



# BEA Tuxedo®

## CORBA ActiveX Online Help

Release 8.1  
January 2003

## Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

## Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

## Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

---

# Contents

## About This Online Help

How to Use the Online Help .....	vii
What if the Help System Doesn't Display Properly? .....	viii
Make Sure You Are Using an Up-to-Date Browser .....	1-ix
Customize the Font Size so the Help Is Easy to Read .....	1-ix
Important Considerations About the BEA Builder Installed Browser .....	x
Using Your Favorite Web Browser .....	xi
e-docs Web Site .....	xi
How to Print the Document .....	xii
Related Information .....	xii
Contact Us! .....	xiii
Documentation Conventions .....	xiv

## 1. Overview

What Is ActiveX? .....	1-1
Views and Bindings .....	1-2
How It Works .....	1-2
Naming Conventions for ActiveX Views .....	1-3
OMG IDL .....	1-4
Interface Repository .....	1-5
Domains .....	1-5
Environmental Objects .....	1-6
Bootstrap Object .....	1-7
Factories and the FactoryFinder Object .....	1-8
Naming Conventions and BEA Tuxedo Extensions to the FactoryFinder Object .....	1-9
SecurityCurrent Object .....	1-11

TransactionCurrent Object .....	1-12
InterfaceRepository Object.....	1-13

## **2. Creating ActiveX Client Applications**

Summary of the Development Process for ActiveX Client Applications .....	2-2
The BEA Application Builder .....	2-3
Step 1: Loading the Automation Environmental Objects into the Interface Repository.....	2-5
Step 2: Loading the CORBA Interfaces into the Interface Repository .....	2-5
Step 3: Starting the Interface Repository Server Application .....	2-6
Step 4: Creating ActiveX Bindings for the CORBA Interfaces .....	2-7
Step 5: Loading the Type Library for the ActiveX Bindings .....	2-8
Step 6: Writing the ActiveX Client Application .....	2-9
Including Declarations for the Automation Environmental Objects, Factories, and ActiveX Views of CORBA Objects .....	2-9
Establishing Communication with the BEA Tuxedo Domain .....	2-10
Obtaining References to the FactoryFinder Object .....	2-11
Using a Factory to Get an ActiveX View.....	2-11
Invoking Operations on the ActiveX View .....	2-12
Creating an Automation Server for Callbacks.....	2-13
Creating Instances of the COM Objects.....	2-14
Step 7: Deploying the ActiveX Client Application .....	2-15

## **3. Application Builder Main Window**

Application Builder Main Window .....	3-1
Services Window .....	3-3
Workstation Views Window .....	3-3
Application Builder Objects .....	3-4
Menu Options .....	3-6
File Menu Options .....	3-6
Edit Menu Options .....	3-6
View Menu Options .....	3-7
Tools Menu Options .....	3-8
Window Menu Options .....	3-8
Help Menu Options .....	3-9
Toolbar Buttons .....	3-10

---

## 4. Tasks

Loading CORBA Interfaces into the Interface Repository .....	4-1
Starting Application Builder.....	4-2
Creating ActiveX Bindings for CORBA Interfaces .....	4-3
Changing the Settings for Creating ActiveX Bindings for CORBA Interfaces	4-4
Creating Deployment Packages.....	4-6
Changing the Directory Location for Deployment Packages.....	4-7
Changing the Default Directory Locations.....	4-7
Filtering Objects Displayed in the Main Window.....	4-8
Displaying Properties .....	4-9

## 5. Using Security

Overview of BEA Tuxedo Security .....	5-1
Summary of the Development Process for Security.....	5-2
Step 1: Using the Bootstrap Object to Obtain the SecurityCurrent Object .....	5-2
Step 2: Getting the PrincipalAuthenticator Object from the SecurityCurrent Object .....	5-3
Step 3: Obtaining the Authentication Level .....	5-3
Step 4: Logging On to the BEA Tuxedo Domain with Proper Authentication.	5-4
Step 5: Logging Off the BEA Tuxedo Domain.....	5-5

## 6. Using Transactions

Overview of Transactions.....	6-1
Summary of the Development Process for Transactions .....	6-1
Step 1: Using the Bootstrap Object to Obtain the TransactionCurrent Object..	6-2
Step 2: Using the TransactionCurrent Methods .....	6-3

## 7. Command-Line Options

### Glossary

### Index



---

# About This Online Help

This help guide describes how to use the BEA Application Builder to develop ActiveX client applications.

This document includes the following sections:

- How to Use the Online Help
- What if the Help System Doesn't Display Properly?
- Important Considerations About the BEA Builder Installed Browser
- Using Your Favorite Web Browser
- e-docs Web Site
- How to Print the Document
- Related Information
- Contact Us!
- Documentation Conventions

## How to Use the Online Help

You need to open Application Builder to get Help on using the application.

To bring up Help on a main topic, choose **Help** on the Application Builder main window and select any of the following menu topics:

- 
- Overview—explains what Rose Expert is and gives an overview of the application development tasks you can accomplish with the Rose Expert Application Builder GUI.
  - Creating ActiveX Client Applications—a procedural view of building ActiveX client applications using Application Builder.
  - Application Builder Main Window—an explanation of the various components of the Application Builder Main Window.
  - Tasks—explains how to use the various task windows. (You can also access this help information for a particular window by pressing F1 while that window is open.)
  - Using Security—describes how to use security in ActiveX client applications.
  - Using Transactions—describes how to use transactions in ActiveX client applications.
  - Command-line Options—describes the command-line version of the Application Builder.
  - Glossary—provides an explanation of relevant BEA Tuxedo, BEA Builder, Application Builder, ActiveX and object oriented development terms.
  - About Application Builder—provides version and copyright information.

You can also click the Help button on any task window that is currently open.

## What if the Help System Doesn't Display Properly?

The Help system relies on the Netscape Navigator for its functionality. Therefore, display problems are generally related to what version of the Netscape browser is active on your system, and the font size preference settings on that browser. Additionally, some specific problems on UNIX platforms (such as Help not displaying or the search feature not working) are generally related to incomplete user PATH and CLASSPATH environment variable settings.

---

The following topics provide some troubleshooting tips on problems that can affect various aspects of Help start-up and display:

- Make Sure You Are Using an Up-to-Date Browser
- Customize the Font Size so the Help Is Easy to Read

## Make Sure You Are Using an Up-to-Date Browser

The context-sensitive Help system requires that Netscape Navigator version 4.0 or above be present on the local system and in use. If you are using an earlier version of the Netscape browser, you will get an error message when you try to use the Find or Print buttons.

Note that even if you have Navigator 4.0 installed, you can still get this error if you also have *earlier versions* of the Netscape browser and one of these earlier versions was the last browser used. The remedy for this problem is to close out of the current BEA Builder application, close any earlier versions of the Netscape browser (if you have some open) and open Navigator 4.0. (You can close Navigator 4.0 as soon as you have opened it.)

When you restart the Builder application, the Help system should work properly.

**Note:** If you want to view the online help information in a Web browser, keep in mind that older versions of browsers may not support some of the features built into the HTML Help files. Therefore, we recommend using Internet Explorer version 4.0 or above, Netscape Navigator version 4.0 or above, or other browsers with equivalent HTML support. For information on how to access the Help information in any Web browser, see the section Using Your Favorite Web Browser.

## Customize the Font Size so the Help Is Easy to Read

The context-sensitive Help system relies on your Netscape browser font preference settings. If the information shown in the Help system is difficult to read because the print is too small (or too large), you can change the font size. To do this, simply reset your font preferences in the Netscape Navigator browser. The fonts sizes and styles you set in the browser also will show up in the Help system.

---

If you have more than one version of the Netscape browser on your system, make sure you set the font preferences in the active browser (which is preferably the most up-to-date browser). The Help system uses the last active browser. If you might be using more than one browser version to view Help files, set preferences in all browsers for optimal readability.

For more information about why it is important to use an up-to-date browser, refer to the section *Make Sure You Are Using an Up-to-Date Browser*.

## Important Considerations About the BEA Builder Installed Browser

If you did not have the Netscape Navigator on your system when you installed the BEA Builder products, it is likely that you have a BEA Builder installed version of this browser.

The context-sensitive Help system requires that Netscape Navigator version 4.0 or above be present on the local system. So, the BEA Builder product installation checks to see if the Netscape Navigator 4.0 browser is already present on the target system. On Windows systems, if the appropriate version of the browser is not found; the install script gives you the option of installing it as a part of the BEA Builder product installation to support the online Help system.

The Netscape Navigator 4.0 that gets installed during the BEA Builder product installation contains a level of encryption that is allowed to be exported from the United States. If you use this browser for anything other than the Help system, please note that this is not the most secure version of the Netscape Navigator.

**Note:** This consideration does not apply to UNIX systems because the BEA product installation for UNIX does not automatically install the right version of the browser. You have to do this manually on UNIX systems.

---

# Using Your Favorite Web Browser

The ActiveX Application Builder graphical user interface (GUI) is designed and configured to use Netscape NetHelp as an HTML-based, context-sensitive Help solution.

However, you can also view the online Help for BEA Tuxedo ActiveX Client with the Microsoft Internet Explorer 4.0 browser, Netscape Navigator or Communicator 4.0, or any other Web browser that supports HTML 3.0 and above. If you choose to use the Internet Explorer (or some other browser) to view the Builder documentation, the primary difference is that you will not get the context-sensitive menu and dialog access that you do when you view the Help by means of the GUIs. You may find some discrepancies in display since browsers other than the Netscape NetHelp viewer are not officially supported for this documentation set.

To view the BEA Tuxedo ActiveX Client documentation with a Web browser, open the following file in the browser:

*YourDrive*:tuxdir\help\AppBuilderHtm\default.htm (Windows)

**Note:** Older versions of browsers may not support some of the features built into the HTML help files. Therefore, we recommend using Internet Explorer version 4.0 or above, Netscape Navigator version 4.0 or above, or other browsers with equivalent HTML support.

## e-docs Web Site

BEA Tuxedo online documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the “e-docs” Product Documentation page at <http://e-docs.bea.com/>.

---

# How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser. (Be sure to first click anywhere within the HTML content frame you want to print, so that that frame is selected.)

The information in this online help is also available as the *Using CORBA ActiveX* document in the BEA Tuxedo online documentation. *Using CORBA ActiveX* is available as a PDF file. You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the BEA Tuxedo documentation Home page, click the PDF files button and select the document you want to print.

Also, a PDF version of this online Help is made available on your system in the following location:

*YourDrive*:tuxdir\help\app\_builder\_help.pdf (Windows)

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at <http://www.adobe.com/>.

## Related Information

The following documentation will be helpful in understanding the BEA Tuxedo system. BEA Tuxedo documentation is shipped with your BEA Tuxedo software on a separate documentation CD and is also available on the BEA Web site at <http://e-docs.beasys.com/>.

- *Getting Started with BEA Tuxedo CORBA Applications*. This document introduces BEA Tuxedo CORBA—components and APIs and how to get started with the application development process.
- *Creating CORBA Server Applications*. Describes how C++ programmers can implement key features in the BEA Tuxedo product to design and implement scalable, high-performance CORBA C++ server applications that run in a BEA Tuxedo domain.

- 
- *CORBA Programming Reference*. Provides reference material about the CORBA programming environment, including the OMG IDL syntax, the Interface Configuration File, and the `buildobjserver` command.
  - *Guide to the CORBA University Sample Applications*. Provides information on building and running the University sample client and server applications included with BEA Tuxedo.

## Contact Us!

Your feedback on the BEA Tuxedo documentation is important to us. Send us e-mail at [docsupport@bea.com](mailto:docsupport@bea.com) if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the BEA Tuxedo documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA Tuxedo 8.0 release.

If you have any questions about this version of BEA Tuxedo, or if you have problems installing and running BEA BEA Tuxedo, contact BEA Customer Support through BEA WebSUPPORT at <http://www.bea.com>. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

---

# Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
<b>boldface text</b>	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
<i>italics</i>	Indicates emphasis or book titles.
monospace text	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard. <i>Examples:</i> <pre>#include &lt;iostream.h&gt; void main ( ) the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
<b>monospace boldface text</b>	Identifies significant words in code. <i>Example:</i> <pre>void <b>commit</b> ( )</pre>
<i>monospace italic text</i>	Identifies variables in code. <i>Example:</i> <pre>String <i>expr</i></pre>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> <pre>LPT1 SIGNON OR</pre>

---

Convention	Item
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[ ]	Indicates optional items in a syntax line. The brackets themselves should never be typed. <i>Example:</i> buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...
	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	Indicates one of the following in a command line: <ul style="list-style-type: none"> <li>■ That an argument can be repeated several times in a command line</li> <li>■ That the statement omits additional optional arguments</li> <li>■ That you can enter additional parameters, values, or other information</li> </ul> The ellipsis itself should never be typed. <i>Example:</i> buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

---



# 1 Overview

This Help topic provides an overview of the ActiveX client application development process and the concepts that you need to understand before you develop ActiveX client applications for the BEA Tuxedo CORBA environment.

The following sections are included:

- What Is ActiveX?
- How It Works
- Naming Conventions for ActiveX Views
- OMG IDL
- Interface Repository

## What Is ActiveX?

ActiveX is a set of technologies from Microsoft that enables software components to interact with one another in a networked environment, regardless of the language in which the components were created. ActiveX is built on the Component Object Model (COM) and integrates with Object Linking and Embedding (OLE). OLE provides an architecture for document embedding. Automation is the part of COM that allows applications such as Visual Basic, Delphi, and PowerBuilder to manipulate Automation objects, ActiveX controls, and ActiveX documents.

The BEA ActiveX Client provides interoperability between the BEA Tuxedo and COM object systems. The ActiveX Client transforms the interfaces of CORBA objects in a BEA Tuxedo domain into methods on Automation objects.

## Views and Bindings

ActiveX client applications use views of CORBA interfaces. Views represent the CORBA interfaces in a BEA Tuxedo domain locally as Automation objects. To use an ActiveX view of a CORBA object (referred to as an ActiveX view), you need to create a binding for ActiveX. The binding describes the interface of a CORBA object to ActiveX. The interfaces of the CORBA objects are loaded into the Interface Repository. You then use the BEA ActiveX Application Builder to create Automation bindings for the interfaces.

The combination of the ActiveX client application and the generated binding creates the ActiveX view of the object.

