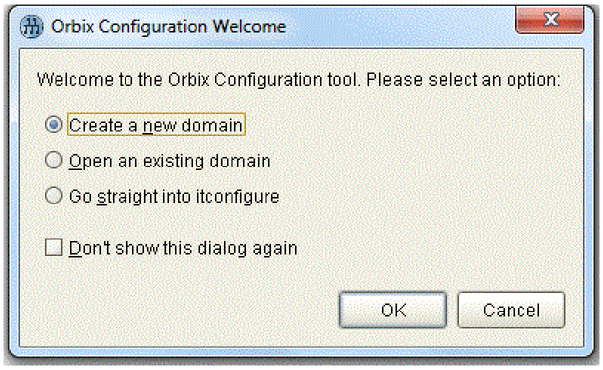

Orbix Configuration Welcome
Welcome to the Orbix Configuration tool. Please select an option:
[✓ Create a new domain]
Open an existing domain
Go straight into itconfigure
Don't show this dialog again
OK
Cancel

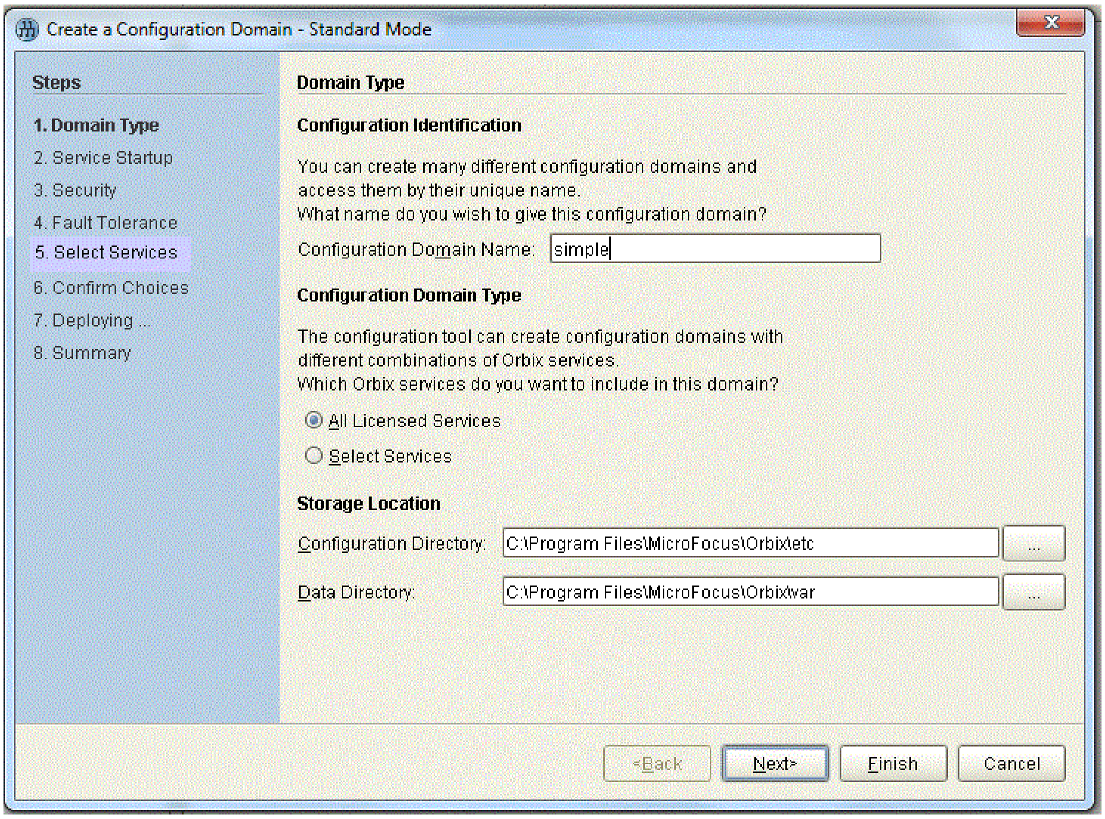
Figure 1: *The Orbix Configuration Welcome Dialog Box*

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.



Configuration Identification

You can create many different configuration domains and access them by their unique name.
What name do you wish to give this configuration domain?

Configuration Domain Name: simple

Configuration Domain Type

The configuration tool can create configuration domains with different combinations of Orbix services.
Which Orbix services do you want to include in this domain?

[✓ All Licensed Services]
Select Services

Storage Location

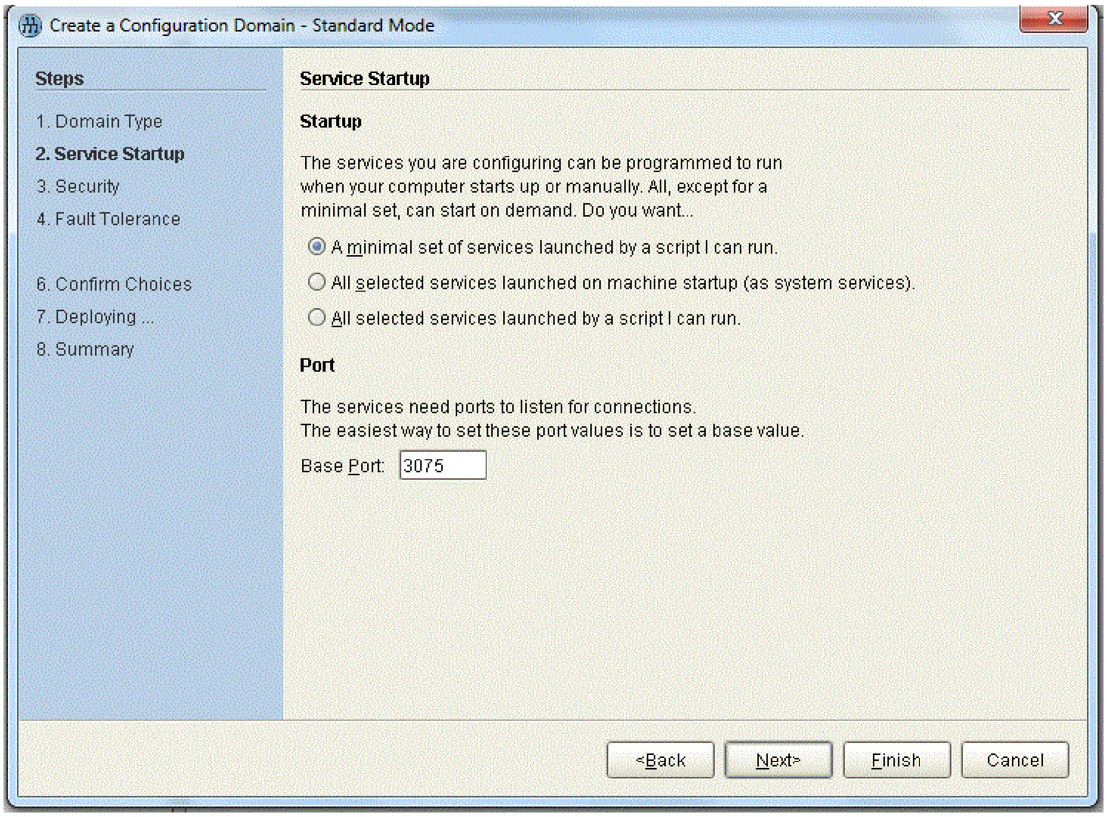
Configuration Directory: C:\Program Files\MicroFocus\Orbix\etc

Data Directory: C:\Program Files\MicroFocus\Orbix\var

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.


Create a Configuration Domain - Standard Mode

Steps

1. Domain Type
2. Service Startup
3. Security
4. Fault Tolerance

6. Confirm Choices
7. Deploying ...
8. Summary

Service Startup

Startup

The services you are configuring can be programmed to run when your computer starts up or manually. All, except for a minimal set, can start on demand. Do you want...