## How It Works

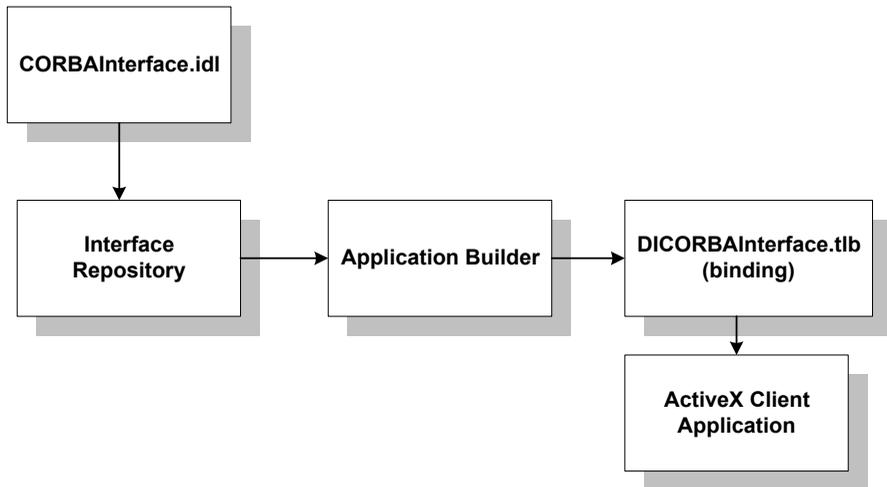
The BEA ActiveX Client makes it possible for ActiveX client applications to interact with CORBA objects in a BEA Tuxedo domain. The ActiveX client application uses the Automation environmental objects to access CORBA objects in the BEA Tuxedo domain. The ActiveX Client creates ActiveX views of the CORBA objects. The ActiveX views of CORBA objects convert and forward all requests they receive from ActiveX client applications to the appropriate CORBA object in the BEA Tuxedo domain.

The Application Builder is a development tool that you use along with a client development tool (such as Visual Basic) to select which CORBA objects in a BEA Tuxedo domain you want your ActiveX client application to interact with.

The Application Builder is the primary user interface to the ActiveX Client. The Application Builder can be used to select which CORBA objects are available to desktop applications, to create ActiveX views of the CORBA objects, and to create packages for deploying ActiveX views of CORBA objects to client computers.

Figure 1-1 illustrates how the ActiveX Client works.

Figure 1-1 How the ActiveX Client Works



## Naming Conventions for ActiveX Views

Naming conventions describe an algorithm for mapping CORBA interfaces to ActiveX to avoid type and variable name conflicts. Naming conventions also indicate how to use a given object. The names of all ActiveX methods begin with `DI`.

The ActiveX Client observes this naming convention when it creates Automation bindings for CORBA interfaces. If a CORBA interface has the name `Account`, the Automation binding for that interface has the name `DIAccount`.

CORBA interface names are often scoped within nested levels known as modules; however, in ActiveX, there is no scoping. To avoid name conflicts, the ActiveX Client exposes a CORBA interface into ActiveX with the name of the different scopes prepended to the name of the interface.

For example, a CORBA interface named `Account` is defined in the OMG IDL file as:

```
module University
{
    module Student
```

```
{
    interface Account
        { //Operations and attributes of the Account interface
        };
};
};
```

In CORBA, this interface is named `University::Student::Account`. The ActiveX Client translates this name to `DIUniversity_Student_Account` for ActiveX.

ActiveX client applications use OLE Automation environmental objects to access CORBA objects in a BEA Tuxedo domain. ActiveX client applications use the BEA ActiveX Client to process requests to CORBA objects. You use the ActiveX Application Builder to select the CORBA interfaces that are available to ActiveX client applications, to create ActiveX views of the CORBA interfaces, and to create packages for deploying ActiveX views of CORBA interfaces to client machines. These client applications are built using an automation development tool such as Visual Basic or PowerBuilder.

## OMG IDL

With any distributed application, the client/server application needs some basic information to communicate. For example, the client application needs to know which operations it can request, and the arguments to the operations.

You use the Object Management Group (OMG) Interface Definition Language (IDL) to describe available CORBA interfaces to client applications. An interface definition written in OMG IDL completely defines the CORBA interface and fully specifies each operation's arguments. OMG IDL is a purely declarative language. This means that it contains no implementation details. Operations specified in OMG IDL can be written in and invoked from any language that provides CORBA bindings.

Generally, the application designer provides the OMG IDL files for the available CORBA interfaces and operations to the programmer who creates the client applications.

# Interface Repository

The Interface Repository contains descriptions of a CORBA object's interfaces and operations. The information stored in the Interface Repository is equivalent to the information defined in an OMG IDL file, but the information is accessible programmatically at run time.

ActiveX client applications are not aware that they are using the Interface Repository. The BEA ActiveX Client uses CORBA operations to obtain information about CORBA objects from the Interface Repository.

You use the following BEA Tuxedo development commands to manage the Interface Repository:

- The `idl2ir` command populates the Interface Repository with CORBA interfaces. This command creates an Interface Repository if an Interface Repository does not exist. Also use this command to update the CORBA interfaces in the Interface Repository.
- The `ir2idl` command creates an OMG IDL file from the contents of the Interface Repository.
- The `irdel` command deletes CORBA interfaces from the Interface Repository.

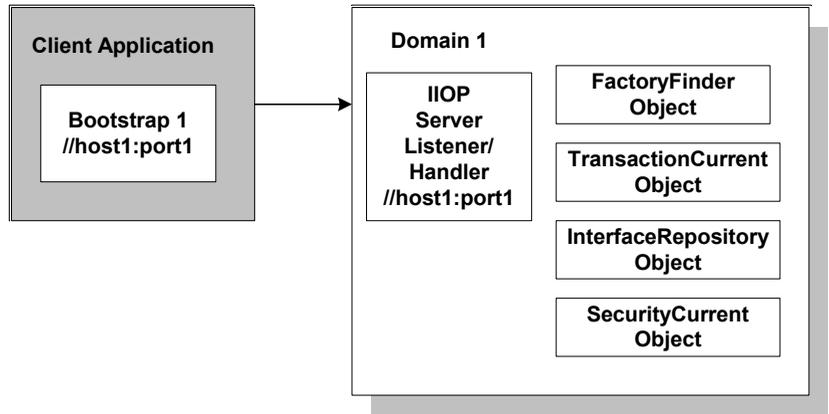
## Domains

A domain is a way of grouping objects and services together as a management entity. A BEA Tuxedo domain has at least one IOP Server Listener/Handler (ISL/ISH) and is identified by a name. One client application can connect to multiple BEA Tuxedo domains using different Bootstrap objects. For each BEA Tuxedo domain, a client application can get a FactoryFinder object, an InterfaceRepository object, a SecurityCurrent object, and a TransactionCurrent object, which correspond to the services offered within the BEA Tuxedo domain. For a description of the Bootstrap object, the FactoryFinder object, the InterfaceRepository object, the SecurityCurrent object, and the TransactionCurrent object, see "Environmental Objects" in this chapter.

**Note:** Only one TransactionCurrent object and one SecurityCurrent object can exist at the same time, and they must be associated with the same Bootstrap object.

Figure 1-2 illustrates how an BEA Tuxedo domain works.

**Figure 1-2 How a BEA Tuxedo Domain Works**



## Environmental Objects

The BEA Tuxedo software provides a set of environmental objects that set up communication between client applications and server applications in a particular BEA Tuxedo domain. The BEA Tuxedo software provides the following environmental objects:

- **Bootstrap**

This object establishes communication between a client application and a BEA Tuxedo domain. It also obtains object references for the other environmental objects in the BEA Tuxedo domain.

- **FactoryFinder**

This CORBA object locates a factory, which in turn can create object references for CORBA objects.

- **SecurityCurrent**

This object can be used to log a client application into a BEA Tuxedo domain with the proper security. The BEA Tuxedo software provides an implementation of the CORBAservices Security Service.

- **TransactionCurrent**

This object allows a client application to participate in a transaction. The BEA Tuxedo software provides an implementation of the CORBAservices Object Transaction Service (OTS).

- **UserTransaction**

This object allows a client application to participate in a transaction. The BEA Tuxedo software provides an implementation of the Sun Microsystems, Inc. Java Transaction Application Programming Interface (JTA API). This object is supported with Java client and server applications only.

- **InterfaceRepository**

This CORBA object contains interface definitions for all the available CORBA interfaces and the factories used to create object references to the CORBA interfaces.

The BEA Tuxedo software provides environmental objects for the Automation programming environment.

## Bootstrap Object

The client application creates a Bootstrap object. A list of ISLs/ISHs can be supplied either as a parameter or via the `TOBJADDR` environmental variable or Java property. A single ISL/ISH is specified as follows:

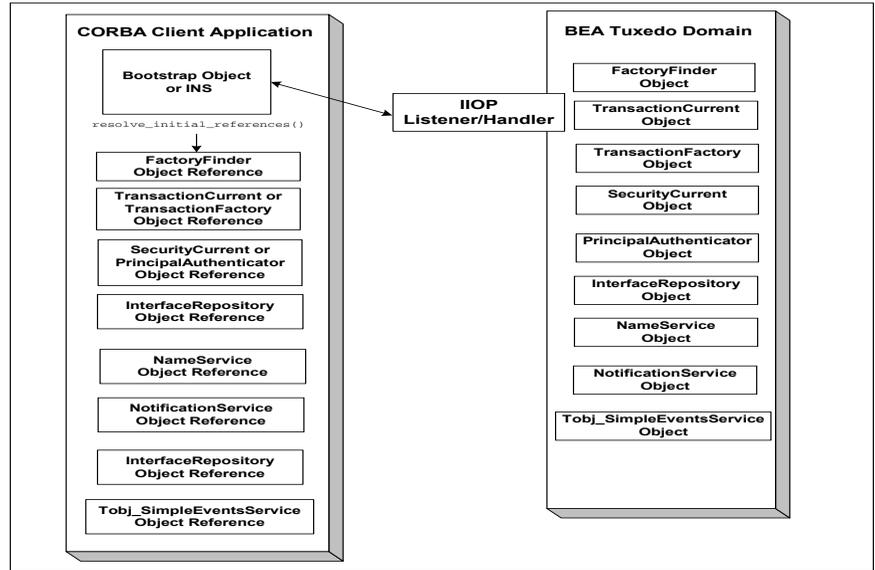
```
//host:port
```

For example, `//myserver:4000`

Once the Bootstrap object is instantiated, the `resolve_initial_references` method is invoked, passing in a string ID, to obtain a reference to an available object. The valid values for the string ID are `FactoryFinder`, `TransactionCurrent`, `SecurityCurrent`, and `InterfaceRepository`.

Figure 1-3 illustrates how the Bootstrap object works in a BEA Tuxedo domain.

**Figure 1-3 How the Bootstrap Object Works**



## Factories and the FactoryFinder Object

Client applications get object references to CORBA objects from a factory. A factory is any CORBA object that returns an object reference to another CORBA object and registers itself with the FactoryFinder object.

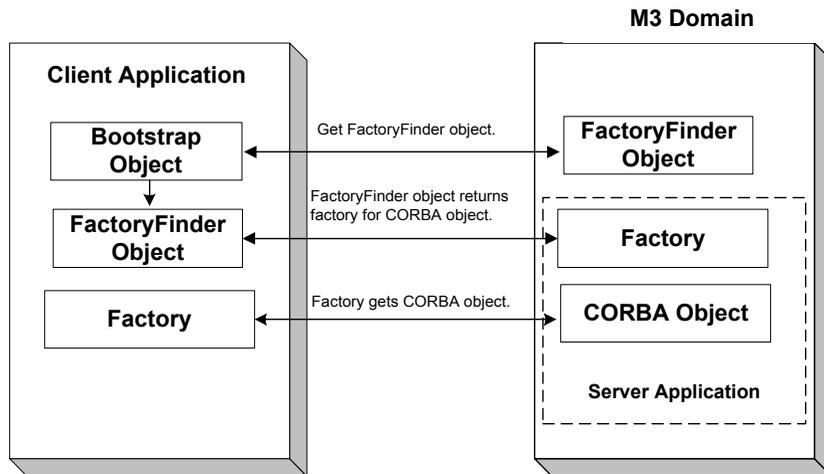
To use a CORBA object, the client application must be able to locate the factory that creates an object reference for the CORBA object. The BEA Tuxedo software offers the FactoryFinder object for this purpose. The factories available to client applications are those that are registered with the FactoryFinder object by BEA Tuxedo server applications at startup.

The client application uses the following sequence of steps to obtain a reference to a CORBA object:

1. Once the Bootstrap object is created, the `resolve_initial_references` method is invoked to obtain the reference to the FactoryFinder object.
2. Client applications query the FactoryFinder object for object references to the desired factory.
3. Client applications call the factory to obtain an object reference to the CORBA object.

Figure 1-4 illustrates the client application interaction with the FactoryFinder object.

**Figure 1-4 How Client Applications Use the FactoryFinder Object**



## Naming Conventions and BEA Tuxedo Extensions to the FactoryFinder Object

The factories available to client applications are those that are registered with the FactoryFinder object by the BEA Tuxedo server applications at startup. Factories are registered using a key consisting of the following fields:

- The Interface RepositoryId of the factory's interface
- An object reference to the factory

The `FactoryFinder` object used by the BEA Tuxedo software is defined in the CORBAServices Life Cycle Service. The BEA Tuxedo software implements extensions to the `COS::LifeCycle::FactoryFinder` interface that make it easier for client applications to locate a factory using the `FactoryFinder` object.

The CORBAServices Life Cycle Service specifies the use of names as defined in the CORBAServices Naming Service to locate factories with the `COS::LifeCycle::FactoryFinder` interface. These names consist of a sequence of `NameComponent` structures, which consist of `ID` and `kind` fields.

The use of CORBA names to locate factories is cumbersome for client applications; it involves many calls to build the appropriate name structures and assemble the Naming Service name that must be passed to the `find_factories` method of the `COS::LifeCycle::FactoryFinder` interface. Also, since the method can return more than one factory, client applications must manage the selection of an appropriate factory and the disposal of unwanted object references.

The `FactoryFinder` object is designed to make it easier for client applications to locate factories by extending the interface with simpler method calls.

The extensions are intended to provide the following simplifications for the client application:

- Let the client application locate factories by ID, using a simple string parameter for the ID field. This reduces the work needed by the client application to build name structures.
- Permit the `FactoryFinder` object to implement a load balancing scheme by choosing from a pool of available factories.
- Provide methods that return one object reference to a factory, instead of a sequence of object references. This eliminates the need for client applications to provide code to handle the selection of a single factory from a sequence, and then dispose of the unneeded references.

The most straightforward application design can be achieved by using the `Tobj::FactoryFinder::find_one_factory_by_id` method in client applications. This method accepts a simple string for factory ID as input and returns one factory to the client application. The client application is freed from the necessity of manipulating name components and selecting among many factories.

To use the `Tobj::FactoryFinder::find_one_factory_by_id` method, the application designer must establish a naming convention for factories that client applications can use to easily locate factories for specific CORBA object interfaces.

Ideally, this convention should establish some mnemonic types for factories that supply object references for certain types of CORBA object interfaces. Factories are then registered using these conventions. For example, a factory that returns an object reference for Student objects might be called StudentFactory.