[✓ A minimal set of services launched by a script I can run.]
All selected services launched on machine startup (as system services).
All selected services launched by a script I can run.

Port

The services need ports to listen for connections.
The easiest way to set these port values is to set a base value.

Base Port: 3075

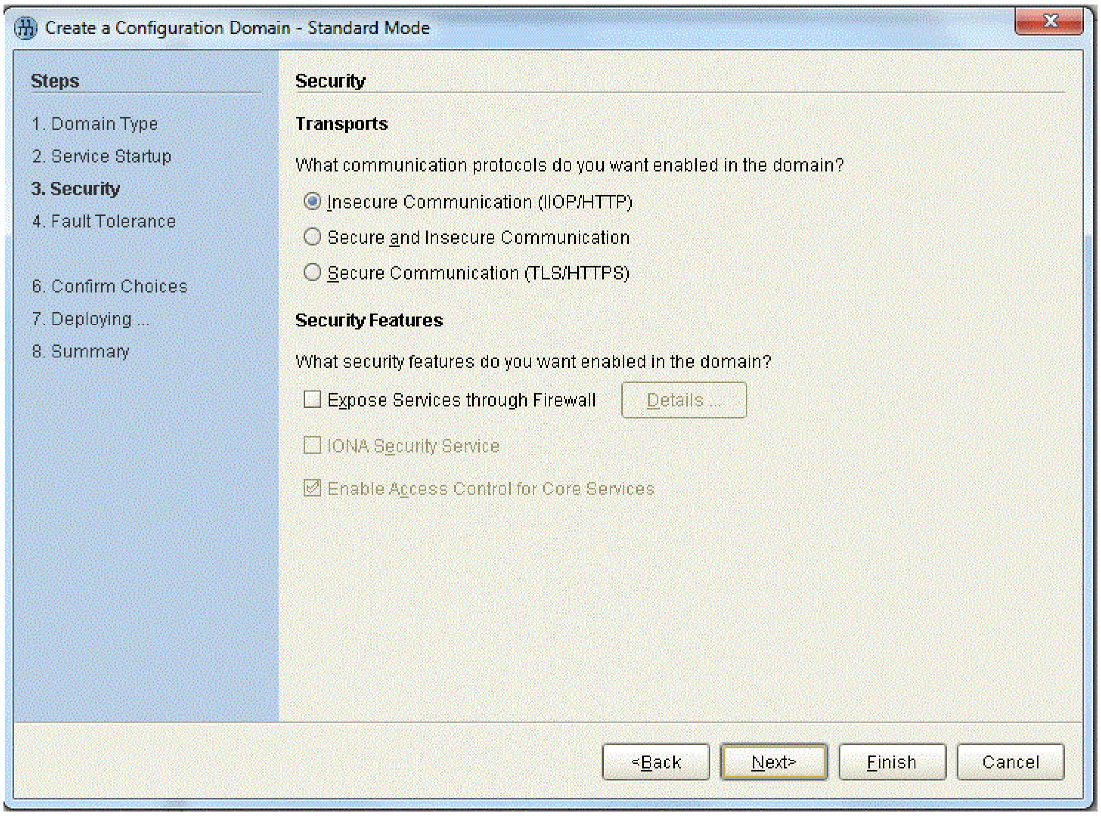
Figure 3: *The Service Startup Window*

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.



Create a Configuration Domain - Standard Mode

Steps

1. Domain Type
2. Service Startup
3. **Security**
4. Fault Tolerance

6. Confirm Choices
7. Deploying ...
8. Summary

Security

Transports

What communication protocols do you want enabled in the domain?

- ● Insecure Communication (IIOP/HTTP)
- ○ Secure and Insecure Communication
- ○ Secure Communication (TLS/HTTPS)

Security Features

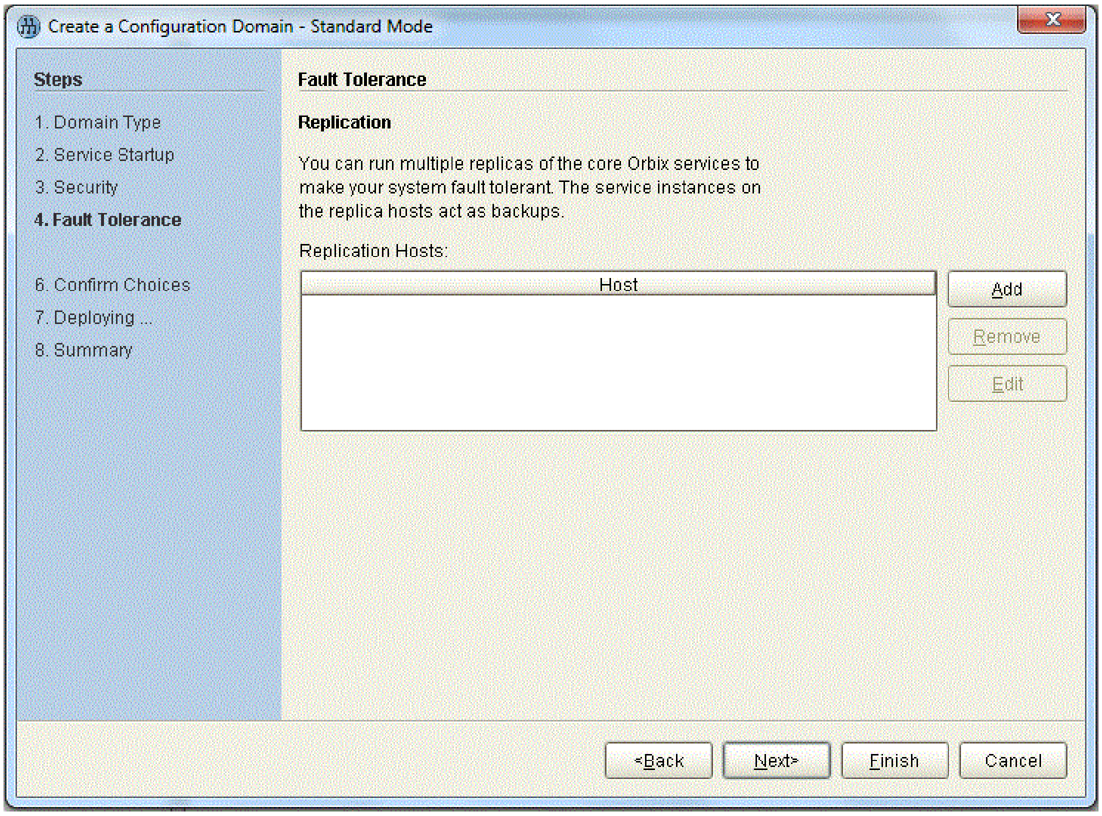
What security features do you want enabled in the domain?

- ☐ Expose Services through Firewall
- ☐ IONA Security Service
- ☑ Enable Access Control for Core Services

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.


Create a Configuration Domain - Standard Mode

Steps

1. Domain Type
2. Service Startup
3. Security
4. **Fault Tolerance**

6. Confirm Choices
7. Deploying ...
8. Summary

Fault Tolerance

Replication

You can run multiple replicas of the core Orbix services to make your system fault tolerant. The service instances on the replica hosts act as backups.