It is recommended that you either use the actual interface ID of the factory in the OMG IDL file, or specify the factory ID as a constant in the OMG IDL file. This technique ensures naming consistency between the client application and the server application.

## SecurityCurrent Object

The SecurityCurrent object is a BEA Tuxedo implementation of the CORBAServices Security Service. The BEA Tuxedo security model is based on authentication. You use the SecurityCurrent object to specify the appropriate level of security. The following levels of authentication are provided:

- `TOBJ_NOAUTH`

No authentication is needed; however, the client application can still authenticate itself, and must specify a username and a client name, but no password is required.

- `TOBJ_SYSAUTH`

The client application must authenticate itself to the BEA Tuxedo domain, and must specify a username, client name, and client application password.

- `TOBJ_APPAUTH`

The client application must provide information in addition to that which is required by `TOBJ_SYSAUTH`. If the default BEA Tuxedo CORBA authentication service is used in the BEA Tuxedo domain configuration, the client application must provide a user password; otherwise, the client application provides authentication data that is interpreted by the custom authentication service in the BEA Tuxedo domain.

**Note:** If a client application is not authenticated and the security level is `TOBJ_NOAUTH`, the ISL/ISH of the BEA Tuxedo domain registers the client application with the username and client application name sent to the ISL/ISH.

In the BEA Tuxedo CORBA, only the PrincipalAuthenticator and Credentials properties on the SecurityCurrent object are supported. For information about using the SecurityCurrent object in client applications, see Chapter 5, “Using Security.”

## TransactionCurrent Object

The TransactionCurrent object is a BEA Tuxedo implementation of the CORBA services Object Transaction Service. The TransactionCurrent object maintains a transactional context for the current session between the client application and the server application. Using the TransactionCurrent object, the client application can perform transactional operations, such as initiating and terminating a transaction and getting the status of a transaction.

Transactions are used on a per-interface basis. During design, the application designer decides which interfaces within a BEA Tuxedo application will handle transactions. A transaction policy for each interface is then defined in an Implementation Configuration File (ICF). The transaction policies are:

- Never

This interface is not transactional. Objects created for this interface can never be involved in a transaction. The BEA Tuxedo software generates an exception (`INVALID_TRANSACTION`) if an implementation with this policy is involved in a transaction. An `AUTOTRAN` policy specified in the `UBBCONFIG` file for the interface is ignored.

- Optional (The is the default `transaction_policy`.)

This interface may be transactional. Objects can be involved in a transaction if the request is transactional. If the `AUTOTRAN` parameter is specified in the `UBBCONFIG` file for the interface, `AUTOTRAN` is on.

- Always

This interface must always be part of a transaction. If the interface is not part of a transaction, a transaction will be automatically started by the TP framework. The transaction is committed when the method ends. (This is the same behavior that results from specifying `AUTOTRAN` for an object with the `optional` transaction policy, except that no administrative configuration is necessary to achieve this behavior, and it cannot be overridden by administrative configuration.)

- Ignore

This interface is not transactional. This interface can be included in a transaction, however, the `AUTOTRAN` policy specified for this implementation in the `UBBCONFIG` file is ignored.

For information about using the `TransactionCurrent` object in client applications, see Chapter 6, “Using Transactions.”

## **InterfaceRepository Object**

The `InterfaceRepository` object returns information about the **Interface Repository** in a specific BEA Tuxedo domain. The `InterfaceRepository` object is based on the CORBA definition of an Interface Repository. It offers the proper set of CORBA interfaces as defined by the *Common Request Broker Architecture and Specification, Version 2.2*.

ActiveX client applications are not aware they are using the `InterfaceRepository` object. ActiveX client applications use the `Bootstrap` object to obtain a reference to the `InterfaceRepository` object.



# 2 Creating ActiveX Client Applications

This Help topic includes the following sections:

- Summary of the Development Process for ActiveX Client Applications
- The BEA Application Builder
- Step 1: Loading the Automation Environmental Objects into the Interface Repository
- Step 2: Loading the CORBA Interfaces into the Interface Repository
- Step 3: Starting the Interface Repository Server Application
- Step 4: Creating ActiveX Bindings for the CORBA Interfaces
- Step 5: Loading the Type Library for the ActiveX Bindings
- Step 6: Writing the ActiveX Client Application
- Step 7: Deploying the ActiveX Client Application

For a description of the concepts you need to understand before developing an ActiveX client application, see Chapter 1, “Overview.”

# Summary of the Development Process for ActiveX Client Applications

The steps for creating an **ActiveX client application** are as follows:

Step	Description
1	Load the Automation <b>environmental objects</b> into the Interface Repository.
2	Verify that the CORBA interfaces you want to access from your ActiveX client application are loaded in the Interface Repository. If necessary, load the Object Management Group (OMG) Interface Definition Language (IDL) definitions for the CORBA interfaces into the Interface Repository.
3	Start the server application process for the Interface Repository.
4	Use the BEA Application Builder to create ActiveX bindings for the interfaces of the CORBA object.
5	Load the type library for the ActiveX binding in your development tool.
6	Write the ActiveX client application. This chapter describes creating a basic client application. You can also implement security and transactions in your ActiveX client applications. <ul style="list-style-type: none"><li>■ For information about implementing security in your ActiveX client application, see Chapter 5, “Using Security.”</li><li>■ For information about using transactions in your ActiveX client application, see Chapter 6, “Using Transactions.”</li></ul>
7	Create a deployment package for the ActiveX client application.

Each step in the process is explained in detail in the following sections.

The BEA Tuxedo development environment for ActiveX client applications includes the following:

- The `idl2ir` command, which loads interface definitions defined in OMG IDL into the Interface Repository.
- The Application Builder, which creates ActiveX bindings for the interfaces of CORBA objects and creates deployment packages for the interfaces.
- The Automation environmental objects, which provide access to ActiveX views of CORBA objects (referred to as ActiveX views) in a BEA Tuxedo domain and the services provided by the ActiveX views.

## The BEA Application Builder

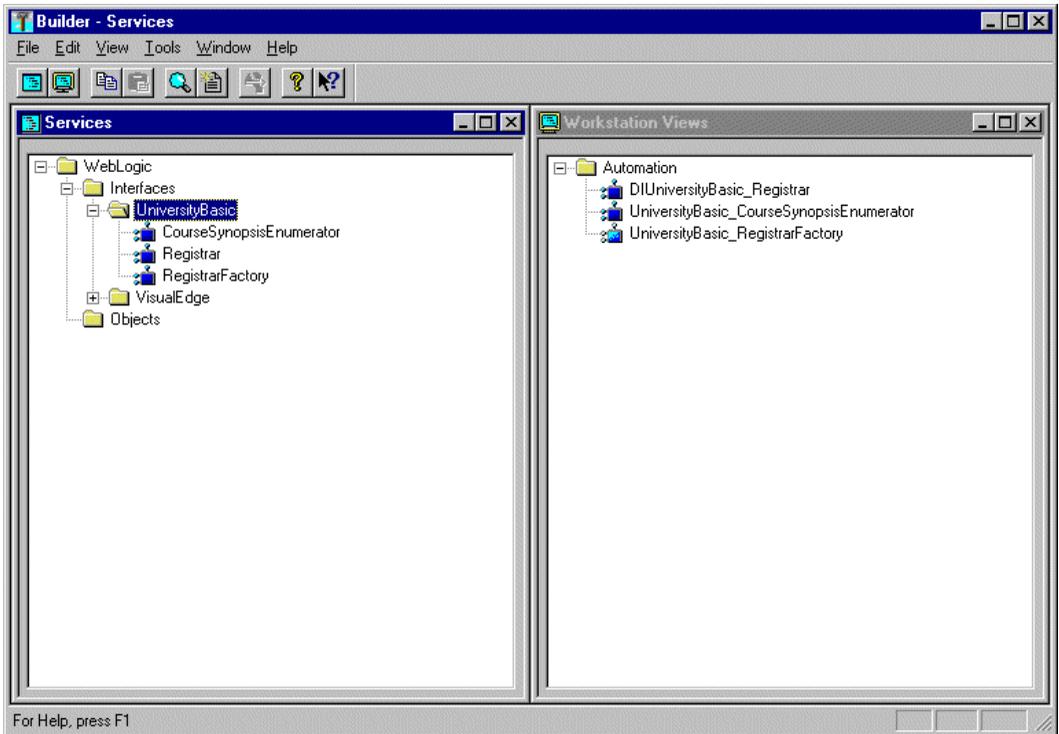
The Application Builder is the development tool that creates ActiveX views of CORBA objects. The Application Builder is the primary user interface to the BEA ActiveX Client. It can be used to select which CORBA objects are available to desktop applications, to create ActiveX views of the CORBA objects, and to create packages for deploying ActiveX views of CORBA objects to client machines.

To use an ActiveX view, you load the interfaces of the CORBA objects into the Interface Repository. You then create an ActiveX binding for the CORBA interface. The binding describes the interface of a CORBA object to ActiveX. The combination of the ActiveX client application and the generated binding creates the view of the object.

For information on how to invoke Application Builder, see Starting Application Builder in Chapter 4, “Tasks.”

As shown in Figure 2-1, the Application Builder main window is partitioned into two parts: the Services window and the Workstation Views window.

Figure 2-1 Application Builder Main Window



The Services window presents all the CORBA modules, interfaces, and operations contained in the Interface Repository in the local BEA Tuxedo domain (referred to as the M3 domain in the BEA Application Builder software that is installed as part of the BEA Tuxedo software kit). You can create bindings for all the interfaces in the Interface Repository.

At the top of the Services window are entries for each object system that is available from the BEA Tuxedo domain. The ActiveX Client supports only the BEA Tuxedo object system. The objects are displayed in the same hierarchical format used in the Interface Repository, that is, as modules, interfaces, operations, and the parameters contained in operations. The [+] symbol indicates an object that can be expanded to display the other objects.

The Workstation Views window presents all the ActiveX bindings that have been created for CORBA interfaces. To create a binding for a CORBA interface, you drag an entry from the Services window and into the Workstation Views window.

## Step 1: Loading the Automation Environmental Objects into the Interface Repository

---

For a better description of the Application Builder Main Window, see Chapter 3, “Application Builder Main Window.”

The steps below refer to the University sample applications shipped with BEA Tuxedo. For more information on the sample applications, see the [Guide to the CORBA University Sample Applications](#) in the BEA Tuxedo online documentation.

# Step 1: Loading the Automation Environmental Objects into the Interface Repository

Load the Automation environmental objects into the Interface Repository so that the interface definitions for the objects are available to ActiveX client applications. From the MS-DOS prompt, enter the following command to load the OMG IDL file (`TOBJIN.idl`) into the Interface Repository:

```
prompt> idl2ir -D _TOBJ -I drive:\tuxdir\include drive:\tuxdir\include\tobjin.idl
```

# Step 2: Loading the CORBA Interfaces into the Interface Repository

Before you can create an ActiveX view for a CORBA object, the interfaces of the CORBA object need to be loaded into the Interface Repository. If the interfaces of a CORBA object are not loaded in the Interface Repository, they do not appear in the Services window of the Application Builder. If a desired CORBA interface does not appear in the Services window, use the `idl2ir` command to load the OMG IDL that defines the CORBA into the Interface Repository. The syntax for the `idl2ir` command is as follows:

```
idl2ir [repositoryfile.idl] file.idl
```

Option	Description
<i>repositoryfile</i>	Directs the command to load the OMG IDL files for the CORBA interface into the specified Interface Repository. Specify the name of the Interface Repository in the BEA Tuxedo domain that the ActiveX client application will access.
<i>file.idl</i>	Specifies the OMG IDL file containing definitions for the CORBA interface.

For a complete description of the `idl2ir` command, see the [BEA Tuxedo Command Reference](#) in the BEA Tuxedo online documentation.

For example, if the University sample application OMG IDL file has been loaded into the interface repository, the following CORBA interfaces should appear in the Application Builder window:

- RegistrarFactory
- Registrar
- CourseSynopsisEnumerator

## Step 3: Starting the Interface Repository Server Application

ActiveX client applications read the interface definitions for CORBA objects from the Interface Repository dynamically at run time and translate them to Automation objects. Therefore, the server application for the Interface Repository needs to be started so that the interface definitions are available. Use the `UBBCONFIG` file to start the server application process for the Interface Repository.

**Note:** In some cases, the system administrator may have performed this step.

In the `UBBCONFIG` file for the BEA Tuxedo domain, check that `TMIFRSVR`, the server application for the Interface Repository, is started. The following entry should appear in the `UBBCONFIG` file:

```
TMIFRSVR
  SRVGRP = SYS_GRP
  SRVID = 6
  RESTART = Y
  MAXGEN = 5
  GRACE = 3600
```

In addition, make sure that the `ISL` parameter to start the ISL/ISH is specified. The following entry should appear in the `UBBCONFIG` file:

```
ISL
  SRVGRP = SYS_GRP
  SRVID = 5
  CLOPT = "-A -- -n //TRIXIE:2500"
```

where `TRIXIE` is the name of the host (server) system and `2500` is the port number.

For more information about starting server applications and specifying the `ISL` parameter, see [Setting Up a BEA Tuxedo Application](#) in the BEA Tuxedo online documentation.

# Step 4: Creating ActiveX Bindings for the CORBA Interfaces

For an ActiveX client application to access a CORBA object, you must generate ActiveX bindings for the interfaces of the CORBA object. You use the Application Builder to create the ActiveX bindings for CORBA interfaces.

To create an ActiveX binding for a CORBA interface:

1. Click the BEA Application Builder icon in the BEA Tuxedo (C++) program group.  
The Domain logon window appears.
2. Enter the host name and port number that you specified in the `ISL` parameter in the `UBBCONFIG` file in the logon window. You must match exactly the capitalization used in the `UBBCONFIG` file.

The Application Builder logon window appears.

3. Highlight the desired CORBA interface in the Services window and drag it to the Workstation Views window, or cut the CORBA interface from the Services window and paste it into the Workstation Views window.

The Application Builder:

- Creates a type library. By default, the type library is placed in `\tuxdir\TypeLibraries`.

The type library file is named: `DImodulename_interfacename.tlb`

- Creates a Windows system registry entry, including unique Program IDs for each object type, for the CORBA interface.

You can now use the ActiveX view from an ActiveX client application.

# Step 5: Loading the Type Library for the ActiveX Bindings

Before you start writing your ActiveX client application, you need to load the type library that describes the ActiveX binding for the CORBA interface in your development tool. Follow your development product's instructions for loading type libraries.

For example, in Visual Basic version 5.0, you use the References option on the Project menu to get a list of available type libraries. You then select the desired type libraries from the list.