Replication Hosts:

Host

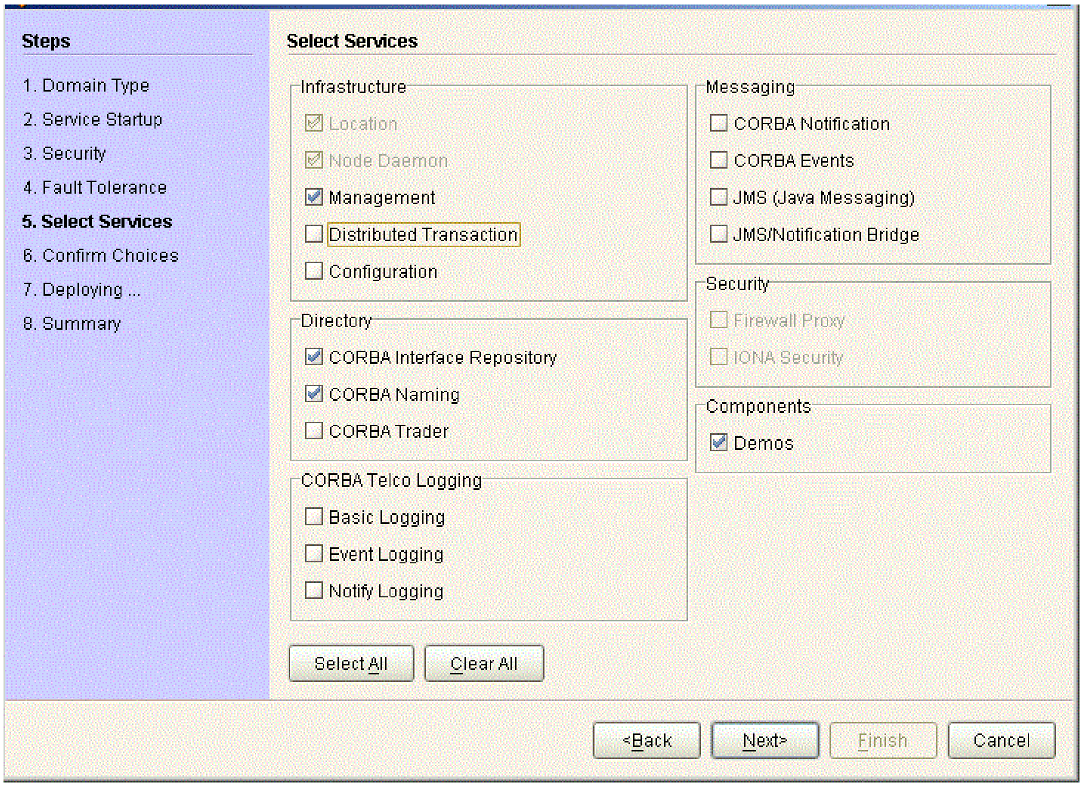
Figure 5: *The Fault Tolerance Window*

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.



Steps

1. Domain Type
2. Service Startup
3. Security
4. Fault Tolerance
5. **Select Services**
6. Confirm Choices
7. Deploying ...
8. Summary

Select Services

Infrastructure
- [✓] Location
- [✓] Node Daemon
- [✓] Management
- [ ] Distributed Transaction
- [ ] Configuration

Directory
- [✓] CORBA Interface Repository
- [✓] CORBA Naming
- [ ] CORBA Trader

CORBA Telco Logging
- [ ] Basic Logging
- [ ] Event Logging
- [ ] Notify Logging

Messaging
- [ ] CORBA Notification
- [ ] CORBA Events
- [ ] JMS (Java Messaging)
- [ ] JMS/Notification Bridge

Security
- [ ] Firewall Proxy
- [ ] IONA Security

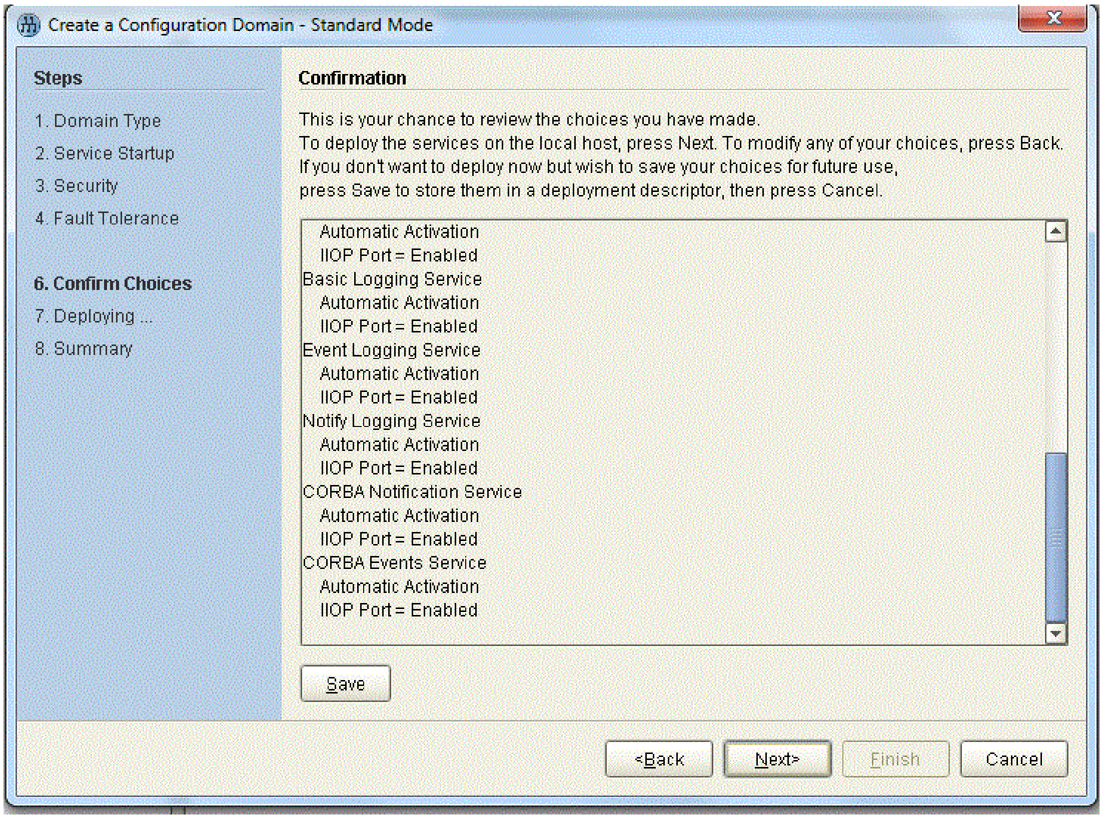
Components
- [✓] Demos

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.



Create a Configuration Domain - Standard Mode

Steps

1. Domain Type
2. Service Startup
3. Security
4. Fault Tolerance

6. Confirm Choices

7. Deploying ...
8. Summary

Confirmation

This is your chance to review the choices you have made.
To deploy the services on the local host, press Next. To modify any of your choices, press Back.
If you don't want to deploy now but wish to save your choices for future use,
press Save to store them in a deployment descriptor, then press Cancel.

```
  Automatic Activation
  IIOP Port = Enabled
Basic Logging Service
  Automatic Activation
  IIOP Port = Enabled
Event Logging Service
  Automatic Activation
  IIOP Port = Enabled
Notify Logging Service
  Automatic Activation
  IIOP Port = Enabled
CORBA Notification Service
  Automatic Activation
  IIOP Port = Enabled
CORBA Events Service
  Automatic Activation
  IIOP Port = Enabled
```