By default, the Application Builder places all generated type libraries in `\tuxdir\TypeLibraries`. The type library for the ActiveX binding of the CORBA interface has the following format:

```
DImodulename_interfacename.tlb
```

## **Step 6: Writing the ActiveX Client Application**

The ActiveX client application must do the following:

1. Include declarations for the Automation environmental objects, the factory for the ActiveX view, and the ActiveX view.
2. Establish communication with the BEA Tuxedo domain.
3. Use the Bootstrap object to obtain a reference to the FactoryFinder object.
4. Use a factory to obtain an object reference to an ActiveX view.
5. Invoke operations on the ActiveX view.
6. Creating an Automation Server for Callbacks.
7. Deploy the ActiveX client application.

The following sections use portions of the ActiveX client applications in the Basic University sample application to illustrate the steps.

### **Including Declarations for the Automation Environmental Objects, Factories, and ActiveX Views of CORBA Objects**

To prevent errors at run time, you need to declare the object types of:

- The Automation environmental objects
- The factories that create the ActiveX views of the CORBA objects
- The ActiveX views

The following example is Visual Basic code that declares the Bootstrap and FactoryFinder objects, the factory for the ActiveX view of the Registrar object, and the ActiveX view of the Registrar object:

```
\\Declare Bootstrap object\\
    Public objBootstrap As DITobj_Bootstrap
\\Declare FactoryFinder object\\
    Public objFactoryFinder As DITobj_FactoryFinder
\\Declare factory object for Registrar Object\\
    Public objRegistrarFactory As DIUniversityB_RegistrarFactory
\\Declare the ActiveX view of the Registrar object\\
    Public objRegistrar As DIUniversityB_Registrar
```

## Establishing Communication with the BEA Tuxedo Domain

When writing an ActiveX client application, there are two steps to establishing communication with the BEA Tuxedo domain:

1. Create the Bootstrap object.
2. Initialize the Bootstrap object.

The following Visual Basic example illustrates using the `CreateObject` operation to create a Bootstrap object:

```
Set objBootstrap = CreateObject("Tobj.Bootstrap")
```

You then initialize the Bootstrap object. When you initialize the Bootstrap object, you supply the host and port of the ISL/ISH of the desired BEA Tuxedo domain, as follows:

```
objBootstrap.Initialize "//host:port"
```

The host and port combination for the ISL/ISH is defined in the `ISL` parameter of the `UBBCONFIG` file. The host and port combination that is specified for the Bootstrap object must exactly match the `ISL` parameter. The format of the host and port combination, as well as the capitalization, must match. If the addresses do not match, the call to the Bootstrap object will fail and the following message appears in the log file:

```
Error: Unofficial connection from client at <tcp/ip address>/<portnumber>
```

For example, if the network address is specified as `//TRIXIE::3500` in the ISL parameter in the `UBBCONFIG` file, specifying either `//192.12.4.6.:3500` or `//trixie:3500` in the Bootstrap object will cause the connection attempt to fail.

A BEA Tuxedo domain can have multiple ISL/ISHs. If you are accessing a BEA Tuxedo domain with multiple ISL/ISHs, you supply a list of `host:port` combinations to the Bootstrap object. The Bootstrap object walks through the list until it connects to a BEA Tuxedo domain. The list of ISL/ISHs can also be specified in the `TOBJADDR` environmental variable.

If you want to access multiple BEA Tuxedo domains, you must create a Bootstrap object for each BEA Tuxedo domain you want to access.

## Obtaining References to the FactoryFinder Object

The client application must obtain initial references to the objects that provide services for the application. The Bootstrap object is used to obtain references to the FactoryFinder object, SecurityCurrent object, and TransactionCurrent object. The argument passed to the operation is a string containing the `progid` of the desired object. You have to get references only for the objects that you plan to use in your ActiveX client application.

The following Visual Basic example shows how to use the Bootstrap object to obtain a reference to the FactoryFinder object:

```
Set objFactoryFinder = objBootstrap.CreateObject("Tobj.FactoryFinder")
```

## Using a Factory to Get an ActiveX View

ActiveX client applications get interface pointers to ActiveX views of CORBA objects from factories. A factory is any CORBA object that returns an object reference to another CORBA object. The ActiveX client application invokes an operation on a factory to obtain an object reference to a CORBA object of a specific type. To use factories, the ActiveX client application must be able to locate the factory it needs. The FactoryFinder object serves this purpose.

Use the `CreateObject` function to create the `FactoryFinder` object, and then use one of the `FactoryFinder` object methods to find a factory. The `FactoryFinder` object has the following methods:

- `find_factories()`  
Returns a sequence of factories that match the input key exactly.
- `find_one_factory()`  
Returns one factory that matches the input key exactly.
- `find_factories_by_id()`  
Returns a sequence of factories whose ID field in the name component matches the input argument.
- `find_one_factory_by_id()`  
Returns one factory whose ID field in the factory's CORBA name component matches the input argument.
- `list_factories()`  
Lists factory objects currently registered with the `FactoryFinder`.

The following Visual Basic example shows how to use the `FactoryFinder` `find_one_factory_by_id()` method to get a factory for the `Registrar` object used in the client application for the BEA Tuxedo University sample applications:

```
Set objRegistrarFactory =  
    objBsFactoryFinder.find_one_factory_by_id ("RegistrarFactory")  
Set objRegistrar = RegistrarFactory.find_registrar
```

## Invoking Operations on the ActiveX View

Invoke **operations** on the ActiveX view by passing it a pointer to the **factory** and any arguments that the operation requires.

The following Visual Basic example shows how to invoke operations on an ActiveX view:

```
'Get course details from the Registrar object'  
aryCourseDetails =  
    objRegistrar.get_course_details(aryCourseNumbers)
```

## Creating an Automation Server for Callbacks

In some application development scenarios, it may be desirable to allow the ActiveX client application to respond to requests from the CORBA server application. Rationales for callbacks from the CORBA server might include notifying the client application when a certain event has occurred, validating security, or obtaining additional information from the client. For example, a client application that tracks stock prices might request of a CORBA server that it be notified when a specified stock changes value. The client might do this by passing a notification object reference to the CORBA server, which the server then uses to call back to notify the client when the stock has changed price. The following description of the process for developing an ActiveX client application that can function as a COM server assumes you are developing the ActiveX client in Visual Basic.

To develop an ActiveX application that can act as a COM server in relation to a CORBA application, you follow the six steps described above. In addition, however, you implement the COM server functionality for a CORBA interface in Visual Basic by creating an appropriate Visual Basic class.

One way to do this is to start by selecting the **Add Class** option in the Visual Basic **Project** menu. Add an `Implements` clause to the class naming the Automation view of the CORBA interface, as it appears in the type library that you created using Application Builder. (See Step 4: Creating ActiveX Bindings for the CORBA Interfaces.) For example:

```
Implements ChatClient_Listener
```

This example is taken from the chatroom Visual Basic client sample that is packaged with BEA Tuxedo. The chatroom sample is by default located at:

```
tuxdir\samples\corba\chatroom
```

In this example, `ChatClient_Listener` is the name of the interface. You would then write private Visual Basic subroutines to implement each of the methods included in the interface. For example:

```
Private Sub ChatClient_Listener_post(ByVal from As String,  
ByVal output_line As String, Optional exceptionInfo As Variant)  
MsgBox "User " + from + ": " + output_line  
End Sub
```

### **Creating Instances of the COM Objects**

Now that you have implemented the COM object, you can create instances of it in your ActiveX client application and pass those instances to CORBA services. You create instances of these COM objects in exactly the same way that you create instances of any COM object. For example:

```
Dim aListener as ChatClient_Listener  
Set aListener = New MyListener
```

The call to `New` creates the instance, where `ChatClient_Listener` is the name of the interface and `MyListener` is the name of the class you created to implement it. Once an instance exists, it can be specified as a parameter to a CORBA method. For example:

```
aModerator.signon "Hansel", aListener
```

where `aModerator` is a CORBA object and `aListener` is the COM object that the CORBA object will call back to as necessary.

## **Step 7: Deploying the ActiveX Client Application**

To distribute ActiveX client applications to other client machines, you need to create a deployment package. A deployment package contains all the data needed by the client application to use ActiveX views of CORBA objects, including the bindings, the type libraries, and the registration information. The deployment package is a self-registering ActiveX control with the file extension `.ocx`.

To create a deployment package for an ActiveX view:

1. Select an ActiveX view from the Workstation Views window.
2. Click Tools->Deploy Modules, or click the right mouse button on the desired view and choose the Deploy Modules option from the menu. A confirmation window is displayed.
3. Click Create to create the deployment package.

By default, the deployment package is placed in `\tuxdir\Packages`.

## **2** *Creating ActiveX Client Applications*

---

# 3 Application Builder Main Window

This Help topic includes the following sections:

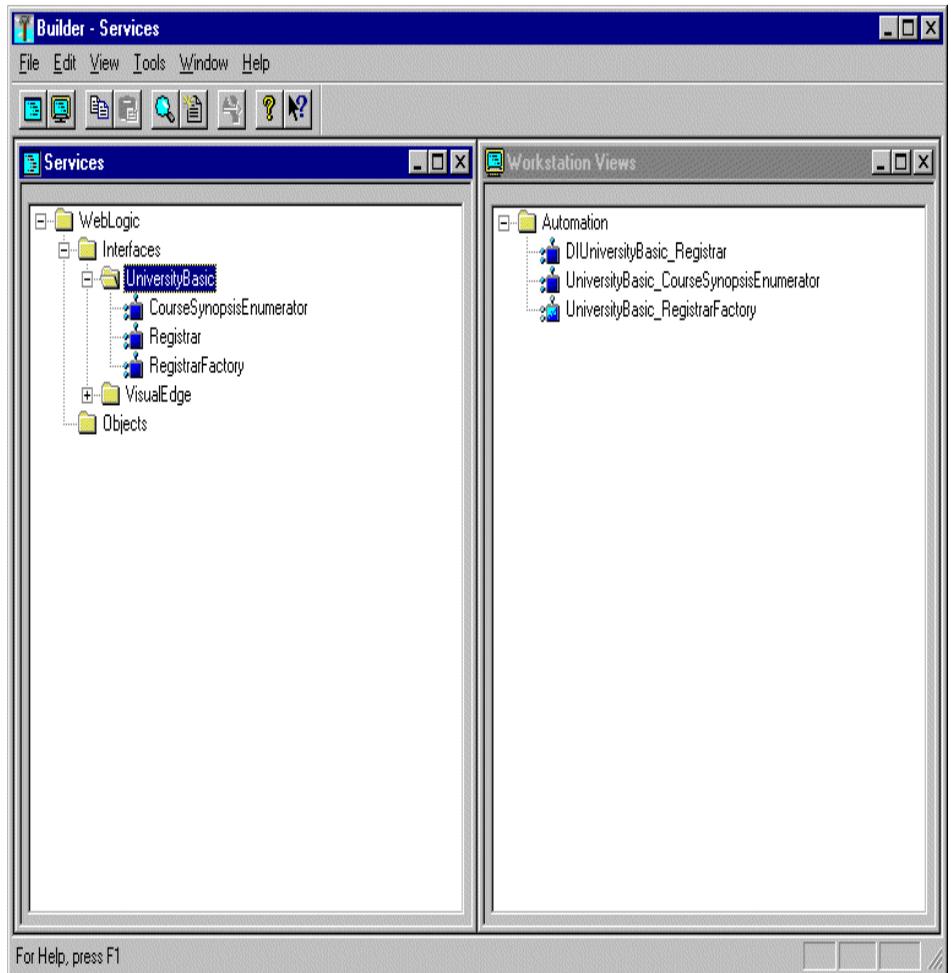
- Application Builder Main Window
- Services Window
- Workstation Views Window
- Application Builder Objects
- Menu Options
- Toolbar Buttons

## Application Builder Main Window

As shown in Figure 3-1, the Application Builder main window is divided into two parts: the Services window and the Workstation Views window.

### 3 Application Builder Main Window

Figure 3-1 Application Builder Main Window



When you start the Application Builder, the main window displays one Services window and one Workstation Views window. You can use the New option on the File menu to create additional Services and Workstation Views windows. You can also use the Window Menu options to change the arrangement of the Services and Workstation Views windows.

## Services Window

The Services window presents all the CORBA modules, interfaces, and operations contained in the Interface Repository in the local BEA Tuxedo domain. You can create bindings for all the items in the Interface Repository.

At the top of the Services window are entries for each object system that is available from the BEA Tuxedo domain. This release of the ActiveX Client supports only the BEA Tuxedo object system. The objects are displayed in the same hierarchical format used in the Interface Repository, that is, as modules, interfaces, methods, and the parameters contained in methods. The [+ ] symbol indicates an object that can be expanded to display the other objects.

The Services window also can be used to display other kinds of Interface Repository definitions such as attributes, methods, and data types. Use the options on the Display tab page on the Options window to select which kinds of Interface Repository definitions are displayed in the Services window. (For information about the Display tab page, see *Filtering Objects Displayed in the Main Window*.)

To open an additional Services window from within the Application Builder, choose File->New->Services Window.

## Workstation Views Window

The Workstation Views window presents all the ActiveX bindings that have been created for CORBA interfaces. To create a binding for a CORBA interface, you drag an entry in the Services window and drop it into the Workstation Views window.

The ActiveX object system does not support a hierarchical module structure; therefore, the tree structure of the ActiveX bindings in the Workstation Views window does not necessarily match the tree structure in the Services window. The Application Builder alters the names of the bindings to ensure uniqueness and to conform with the naming convention of the ActiveX object model.

### 3 *Application Builder Main Window*

---

The Workstation Views window also can be used to display other kinds of Interface Repository definitions such as attributes, methods, and data types. Use the options on the Display tab page on the Options Window to select which kinds of Interface Repository definitions are displayed in the Workstation Views window. (For more information about the Display tab page, see Filtering Objects Displayed in the Main Window.)

To open an additional Workstation Views window from within the Application Builder, choose File->New->Workstation Views Window.

## Application Builder Objects

Table 3-1 explains the objects represented in the Application Builder main window.

**Table 3-1 Explanation of Objects in the Application Builder**

Icons	Description
	Indicates the available object systems such as BEA Tuxedo and ActiveX. For this release of the ActiveX Client, only the BEA Tuxedo system is supported.
	An argument that is passed as a parameter to a method. An argument is based on a single data type (for example, integer, floating point, character) or a structure (for example, float, string, enumerator).
	An input argument.
	An output argument.

**Table 3-1 Explanation of Objects in the Application Builder (Continued)**

Icons	Description
	An input/output argument.
	A data structure (for example, float, string, enumerator).
	An exception.
	An interface which is a set of methods and properties.
	A method which is an operation that can be invoked on an object.
	A module which is a group of one or more interfaces.
	A property which is a data attribute associated with an object.
	Indicates that this ActiveX view is a server application that can be a source of objects.