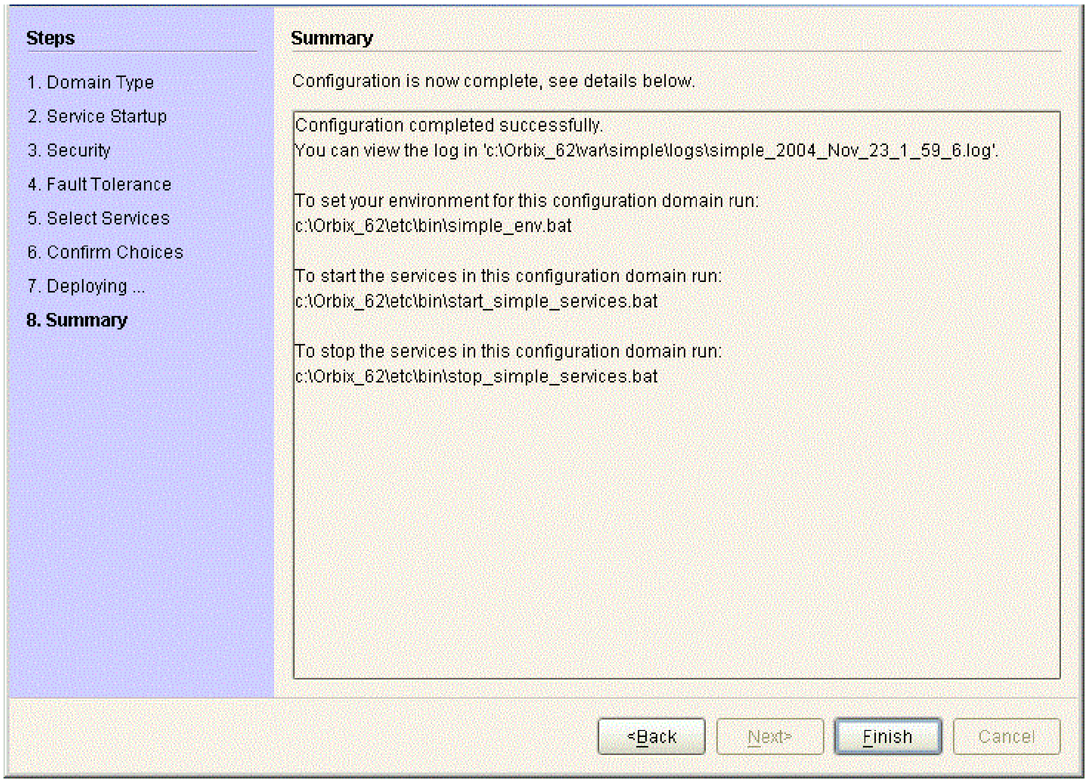
Figure 7: *The Confirm Choices Window*

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.



Configuration is now complete, see details below.

```
Configuration completed successfully.
You can view the log in 'c:\Orbix_62\var\simple\logs\simple_2004_Nov_23_1_59_6.log'.

To set your environment for this configuration domain run:
c:\Orbix_62\etc\bin\simple_env.bat

To start the services in this configuration domain run:
c:\Orbix_62\etc\bin\start_simple_services.bat

To stop the services in this configuration domain run:
c:\Orbix_62\etc\bin\stop_simple_services.bat
```

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 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.

Note: OS/390, both native and UNIX system services, do not support the code generation toolkit and distributed genies. For information about building applications in a native OS/390 environment, see the readme files and JCL that are supplied in the `DEMO` data sets of your iPortal OS/390 Server product installation.

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

```
> config-dir\etc\bin\domain-name_env.bat
```

UNIX

```
% . config-dir/etc/bin/domain-name_env
```

`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.

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.

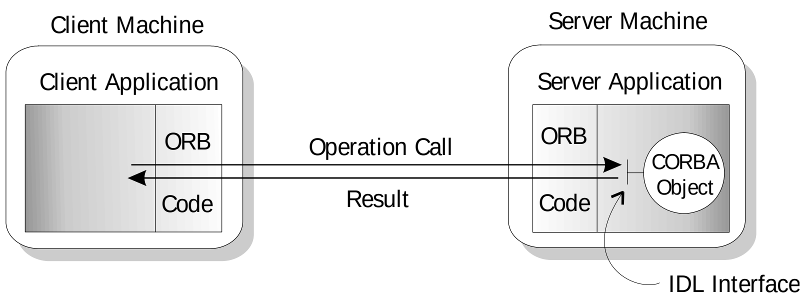


Figure 9: *Client makes a single operation call on a server*

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 C++ 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 also 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 cpp_poa_genie.tcl -all hello.idl
```

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