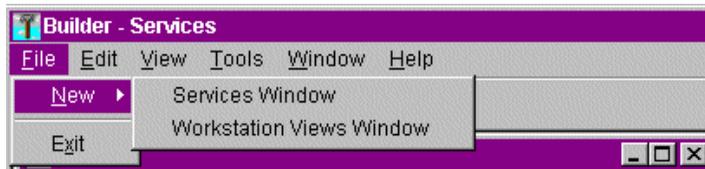
## Menu Options

This Help describes the menu options in the Application Builder.

### File Menu Options

Figure 3-2 shows the File menu options.

**Figure 3-2 Expanded File Menu**



From the File menu, you can use:

- The New->Services Window option to open an additional Services window.
- The New->Workstation Views Window option to open an additional Workstation Views window.
- The Exit option to end the Application Builder session.

### Edit Menu Options

Figure 3-3 shows the Edit menu options.

Figure 3-3 Expanded Edit Menu



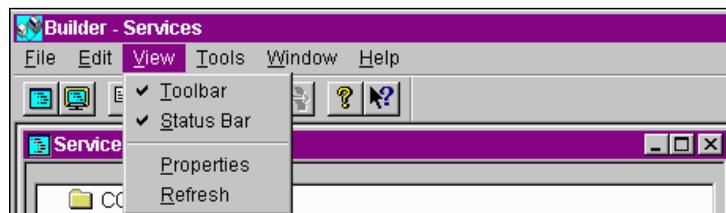
From the Edit menu, you can use:

- The Copy option to copy a CORBA object from the Services window to the clipboard. You can then paste the CORBA object into the Workstation Views window to create a view of the CORBA object. You can also use the CTRL+C keyboard shortcut to perform the copy action. The Copy option is not enabled from the Workstation Views window.
- The Paste option to copy a CORBA object from the Services window to the Workstation Views window. You can also use the CTRL+V keyboard shortcut to perform this paste action. Pasting the CORBA object into the Workstation Views window creates an ActiveX view of the CORBA object.

## View Menu Options

Figure 3-4 shows the View menu options.

Figure 3-4 Expanded View Menu



From the View menu, you can use:

- The Toolbar option to hide or display the toolbar of shortcuts.
- The Status Bar option to hide or display the status window at the bottom of the Application Builder main window.

## 3 Application Builder Main Window

---

- The Properties option to view the characteristics of a CORBA object or an ActiveX view of a CORBA object.
- The Refresh option to update all the windows with new data from the Interface Repository.

## Tools Menu Options

Figure 3-5 shows the Tools menu options.

**Figure 3-5 Expanded Tools Menu**



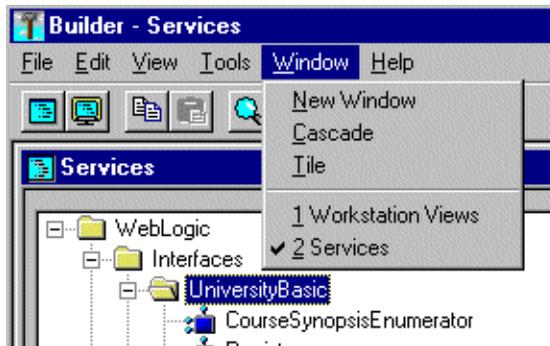
From the Tools menu, you can use the Options option to open the Options window which has the following dialog windows:

- Workstation Bindings—use this window to control the default settings used when creating bindings.
- Deployment Packages—use this window to change the default directory location for the deployment packages.
- Display—use this window to determine the types of objects displayed in the Services window and the Workstation Views window.

## Window Menu Options

Figure 3-6 shows the Window menu options.

Figure 3-6 Expanded Window Menu



From the Window menu, you can use:

- The New option to open either a new Services or Workstation Views window. The Application Builder creates a new window of the same type as the active window.
- The Cascade option to arrange the open Services and Workstation Views windows in an overlapping titled pattern.
- The Tile option to arrange the open Services and Workstation Views windows in a nonoverlapping titled pattern.

The bottom half of the menu lists the open Services and Workstation Views windows. A check mark indicates the active window.

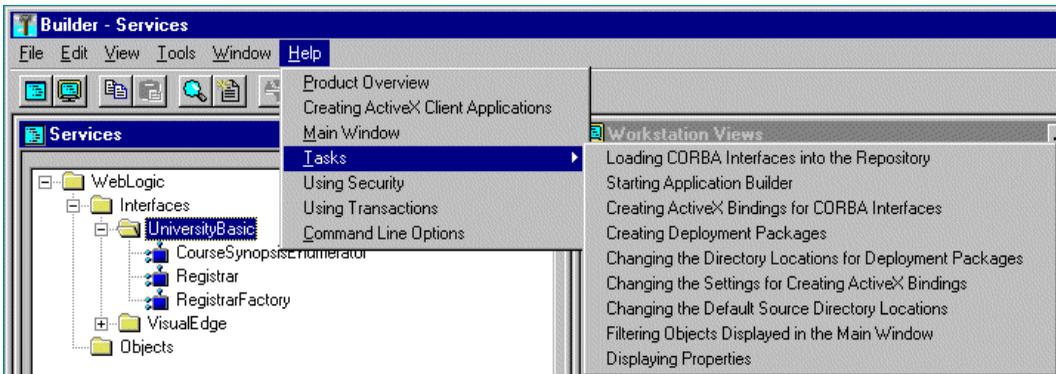
## Help Menu Options

The Help menu options direct you to categories of the Application Builder component descriptions.

Figure 3-7 shows the Help Menu options.

### 3 Application Builder Main Window

Figure 3-7 Expanded Help Menu



From the Help menu, you can bring up descriptions of the Application Builder windows and features.

## Toolbar Buttons

Figure 3-8 shows the Application Builder toolbar.

Figure 3-8 Application Builder Toolbar



The toolbar is located below the menu bar on the main window. The toolbar buttons, from left to right, perform the following functions:

- Opens a new Services window
- Opens a new Workstation Views window
- Copies the selected interface to the clipboard
- Pastes the contents of the clipboard to the designated window
- Displays the properties of the selected interface or view

- Refreshes the active window
- Creates a deployment package for the selected interface
- Provides information about the product, version number, and copyright
- Provides context-sensitive help

### **3** *Application Builder Main Window*

---

# 4 Tasks

This Help topic includes the following sections:

- Loading CORBA Interfaces into the Interface Repository
- Starting Application Builder
- Creating ActiveX Bindings for CORBA Interfaces
- Changing the Settings for Creating ActiveX Bindings for CORBA Interfaces
- Creating Deployment Packages
- Changing the Directory Location for Deployment Packages
- Changing the Settings for Creating ActiveX Bindings for CORBA Interfaces
- Changing the Default Directory Locations
- Filtering Objects Displayed in the Main Window
- Displaying Properties

## Loading CORBA Interfaces into the Interface Repository

Before you can create an ActiveX view of a CORBA object, you need to load the interfaces of the CORBA object into the Interface Repository. If the interfaces of a CORBA object are not loaded in the Interface Repository, they are not displayed in the Services window. If a desired CORBA interface is not displayed in the Services

window, use the `idl2ir` command to load the Object Management Group (OMG) Interface Definition Language (IDL) for the CORBA interface into the Interface Repository. The syntax for the `idl2ir` command is as follows:

```
idl2ir -f repository-name file.idl
```

The following table describes the options for the `idl2ir` command.

Option	Description
<code>-f repository-name</code>	Loads the OMG IDL files for the CORBA interface into the Interface Repository. Specify the Interface Repository that is in the same BEA Tuxedo domain as the ActiveX client application.
<code>file.idl</code>	Specifies the OMG IDL file containing definitions for the CORBA interface.

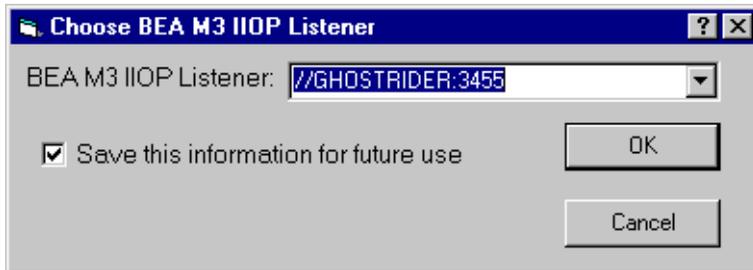
For a complete description of the `idl2ir` command, see the [CORBA Programming Reference](#) in the BEA Tuxedo online documentation.

## Starting Application Builder

To start the Application Builder:

1. Click the BEA Application Builder icon in the BEA BEA Tuxedo System program group.  
A logon window appears.
2. Enter the host name and port number that is specified in the `ISL` parameter in the `UBBCONFIG` file. You must match exactly the capitalization used in the `UBBCONFIG` file. See Figure 4-1.

Figure 4-1 Connecting to the IIOP Listener



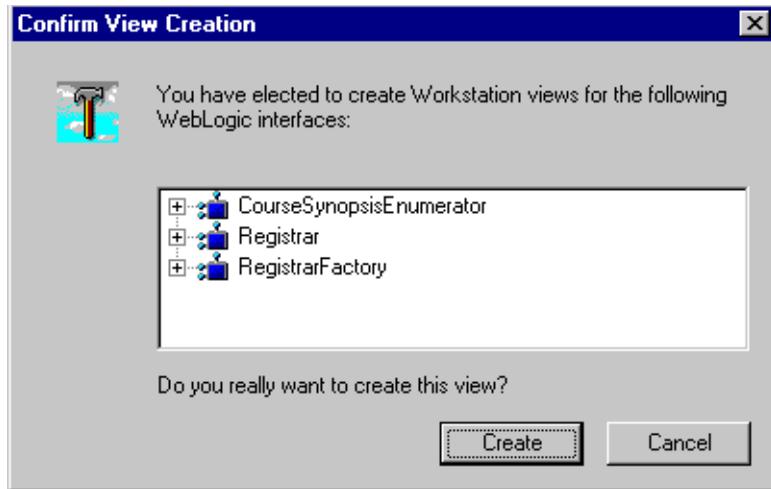
The Application Builder window appears. All the CORBA interfaces loaded in the Interface Repository appear in the Services window of the Application Builder.

## Creating ActiveX Bindings for CORBA Interfaces

To create an ActiveX binding for a CORBA interface:

1. In the Application Builder window, highlight the desired CORBA interface in the Services window.
2. Drag the desired CORBA interface to the Workstation Views window, or cut the CORBA interface from the Services window and paste it into the Workstation Views window.

The Confirm View Creation window appears.



3. To create an ActiveX binding for the CORBA interface, click Create.

The Application Builder creates the following:

- A type library. By default, the type library is placed in `\tuxdir\TypeLibraries`.  
The type library file is named: `DImodulename_interfacename.tlb`.
- A Windows system registry entry, including unique Program IDs for each object type, for the CORBA interface.

You can now use the ActiveX view of a CORBA object from an ActiveX client application.

## Changing the Settings for Creating ActiveX Bindings for CORBA Interfaces

Use the Workstation Bindings tab page on the Options window to change the settings used to create ActiveX bindings for the interfaces of CORBA objects. To get to the Workstation Bindings tab page, click Tools->Options.

Table 4-1 describes the options on the Workstation Bindings tab page.

**Table 4-1 Workstation Bindings Tab Page Options**

<b>Option</b>	<b>Description</b>
Workstation Bindings Options	Lists the types of bindings that can be created for the interfaces of CORBA objects. A check mark appears next to the type of bindings to be created.
Generate COM Views on Workstation Drop	Creates COM bindings for the interfaces of CORBA objects. This release of the ActiveX Client does not support COM views of CORBA objects in a BEA Tuxedo domain.
Generate OLE Automation Views on Workstation Drop	Creates ActiveX bindings for the interfaces of CORBA objects.
Create ActiveX Controls for OLE Automation Views	Adds the necessary interfaces to a CORBA object so that the CORBA object can be used as an ActiveX control. It also registers the CORBA object as an ActiveX control. The CORBA object can then be used in ActiveX Control container applications.
Output Folders	Specifies a directory location for the bindings that are created for the interfaces of a CORBA object.
C++ Headers	C++ header files need to be located in your computer's defined path so that they are compiled properly. By default, the files are placed in: <code>\tuxdir\Include</code> You can click the Browse button to search for a directory location.
MIDL/ODL Files	Microsoft Definition Language (MIDL) and Object Definition Language (ODL) files are for reference only and can be placed anywhere on your computer. By default, the files are placed in: <code>\tuxdir\TypeLibraries</code> You can click the Browse button to search for a directory location.

**Table 4-1 Workstation Bindings Tab Page Options (Continued)**

Option	Description
Type Libraries	Type libraries are registered with a complete directory path and can be placed in any directory that is always available to a client computer. By default, the files are placed in: <code>\tuxdir\TypeLibraries</code> You can click the Browse button to search for a directory location.

# Creating Deployment Packages

To distribute client applications to other client computers, you need to create a deployment package. A deployment package contains all the data the client application needs to have to use ActiveX views of CORBA objects, including the bindings, type libraries, and registration information. The deployment package is a self-registering ActiveX control with the file extension `.ocx`.

To create a deployment package for an ActiveX view of a CORBA object:

1. Select an ActiveX view from the Workstation Views window.
2. Click Tools->Deploy Modules or click the right mouse button on the desired view and select the Deploy Modules option from the menu.

The Confirm Deployment window is displayed.



3. Click Create to create the deployment package.

By default, the deployment package is placed in `\tuxdir\Packages`.

## Changing the Directory Location for Deployment Packages

Use the Deployment Packages tab page on the Options window to change the directory location for deployment packages for ActiveX views of CORBA objects. To access the Deployment Packages tab page, click Tools->Options. The current directory location for the deployment packages is displayed. The default location is `\tuxdir\Packages`.

## Changing the Default Directory Locations

The Application Builder provides default directory locations for C++ header files, MIDL and ODL files, and type libraries. You can change those directory locations.

To change the directory locations:

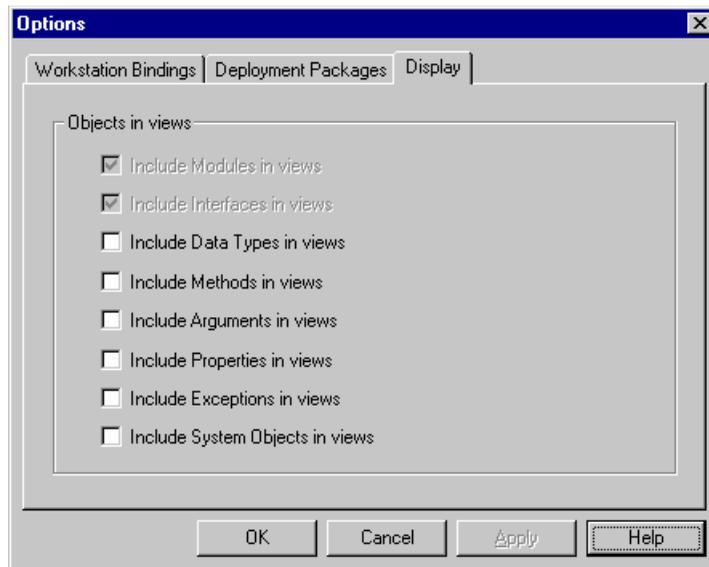
1. From the Tools menu, select the Options option.

The Options window is displayed.

2. Choose the Workstation Bindings tab on the Options window.  
The default directory location is displayed in the C++ Headers, MIDL/ODL Files, and Type Libraries fields.
3. Select the specification for the desired output directory and delete it.
4. Either enter a new directory specification or click the Browse button to search for a new directory.
5. Click OK to save the change.

# Filtering Objects Displayed in the Main Window

Use the Display tab page on the Options window to filter the types of objects displayed in the Application Builder main window. By default, CORBA interfaces and modules are displayed.



You have the option of also displaying the following types of information:

- Data types
- Methods
- Arguments
- Properties
- Exceptions

Use the Include System Objects option to enable the display of a specific set of definitions in the Interface Repository, for example, CosTransactions.

To display additional information in the Application Builder main window, click the desired options and click OK.

## Displaying Properties

Use the Properties window to display one or more pages listing the properties of the selected adapter, module, or interface. The content of the Properties window is object specific.

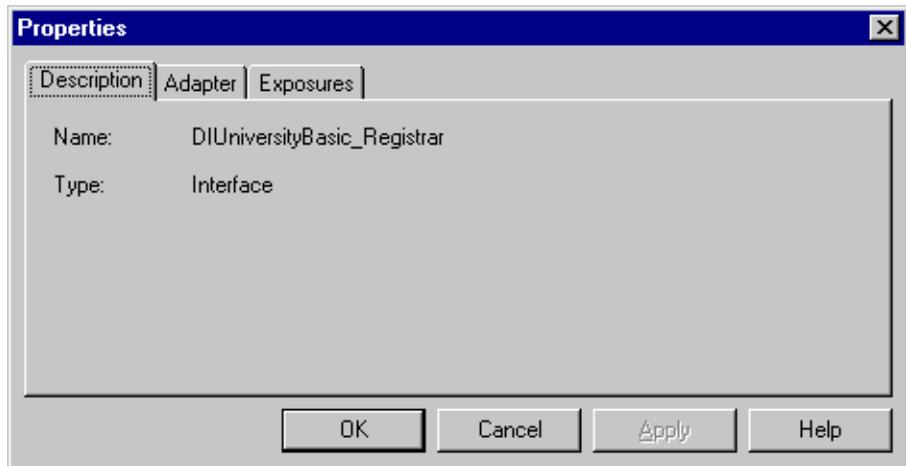


Table 4-2 describes the possible properties.

**Table 4-2 Description of Properties**

<b>Property</b>	<b>Description</b>
Interface->Name	The name of the selected CORBA interface.
Interface->Type	The type of object. For example, interface, module, or exception.
Adapter->Name	The name of the object system. For this release, this option appears as BEA Tuxedo version 8.0.
Adapter->Vendor	The name of the vendor of the object system. For this release, this option appears as BEA Systems.
Adapter->Platform	The version of the object system. This option appears as version 8.0.
Exposure	Describes the source object system of the object. For example, BEA Tuxedo.

# 5 Using Security

This Help topic describes how to use security in ActiveX client applications for the BEA Tuxedo software.

For an overview of the SecurityCurrent object, see Chapter 1, “Overview.”

## Overview of BEA Tuxedo Security

ActiveX client applications use security to authenticate themselves to the BEA Tuxedo domain. Authentication is the process of verifying the identity of a client application. By entering the correct logon information, the client application authenticates itself to the BEA Tuxedo domain. The BEA Tuxedo software uses authentication as defined in the CORBAServices Security Service and provides extensions for ease of use.

A client application must provide security information according to the security level defined in the desired BEA Tuxedo domain. This information is defined by the BEA Tuxedo system administrator in the `UBBCONFIG` file for the BEA Tuxedo domain. When creating client applications, you must work with the BEA Tuxedo system administrator to obtain the correct security information (such as the username and user password) for the BEA Tuxedo domain you want to access from the client application.

# Summary of the Development Process for Security

The steps for adding security to a client application are as follows:

Step	Description
1	Use the Bootstrap object to obtain a reference to the SecurityCurrent object in the specified BEA Tuxedo domain.
2	Get the PrincipalAuthenticator object from the SecurityCurrent object.
3	Use the <code>get_auth_type</code> operation of the PrincipalAuthenticator object to return the type of authentication expected by the BEA Tuxedo domain.
4	Log on to the BEA Tuxedo domain using the required security information.
5	Log off the BEA Tuxedo domain.

The following sections describe these steps and use portions of the client applications in the Security University sample application to illustrate the steps.

## Step 1: Using the Bootstrap Object to Obtain the SecurityCurrent Object

Use the Bootstrap object to obtain an object reference to the SecurityCurrent object for the specified BEA Tuxedo domain. The SecurityCurrent object is a `SecurityLevel12::Current` object as defined by the CORBA Services Security Service.

The following Visual Basic example illustrates how the Bootstrap object is used to return the SecurityCurrent object:

```
Set objSecurityCurrent =  
    objBootstrap.CreateObject("Tobj.SecurityCurrent")
```

## **Step 2: Getting the PrincipalAuthenticator Object from the SecurityCurrent Object**

The SecurityCurrent object returns a reference to the PrincipalAuthenticator for the BEA Tuxedo domain. The PrincipalAuthenticator is used to get the authentication level required for a BEA Tuxedo domain.

The following Visual Basic example illustrates how to obtain the PrincipalAuthenticator for a BEA Tuxedo domain:

```
Set objPrincAuth = objSecurityCurrent.principal_authenticator
```

## **Step 3: Obtaining the Authentication Level**

Use the `Tobj::PrincipalAuthenticator::get_auth_type()` method to get the level of authentication required by the BEA Tuxedo domain.

The following Visual Basic example illustrates how to obtain the PrincipalAuthenticator for a BEA Tuxedo domain:

```
AuthorityType = objPrinAuth.get_auth_type
```

## Step 4: Logging On to the BEA Tuxedo Domain with Proper Authentication

Use the `Tobj::PrincipalAuthenticator::logon()` method to log your client application into the desired BEA Tuxedo domain. The method requires the following arguments:

- *user\_name*

The BEA Tuxedo username. This information is required for `TOBJ_SYSAUTH` and `TOBJ_APPAUTH` authentication levels. This information may be supplied for the `TOBJ_NOAUTH` authentication level; however, it is not required. The system designer decides this name at design time.
- *client\_name*

The BEA Tuxedo client application name. This information is required for `TOBJ_SYSAUTH` and `TOBJ_APPAUTH` authentication levels. This information may be supplied for the `TOBJ_NOAUTH` authentication level; however, it is not required. Obtain this information from the system administrator.
- *system\_password*

The BEA Tuxedo password. This information is required for `TOBJ_SYSAUTH` and `TOBJ_APPAUTH` authentication levels. Obtain this information from the system administrator.
- *user\_password*

The user password for the BEA Tuxedo authentication service. This information is required for the `TOBJ_APPAUTH` authentication level.
- *user\_data*

Application-specific data for authentication. This information is required when the BEA Tuxedo domain the client application is accessing is not using the authentication service provided with the BEA Tuxedo software.

The *user\_password* and *user\_data* arguments are mutually exclusive, depending on the authentication service used in the configuration of the BEA Tuxedo software. If you are using an authentication service other than an authentication service provided by the BEA Tuxedo software, provide the information required for `logon` in the

*user\_data* argument. The `Tobj::PrincipalAuthenticator::logon()` method raises a `CORBA::BAD_PARAM` exception if both *user\_password* and *user\_data* are set.

If a BEA Tuxedo domain has a `TOBJ_NOAUTH` authentication level, the client application is not required to supply a *user\_name* or *client\_name* when logging on to the BEA Tuxedo domain. If the client application does not log on with a *user\_name* and *client\_name*, the IOP Server Listener/Handler (ISL/ISH) of the BEA Tuxedo domain registers the client application with the *user\_name* and the *client\_name* set for the ISL/ISH in the `UBBCONFIG` file. However, the client application can log on with any *user\_name* and *client\_name*.

The `logon()` method returns one of the following:

- `Security::AuthenticationStatus::SecAuthSuccess` if the authentication succeeded
- `Security::AuthenticationStatus::SecAuthFailure` if the authentication failed or if the client application was already authenticated and did not log off the BEA Tuxedo domain

The following Visual Basic example illustrates how to use the `Tobj::PrincipalAuthenticator::logon()` method:

```
If AuthorityType = TOBJ_APPAUTH Then logonStatus =  
    oPrincAuth.Logon(  
        UserName, ClientName, SystemPassword, _  
        UserPassword, UserData)  
End If
```

## Step 5: Logging Off the BEA Tuxedo Domain

The client application must log off the current BEA Tuxedo domain before it can log on as another user in the same BEA Tuxedo domain. Use the `Tobj::PrincipalAuthenticator::logoff()` method to discard the BEA Tuxedo current authentication context and credentials. This method does not close the network connections to the BEA Tuxedo domain. After logging off the BEA Tuxedo domain, calls using the existing authentication fail if the authentication type is not `TP_NOAUTH`.



# 6 Using Transactions

This Help topic describes how to use transactions in ActiveX client applications for the BEA Tuxedo CORBA.

For an overview of the TransactionCurrent object, see Chapter 1, “Overview.”

## Overview of Transactions

Client applications use transaction processing to ensure that data remains correct, consistent, and persistent. The transactions in BEA Tuxedo CORBA allow client applications to begin and terminate transactions and to get the status of transactions. The BEA Tuxedo software uses transactions as defined in the CORBAservices Object Transaction Service, with extensions for ease of use.

Transactions are defined on interfaces. The application designer decides which interfaces within a BEA Tuxedo client/server application will handle transactions. Transaction policies are defined in the Implementation Configuration File (ICF) for C++ server applications. Generally, the ICF file or the Server Description file for the available interfaces is provided to the client programmer by the application designer.

## Summary of the Development Process for Transactions

The steps for adding transactions to a client application are as follows:

Step	Description
1	Use the Bootstrap object to obtain a reference to the TransactionCurrent object in the specified BEA Tuxedo domain.
2	Use the methods of the TransactionCurrent object to include the interface of a CORBA object in a transaction operation.

The following sections describe these steps and use portions of the client applications in the Transactions University sample application to illustrate the steps. For information about the Transactions University sample application, see the [Guide to the CORBA University Sample Applications](#) in the BEA Tuxedo online documentation. The Transactions University sample application is located in the following directory on the BEA Tuxedo software kit:

```
drive:\tuxdir\samples\corba\university\transactions
```

# Step 1: Using the Bootstrap Object to Obtain the TransactionCurrent Object

Use the Bootstrap object to obtain an object reference to the TransactionCurrent object for the specified BEA Tuxedo domain. For a complete description of the TransactionCurrent object, see the [CORBA Programming Reference](#) in the BEA Tuxedo online documentation.

The following Visual Basic example illustrates how the Bootstrap object is used to return the TransactionCurrent object:

```
Set objTransactionCurrent =  
    objBootstrap.CreateObject("Tobj.TransactionCurrent")
```

## Step 2: Using the TransactionCurrent Methods

The TransactionCurrent object has methods that allow a client application to manage transactions. These methods can be used to begin and end transactions and to obtain information about the current transaction. The TransactionCurrent object provides the following methods:

- `begin()`

Creates a new transaction. Future operations take place within the scope of this transaction. When a client application begins a transaction, the default transaction timeout is 300 seconds. You can change this default, using the `set_timeout` method.
- `commit()`

Ends the transaction successfully. Indicates that all operations on this client application have completed successfully.
- `rollback()`

Forces the transaction to roll back.
- `rollback_only()`

Marks the transaction so that the only possible action is to roll back. Generally, this method is used only in server applications.
- `suspend()`

Suspends participation in the current transaction. This method returns an object that identifies the transaction and allows the client application to resume the transaction later.
- `resume()`

Resumes participation in the specified transaction.
- `get_status()`

Returns the status of a transaction with a client application.
- `get_transaction_name()`

Returns a printable string describing the transaction.

- `set_timeout()`

Modifies the timeout period associated with transactions. The default transaction timeout value is 300 seconds. If a transaction is automatically started instead of explicitly started with the `begin()` method, the timeout value is determined by the value of the `TRANTIME` parameter in the `UBBCONFIG` file. For more information about setting the `TRANTIME` parameter, see [Administering a BEA Tuxedo Application at Run Time](#) in the BEA Tuxedo online documentation.

- `get_control()`

Returns a control object that represents the transaction.

A basic transaction works in the following way:

1. A client application begins a transaction using the `Tobj::TransactionCurrent::begin()` method. This method does not return a value.
2. The operations on the CORBA interface execute within the scope of a transaction. If a call to any of these operations raises an exception (either explicitly or as a result of a communications failure), the exception can be caught and the transaction can be rolled back.
3. Use the `Tobj::TransactionCurrent::commit()` method to commit the current transaction. This method ends the transaction and starts the processing of the operation. The transaction is committed only if all of the participants in the transaction agree to commit.

The association between the transaction and the client application ends when the client application calls the `Tobj::TransactionCurrent::commit()` method or the `Tobj::TransactionCurrent::rollback()` method. The following Visual Basic example illustrates using a transaction to encapsulate the operation of a student registering for a class:

```
' Begin the transaction
'
objTransactionCurrent.begin
'
' Try to register for courses
'
NotRegisteredList = objRegistrar.register_for_courses(mStudentID,
    CourseList, exception)
'
```

## *Step 2: Using the TransactionCurrent Methods*

---

```
If exception.EX_majorCode = NO_EXCEPTION then
    ' Request succeeded, commit the transaction
    ,
    Dim report_heuristics As Boolean
    report_heuristics = True
    objTransactionCurrent.commit report_heuristics
Else
    ' Request failed, Roll back the transaction
    ,
    objTransactionCurrent.rollback
    MsgBox "Transaction Rolled Back"
End If
```



# 7 Command-Line Options

This Help topic describes the command-line version of the Application Builder.

The `BEAAppBuilder` command is a command-line version of the Application Builder. The command is used in a makefile, in batch command files, or interactively from the command line. Before using this command, make sure the `ISL` parameter in the `UBBCONFIG` is set to the host and port of your server computer.

## Format

```
BEAAppBuilder -v toAdapterPath, sourcePath [,sourcePath...], -i directorypath, -t directorypath, -o directorypath
```

## Parameters

`-v`

Creates ActiveX bindings for the CORBA interface.

`toAdapterPath`

Specifies the adapter to be used to create the bindings. For this release of the Application Builder, the `toAdapterPath` path is `OLEAutomation`.