```
hello.idl:
cpp_poa_genie.tcl: creating it_servant_base_overrides.h
cpp_poa_genie.tcl: creating it_servant_base_overrides.cxx
cpp_poa_genie.tcl: creating HelloImpl.h
cpp_poa_genie.tcl: creating HelloImpl.cxx
cpp_poa_genie.tcl: creating server.cxx
cpp_poa_genie.tcl: creating client.cxx
cpp_poa_genie.tcl: creating call_funcs.h
cpp_poa_genie.tcl: creating call_funcs.cxx
cpp_poa_genie.tcl: creating it_print_funcs.h
cpp_poa_genie.tcl: creating it_print_funcs.cxx
cpp_poa_genie.tcl: creating it_random_funcs.h
cpp_poa_genie.tcl: creating it_random_funcs.cxx
cpp_poa_genie.tcl: creating Makefile
```

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

Client:

`client.cxx`

Server:

`server.cxx`
`HelloImpl.h`
`HelloImpl.cxx`

Complete the server program

Complete the implementation class, `HelloImpl`, by providing the definition of the `HelloImpl::getGreeting() function` . ThisC++ function provides the concrete realization of the `Hello::getGreeting()` IDL operation.

Edit the HelloImpl.cxx file, and delete most of the generated boilerplate code occupying the body of the HelloImpl::getGreeting() function. Replace it with the line of code highlighted in bold font below:

```
//C++
//File 'HelloImpl.cxx'
...
char *
HelloImpl::getGreeting() throw(
    CORBA::SystemException
)
{
    char *                          _result;

    _result = CORBA::string_dup("Hello World!");

    return _result;
}
...
```

The function CORBA::string_dup() allocates a copy of the "Hello World!" string on the free store. It would be an error to return a string literal directly from the CORBA operation because the ORB automatically deletes the return value after the function has completed. It would also be an error to create a copy of the string using the C++ new operator.

Complete the client program

Complete the implementation of the client main() function in the client.cxx 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.cxx file and search for the line where the call_Hello_getGreeting() function is called. Delete this line and replace it with the two lines of code highlighted in bold font below:

```
//C++
//File: 'client.cxx'
...
     if (CORBA::is_nil(Hello1))
     {
          cerr << "Could not narrow reference to interface "
               << "Hello" << endl;
     }
     else
     {
         CORBA::String_var strV = Hello1->getGreeting();
         cout << "Greeting is:  " << strV << endl;
     }
...
```

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()` function in the server application.

The returned string is put into a C++ object, `strV`, of the type `CORBA::String_var`. The destructor of this object will delete the returned string so that there is no memory leak in the above code.

Build the demonstration

The `Makefile` generated by the code generation toolkit has a complete set of rules for building both the client and server applications.

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:

Windows

```
> nmake
```

UNIX

```
% make -e
```

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). From the `C:\OCGT\HelloExample` directory enter the name of the executable file—`server.exe` (Windows) or `server` (UNIX).The server outputs the following lines to the screen:

```
Initializing the ORB
Writing stringified object reference to Hello.ref
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). From the `C:\OCGT\HelloExample` directory enter the name of the executable file—`client.exe` (Windows) or `client` (UNIX).

   The client outputs the following lines to the screen:

```
Client using random seed 0
Reading stringified object reference from Hello.ref
Greeting is: Hello World!
```

   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

A

Application
    running 14

C

Client
    generating 12
    implementing 13
Code generation toolkit
    idlgen utility 12
cpp_poa_genie.tcl 12

H

Hello World! example 10

M

Memory management
    string type 13

O

Object reference
    passing as a string 11

S

Server
    generating 12
    implementing 12
Services 14, 15
string_dup() 13
String_var 14