`sourcePath`

Specifies one or more CORBA interfaces for which bindings are to be created. You can also specify a module.

`-i directorypath`

## 7 *Command-Line Options*

---

Specifies the directory location for the C++ header files generated from the command. The default location is `\tuxdir\Include`. If you do not specify this option, the Application Builder uses the last defined values.

`-t directorypath`

Specifies the directory location for the type libraries generated from this command. The default location is `\tuxdir\TypeLibraries`. If you do not specify this option, the Application Builder uses the last defined values.

`-o directorypath`

Specifies the directory location for the MIDL/ODL files generated from this command. The default location is `\tuxdir\TypeLibraries`. If you do not specify this option, the Application Builder uses the last defined values.

### **Example**

The following command creates ActiveX bindings for the `Registrar` and `RegistrarFactory` interfaces:

```
BEAAppBuilder -v OLEAutomation, Registrar, RegistrarFactory, -i  
c:\tuxdir\Include, -t c:\tuxdir\TypeLibraries
```

---

# Glossary

**activation**

The process of preparing an object for execution.

**activation policy**

The policy that determines the in-memory activation duration for a CORBA object.

**ActiveX**

A set of technologies from Microsoft that enables software components to interact with one another in a networked environment, regardless of the language in which the components were created. ActiveX is built on the Component Object Model (COM) and includes OLE functionality, such as OLE Automation.

**ActiveX view**

A representation of a CORBA object that conforms to the ActiveX standards, including implementations of all the interfaces and mapping of data types to those data types supported by ActiveX.

**API**

See *application programming interface*.

**application**

In the BEA Tuxedo CORBA system, a single computer program designed to do a certain type of work.

**applications development environment (ADE)**

A set of tools (often presented or accessed via a GUI) to help programmers build applications.

---

**application programming interface (API)**

The verbs and environment that exist at the application level to support a particular system software product. A set of well-defined programming interfaces (that is, entry points, calling parameters, and return values) by which one software program uses the services of another.

**Application-to-Transaction Monitor Interface (ATMI)**

A UNIX international standard interface that BEA Tuxedo application programs can use to start and commit global transactions, send and receive messages, maintain corrections, manage typed buffers, and perform similar tasks. The ATMI interface is supported by all BEA Tuxedo-based systems and is the basis of the X/Open TX and XATMI interfaces.

**asynchronous process**

A process that executes independently of another process. When a request is processed asynchronously, the client application continues to perform other operations while it waits for the service request to be filled.

**asynchronous request**

A request that lets the client do other work while the request is being processed, enhancing parallelism within an application.

**ATMI**

*See application to transaction monitor interface.*

**attribute**

An identifiable association between an object and a value.

**authenticate**

To reliably determine a user's or processor's identity, often using a password or series of passwords. Once authenticated, an identity can be mapped against the authorization tables of services and objects. This mapping generally takes place in the access control list.

**authentication**

A method consisting of application passwords and security services that is used to verify users and allow users to join applications.

---

### **BEA ActiveX Client**

The component of BEA Tuxedo CORBA that provides interoperability between a BEA Tuxedo domain and the ActiveX object system. The ActiveX Client translates into ActiveX methods the interfaces of CORBA objects that are located in the BEA Tuxedo domain.

### **BEA Tuxedo application**

One or more Tuxedo domains cooperating to support a single business function.

### **BEA Tuxedo client application**

A program that was written to be used with BEA Tuxedo CORBA and that requests services from other applications.

### **BEA Tuxedo CORBA server application**

A program that was written to be used with BEA Tuxedo CORBA and that performs a task requested of it by a client application.

### **BEA Tuxedo CORBA TP framework**

A run-time library of default implementations that the BEA Tuxedo CORBA server application build procedure links to the server application executable image. The TP (transaction processing) framework consists of a set of convenience functions that make it easy for you to write code that does the following:

- a. Initializes the server application and executes startup and shutdown routines.
- b. Ties the server application to BEA Tuxedo domain resources.
- c. Manages objects, bringing them into memory when needed, flushing them from memory when no longer needed, and managing reading and writing of data for persistent objects.
- d. Performs object housekeeping.

### **BEA Tuxedo domain**

For CORBA applications, a collection of servers, services, and associated resource managers defined by a single `UBBCONFIG` file.

For ATMI applications, a specific instance of the BEA Tuxedo system, plus customer server applications, plus a single `UBBCONFIG` file to configure the BEA Tuxedo domain.

---

**BEA Tuxedo foreign client application**

A client application that is implemented on an ORB that is not a product of BEA Systems, Inc., such as Netscape Navigator. The ActiveX Client component of BEA Tuxedo CORBA is not a foreign client application; although the ORB is implemented on a Microsoft product, the ORB is provided by BEA Systems, Inc.

**BEA Tuxedo native client application**

A client application that invokes operations defined in OMG IDL statements to talk to BEA Tuxedo CORBA server applications. Remote and native client applications are the same. Their requests are handled transparently and differently depending on whether or not the applications are co-located on a machine that is running in the BEA Tuxedo domain. BEA Tuxedo remote client applications are typically not located on a machine that is running in the BEA Tuxedo domain. The ActiveX Client component of BEA Tuxedo CORBA is a remote client application.

**BEA Tuxedo remote client application**

A client application that invokes operations defined in OMG IDL statements to talk to remote BEA Tuxedo CORBA server applications using IIOP. Remote and native client applications are the same. Their requests are handled transparently and differently depending on whether or not the applications are co-located on a machine that is running in the BEA Tuxedo domain. The ActiveX Client component of BEA Tuxedo CORBA is a remote client application.

**BEA Tuxedo software**

The BEA Tuxedo product as the customer receives it from BEA Systems, Inc.

**BEA Tuxedo system**

The BEA Tuxedo software and the hardware on which the BEA Tuxedo software is running.

**binding**

The association of the interface of a CORBA object to another object system, such as an ActiveX object system.

**broadcast**

To send the same message to every node on a network.

---

**business object**

An application-level component that can be used in unpredictable combinations. A business object is independent of any single application and represents a recognizable, ordinary entity, such as a document processor. It is a self-contained deliverable that has a user interface state, and that can cooperate with other separately developed business objects to perform a desired task.

**C++**

An object-oriented programming language developed at AT&T Bell Laboratories in the early 1980s. C++ is a “hybrid” language based on the non-object-oriented C language.

**call**

An instruction that is used by an application program to request services.

**class**

A template for an object containing variables and methods representing behavior and attributes. Class can inherit public and protected variables and methods from other classes.

**client**

(1) Software that asks a server to perform a task. In client/server terminology, a client application typically contains the user interface, and the server application typically stores and manipulates the data. A software program that makes a request for a service in a client/server architecture. (2) A process that generates service requests handled by BEA Tuxedo software and receives responses to those requests from BEA Tuxedo software.

**client/server**

A programming model in which application programs are structured as clients or servers. A client program is an application program that requests services to be performed. A server program is an entity that dispatches service routines to satisfy requests from client programs. A service routine is an application program module that performs one or more specific functions on behalf of client programs.

**client stub**

A file created by the IDL compiler when you compile an application’s OMG IDL statements. The client stub contains code that is generated during the client application build process. The client stub maps OMG IDL operation definitions for an

---

object type to the methods in the server application that the BEA Tuxedo domain calls when it is invoking a request. The code is used to send the request to the server application.

**command-line interface**

A style of user interface that allows user interaction by entering command strings at a system prompt.

**commit**

(1) Complete a transaction so that changes are recorded and stable. Protected resources are released. (2) The declaration or process of making a transaction's updates and messages visible to other transactions. When a transaction commits, all its effects become public and durable. After commitment, the effects of a transaction cannot be reversed automatically.

**Component Object Model (COM)**

The object model used on Microsoft platforms. COM is different from CORBA in many ways. For example, there are differences in the mechanisms by which objects are referenced, and in the process by which objects are created.

**COM view**

A representation of an object that conforms to the Component Object Model (COM) standards, including implementation of all necessary interfaces.

**constructor**

A pseudo-method that creates an object. In Java, constructors are instance methods with the same name as their class. Java constructors are invoked using the `new` keyword.

**conversational server**

A server whose services conduct conversations with requesters.

**conversational service**

A service routine that is invoked by means of conversational communication from a client program. When the connection is established and the service is invoked, the client and service exchange data in a manner specific to the application. When the service returns, the connection ends.

---

## **CORBA**

Common Object Request Broker Architecture. A multivendor standard published by the Object Management Group for distributed object-oriented computing.

### **CORBA facilities**

The adopted OMG Common Facilities. Common Facilities provide horizontal end user-oriented frameworks that are applicable to most domains, and defined in OMG IDL.

### **CORBA interface**

A set of operations and attributes. A CORBA interface is defined by using OMG IDL statements to create an interface definition. The definition contains operations and attributes that can be used to manipulate an object.

### **CORBA object**

An entity that complies with the CORBA standard upon which operations are performed. An object is defined by its interface.

### **CORBA ORB**

Any Object Request Broker (ORB) that complies with the CORBA standard. A CORBA ORB is a communications intermediary between client and server applications that typically are distributed across a network. The BEA Tuxedo ORB is a CORBA ORB.

### **core class**

A public class (or interface) that is a standard member of the Java platform. The intent is that the Java core classes, at a minimum, are available on all operating systems where the Java platform runs.

### **daemon**

A system process that processes and runs in the background.

### **database**

A collection of interrelated or independent data items stored together without redundancy to serve one or more applications.

### **database management system (DBMS)**

A program or set of programs that let users structure and manipulate the data in the tables of a database. A DBMS ensures privacy, recovery, and integrity of data in a multiuser environment.

---

**data-dependent routing**

Routing that directs a request to be processed by a particular group based on the value in a data field of the message.

**DBMS**

See *database management system*.

**deployment package**

In ActiveX Client, a self-registering OLE custom control executable that contains the type libraries, Windows registration entries, and application needed to use an ActiveX view of a CORBA object in a client application.

**design pattern**

A document that encapsulates, in structured format, solutions to design problems. These patterns are essentially the articulation of rules and forms that have proved useful in the context of object-oriented application design.

**desktop client**

A client application that operates on a Microsoft desktop platform, such as Windows 2000 or Windows 98. Desktop client applications use the Component Object Model (COM) and communicate with the BEA Tuxedo domain by using the ActiveX Client to translate between COM and CORBA.

**distributed application**

An application that is separated into two or more parts (such as a client and a server) on different computers that communicate through a network.

**distributed application framework**

A middleware suite for building and managing client/server applications. The framework also includes products providing connectivity across multiple operating environments, development services, and management.

**distributed transaction**

A transaction involving multiple transaction managers. In a distributed transaction environment, a client application may send requests to several servers resulting in resource updates at multiple resource managers. To complete the transaction, the transaction manager for each participant (client, servers, and resource managers) must be polled to coordinate the commit process for each participant within its domain.

---

**distributed transaction processing (DTP)**

A form of processing in which multiple application programs update multiple resources (such as databases) in a coordinated manner. Programs and resources can reside on one or more computers access a network.

**domain**

*See BEA Tuxedo domain.*

**dynamic link libraries (DLL)**

A collection of functions grouped into a load module that is dynamically linked with an executable program at run time for a Windows or OS/2 application.

**environmental object**

Any support object that provides independence from the underlying environment (for example, independence from the operating system). The Bootstrap object is an environmental object.

**event**

The occurrence of a condition, state change, or the availability of some information, that is of interest to one or more modules.

**exception**

An abnormal condition, such as an I/O error encountered in processing or data set or a file, or using any resource.

**factory**

Any CORBA object that returns an object reference to other CORBA objects. A factory is located in the server application.

**factory finder**

The object that locates the factories that an application needs. Both client applications and server applications can use a factory finder.

**framework**

The software environment tailored to the needs of a specific application domain. Frameworks include a collection of software components that programmers use to build applications for the domain the framework addresses. Frameworks can contain specialized APIs, services, and tools, which reduce the knowledge a user or programmer needs to have to accomplish a specific task.

---

**garbage collection**

The automatic detection and freeing of memory that is no longer in use. The Java run-time system performs garbage collection so that programmers never explicitly free objects.

**global transaction**

(1) A transaction that spans one or more resource managers comprising local transactions. The Transaction Manager name for a transaction that uses multiple servers or multiple resource manager interfaces and is coordinated as an atomic unit of work. (2) The BEA Tuxedo name for a transaction that uses multiple servers or multiple resource manager interfaces and is coordinated as an atomic unit of work.

**graphical user interface (GUI)**

A high-level interface that uses windows and menus with graphic symbols instead of typed system commands to provide an interactive environment for a user.

**GUI**

See *graphical user interface*.

**host**

A computer that is attached to a network and provides services other than acting as a communication switch.

**identifier**

The name of an item in a Java program.

**IDL**

See *OMG IDL*.

**IDL interface**

A declaration in OMG IDL of an interface to a CORBA object. The interface declaration contains IDL operations and attributes. The OMG IDL interface declaration is used to generate stubs and skeletons for BEA Tuxedo CORBA objects.

See also *Java interface*.

---

## **IOP**

Internet Inter-ORB Protocol. A protocol specified by the Object Management Group (OMG). The IOP enables two or more Object Request Brokers (ORBs) to cooperate to deliver requests to the proper object.

See also *CORBA ORB*.

## **IOP Listener/Handler**

The BEA Tuxedo CORBA feature that enables client applications to communicate with the BEA Tuxedo domain, and the reverse. The IOP listener/handler receives a request from a client application via the IOP protocol, and then sends that request to the appropriate server application within the BEA Tuxedo domain.

## **implementation code**

The method code that you write that satisfies the client application's request on a specific object. The interface defines the operation and is implemented in the method.

## **implementation file**

The file that contains, among other data, method declarations for each operation defined in your OMG IDL statements. You need to implement the method with your business logic. When you build the server application, you provide this implementation file to the BEA Tuxedo CORBA build procedure.

## **inheritance**

The ability to pass along the capabilities and behaviors of one object to another object. When an object inherits behavior from a single interface, it is called single inheritance. When an object inherits behavior from more than one interface, it is called multiple inheritance.

## **instance**

An object instance in C++. Object instances are used as servants for CORBA objects in BEA Tuxedo CORBA.

## **Interface Repository**

An online database that contains the definitions of the interfaces that determine the CORBA contracts between client and server applications.

---

### **Interoperable Object Reference (IOR)**

The entity that associates a collection of tagged profiles with object references. An ORB must create an IOR from an object reference whenever an object reference is passed across ORBs.

### **Java**

An object-oriented programming language modeled after C++ designed to be small, simple and portable across platforms and operating systems.

### **Java Development Kit (JDK)**

A package of software for Java developers that includes the Java interpreter, Java classes, and Java development tools: compiler, debugger, disassembler, applet-viewer, stub file generator, and documentation generator.

### **Java interface**

A declaration used in the Java language to define an abstract interface. Since Java does not have multiple inheritance, a Java class can implement one or more interfaces to provide mix-in functionality.

See also *IDL interface*.

### **Java Runtime Environment (JRE)**

A subset of the Java Development Kit for end users and programmers who want to redistribute the JRE. The JRE consists of the Java Virtual Machine, the Java core classes, and supporting files.

### **Java Virtual Machine**

The part of the Java Runtime Environment responsible for interpreting Java bytecodes.

### **JDK**

See *Java Developer's Kit*.

### **legacy application**

An existing application that needs to be modified or wrapped so that it can gain access to the BEA Tuxedo domain.

### **logical machine (LMID)**

A processing element used in a transaction manager application and given a logical name in the configuration file.

---

**makefile**

A file, referenced by the `make` command, that tells the `make` command how to create each of the files needed to generate a complete program. The makefile contains a list of source files, object files, and dependency information.

**managed object**

An entity (such as a process, a piece of hardware, or system performance) that is defined in the MIB and is controlled by a management device.

**management information base (MIB)**

(1) A BEA Tuxedo system component that provides a complete definition of the object classes and their attributes that together comprise the BEA Tuxedo system.  
(2) A virtual storage database that uses ASN.1 notation. The MIB contains an object that represents each attribute that the system manager software monitors and controls. These objects are defined in ASN.1 notation. Each attribute has an object identifier (OID) that guarantees uniqueness within a standard registration hierarchy.

**mapping**

The relationship between OMG IDL statements and the programming language code that results when the OMG IDL statements are compiled. For example, a C++ IDL compiler maps OMG IDL statements into C++ language bindings.

**method**

A method of a C++ or Java class. User-written methods of C++ or Java classes provide implementation of IDL operations for BEA Tuxedo CORBA distributed objects.

**MIB**

See *management information base*.

**MIB group**

A group of objects, represented by the name or object identifier of an object in the OID tree, that contains a collection of managed objects.

**middleware**

A set of services for building distributed client/server applications, such as services for locating other programs in the network, establishing communication with those programs, and passing information between applications. Middleware ser-

---

vices can also be used to resolve disparities between different computing platforms and to provide a uniform authorization model in multivendor and multioperating system networks.

**model**

A simplified representation of something. The representation is simplified in the sense that some of the details have been abstracted.

**modeling**

A design technique used in developing architecture, simulations, and computer systems.

**multithreading**

Use of a process by several transactions.

**naming context**

An object that contains a set of name associations in which each name is unique.

**object**

An entity defined by its state, behavior, and identity. These attributes (also known as properties) are defined by the object's object system.

See also *CORBA object*.

**object ID (OID)**

A value that uniquely identifies a distributed object of a given interface.

**object implementation**

The code you write that implements the operations defined for an interface.

**object interface**

The interface of an object, as defined in an application's OMG IDL statements. The object interface identifies the set of operations that can be performed on an object, such as withdrawals, deposits, and transfers.

**object model**

The model that represents as objects the overall object-oriented design of an application or system.

---

**object reference**

An identifier that associates an object definition with an instance of the object, such as an employee identification number.

**object system**

A software system that stores, manipulates, and uses a collection of objects according to a set of system-specific standards. An object system specifies how information is exchanged between objects, and how objects are implemented in accordance with an object model, such as CORBA or COM.

**octet**

A byte that consists of eight bits.

**OLE**

Object linking and embedding. A set of Microsoft technologies that address problems in software development, ranging from embedding documents from one application into another application to more complex problems. OLE enables the linking of clients and servers in a manner that is transparent to the user.

**OLE Automation**

A technology that lets software packages expose their unique features to scripting tools and other applications. OLE Automation uses the OLE Component Object Model (COM), but may be implemented independently from other OLE features.

**OMG IDL**

Object Management Group Interface Definition Language. A definition language specified by the OMG for describing an object's interface (that is, the characteristics and behavior of an object, including the operations that can be performed on the object).

**operation**

An action that can be performed by an object.

**Portable Object Adapter (POA)**

A run time library of functions that are built in to the server application executable image. The POA creates and manages object references to all objects used by the application. In addition, the POA manages object state and provides the infrastructure for support of persistent objects and the portability of object implementations between different ORB products. The BEA Tuxedo server application

---

procedure automatically builds the POA into the server application. The BEA Tuxedo CORBA TP framework automatically handles all the server application interactions with the POA.

**request**

A message sent by a client application that identifies an operation to be performed. The message is sent to the Object Request Broker (ORB) and is relayed to the appropriate server application, which fulfills the request.

**resource manager**

An interface and associated software that provides access to a collection of information and processes; for example, a database management system. Resource managers provide transaction capabilities and permanence of actions; they are the entities accessed and controlled within a global transaction.

**rollback**

(1) Terminate a transaction such that all resources updated within a transaction revert to the original state before the transaction started. (2) The event that ends a transaction and nullifies or undoes all changes to resources that were specified during that transaction.

**scalability**

The extent to which developers can apply a solution to problems of different sizes. Ideally, a solution should work well across the entire range of complexity. In practice, however, there are usually simpler solutions for problems of lower complexity.

**security**

The protection of information from unauthorized modification or disclosure and the protection of resources from unauthorized use.

**SecurityCurrent**

The object that provides access to the security features of the system.

**servant**

The instance of the class that implements the interface defined in an application's OMG IDL statements. A servant contains the method code that implements the operations of one or more CORBA objects.

---

**server**

See BEA Tuxedo *server application*.

**server group**

A collection of servers on a machine, often associated with a resource manager. A server group is an administrative unit used for booting, shutting down, and migrating servers.

**Server object**

The object that performs server application initialization functions, creates one or more servants, and performs server application shutdown and cleanup procedures.

**skeleton**

The BEA Tuxedo CORBA Object Request Broker (ORB) component that is specific to the object interface and that assists an Object Adapter in passing requests to particular methods. The skeleton is produced by the IDL compiler and is used at run time by the BEA Tuxedo ORB to invoke specific methods to satisfy requests.

**state**

A description (typically in memory) of the current situation of an object.

**stateless application**

An application that flushes state information from memory after a service or an operation has been fulfilled.

**subscriber**

An application program that subscribes to an event or set of events, and declares what action should take place when an event is posted.

**thread**

A unit of execution or an execution context. An executing sequence of instructions and the memory they manipulate.

**three-tier client/server**

An implementation of n-tier client/server.

**TM**

See *transaction manager*.

---

**transaction**

(1) A complete unit of work that transforms a database from one consistent state to another. In DTP, a transaction can include multiple units of work performed on one or more systems. (2) A logical construct through which applications perform work on shared resources (e.g., databases). The work done on behalf of the transaction conforms to the four ACID Properties: atomicity, consistency, isolation, and durability.

**transaction coordinator**

A system software component that provides the infrastructure that guarantees the integrity and consistency of an operation and the data involved in a transaction.

**TransactionCurrent**

The object that is used to manage transactions. The TransactionCurrent object supports APIs to open and close the resource manager.

**transaction manager**

A system software component that manages global transactions on behalf of application programs. A transaction manager coordinates commands from application programs and communication resource managers to start and complete global transactions by communicating with all resource managers that are participating in those transactions. When resource managers fail during global transactions, transaction managers help resource managers decide whether to commit or roll-back pending global transactions.

See also *transaction coordinator*.

**transaction policy**

The policy that determines the TP framework's interaction between the client request (which may be associated with a transaction) and the servant's transaction context.

**TUXCONFIG**

The binary version of the configuration file for a BEA Tuxedo application. This file is accessed by all BEA Tuxedo processes for all configuration information.

**two-phase commit (2PC)**

A method of coordinating a single transaction across more than one DBMS (or other resource manager). It guarantees data integrity by ensuring that transactional updates are committed in all of the participating databases, or are fully rolled back out of all of the databases, reverting to the state prior to the start of the transaction.

---

**two-tier client/server**

An application development approach that splits an application into two parts and divides the processing between a desktop workstation and a server machine.

**type library**

A shared code repository represented by a single file. It stores data types and interface types.

**UBBCONFIG**

An ASCII version of the configuration file for a BEA Tuxedo application. This is the ASCII representation of the `TUXCONFIG` file.

**use case**

Text that describes how a user will interact with the application that is being designed. The use case reflects the processes the user will follow.

**UserTransaction environmental object**

The object that connects the client application to the BEA Tuxedo CORBA transaction subsystem, wherein the client application can perform operations within the context of a transaction. The UserTransaction object exists only with Java client applications.

**view**

A representation of a CORBA object in the BEA Tuxedo domain that resides in another object system, such as ActiveX.

See also *CORBA object* and BEA Tuxedo *domain*.

**wrap**

To enclose an application in a software layer to make the application available to other applications

**wrapper**

The enclosure that is used to wrap a legacy application to make the legacy application available as an implementation to BEA Tuxedo CORBA client applications.

---

## **XML**

Extensible Markup Language. A language written by the World Wide Web Consortium (W3C) organized by Sun Microsystems, Inc. to put SGML on the World Wide Web.

---

# Index

- A**
- accessing
    - CORBA objects 1-2
  - ActiveX 1-1
    - concepts
      - bindings 1-2
      - views 1-2
    - naming conventions 1-3
  - ActiveX Client
    - overview 1-2
  - ActiveX client applications
    - creating
      - bindings 2-7
      - views 2-7
    - defining security 5-2
    - deploying views 2-15
    - development process 2-2
    - establishing communication with the domain 2-10
    - invoking operations on objects 2-11
    - ISL parameter 2-7
    - loading environmental objects into the Interface Repository 2-5
    - loading interfaces into the Interface Repository 2-5
    - resolving initial references to objects 2-11
    - starting a server application for the Interface Repository 2-6
    - using factories 2-11
    - using security 5-2
    - using the Interface Repository 1-5
    - using transactions 6-1
    - using views 1-2
    - writing 2-9
  - Application Builder
    - creating
      - bindings 2-7
      - deployment packages 2-15
      - type libraries 2-8
      - views 2-7
    - description 1-2
    - how it works 1-2
    - ISL parameter 2-7
    - main user tasks 1-2
    - main window 3-1
    - overview 1-1, 1-2
    - windows 2-3
  - authentication levels
    - getting
      - C++ 5-3
      - Java 5-3
      - Visual Basic 5-3
    - in client applications 5-3
    - supported in the BEA Tuxedo software 1-11
    - TOBJ\_APPAUTH 1-11
    - TOBJ\_NOAUTH 1-11
    - TOBJ\_SYSAUTH 1-11
  - Automation environmental objects
    - loading into the Interface Repository 2-5
    - TOBJIN.IDL 2-5

---

writing declarations for 2-9  
automation server, creating 2-13

## B

### bindings

creating 2-7  
deploying 2-15  
description 1-2

### Bootstrap object

declaration  
    Visual Basic 2-10  
description 1-7  
getting SecurityCurrent object 5-2  
getting TransactionCurrent object 6-2  
resolving initial references  
    Visual Basic 2-10

### buttons

toolbar 3-10

## C

### C++ 5-5

code examples  
    logging on to the domain 5-5  
    PrincipalAuthenticator object  
        C++ 5-3  
    SecurityCurrent object 5-3  
    TransactionCurrent object 6-2  
    transactions 6-4

### C++ Header files

directory location 4-4

changing the default directory locations 4-7

### client applications

using security 5-1  
using transactions 6-4

### code examples

Bootstrap object  
    Visual Basic 2-10

declarations  
    Visual Basic 2-10

### factories

Visual Basic 2-11

### FactoryFinder object

Visual Basic 2-11

### invoking operations

Visual Basic 2-11, 2-12

logging on to the BEA Tuxedo domain  
    5-5

    Visual Basic 2-10

logging on to the domain 5-5

logging on to the Tuxedo domain

    C++ 5-4

    Java 5-4

### PrincipalAuthenticator object

    C++ 5-3

    Java 5-3

    Visual Basic 5-3

### SecurityCurrent object

    C++ 5-3

    Java 5-3

    Visual Basic 5-3

### TransactionCurrent object

    C++ 6-2

    Visual Basic 6-2

### transactions

    C++ 6-4

    Visual Basic 6-4

COM objects, creating instances of 2-14

### CORBA C++ client applications

    defining security 5-2

    using security 5-2

    using the Interface Repository 1-5

    using transactions 6-1

### CORBA interfaces

    creating bindings for 2-7

    loading into the Interface Repository 2-5

### CORBA Java client applications

    defining security 5-2

    using security 5-2

    using the Interface Repository 1-5

    using transactions 6-1

---

CORBA services Object Transaction Service  
6-1  
CORBAServices Security service 5-1  
creating  
deployment packages 4-6  
customer support contact information xiii

## D

deploying applications 4-6  
deployment package  
description 2-15  
directory location 2-15  
Deployment Packages window 3-8  
description 1-1  
development commands  
idl2ir 1-5  
ir2idl 1-5  
irdel 1-5  
development process  
ActiveX client applications 2-2  
security 5-2  
transactions 6-1  
directory location  
deployment package 2-15  
type libraries 2-8  
Display window 3-8  
documentation, where to find it xi  
domains  
authentication level 5-3  
defining security for 5-1  
description 1-5  
establishing communication with  
ActiveX client applications 2-10  
figure 1-5  
logging off 5-5  
logging on with PrincipalAuthenticator  
object 5-4

## E

Edit menu 3-6  
environmental objects 1-6  
Automation 1-6, 2-3  
Bootstrap 1-6  
C++ 1-6  
description 1-6  
FactoryFinder 1-6  
Interface Repository 1-6  
Java 1-6  
SecurityCurrent 1-6  
TransactionCurrent 1-6

## F

factories  
code examples  
Visual Basic 2-11  
creating CORBA objects 1-8  
declaration  
Visual Basic 2-10  
description 1-8  
naming conventions 1-9  
writing declarations for 2-9  
FactoryFinder object  
code examples  
Visual Basic 2-11  
declaration  
Visual Basic 2-10  
description 1-8  
illustrated 1-8  
File menu 3-6

## H

Help menu 3-9

## I

ICF file  
defining transaction policies 6-1

---

idl2ir command  
description 1-5  
loading automation environmental  
objects into the Interface  
Repository 2-5  
loading interfaces into the Interface  
Repository 2-5  
populating the Interface Repository 1-5  
syntax 2-5  
using with ActiveX client applications 2-3

## Interface Repository

commands

idl2ir 1-5

ir2idl 1-5

irdel 1-5

description 1-5

information stored in 1-5

loading

automation environmental objects  
2-5

loading CORBA interfaces into 4-1

starting server application 2-6

## InterfaceRepository object

description 1-13

## ir2idl command

creating an OMG IDL file 1-5

description 1-5

## irdel command

deleting CORBA interfaces from the  
Interface Repository 1-5

description 1-5

## ISL parameter 2-7

using in ActiveX client applications 2-10

using with the Application Builder 2-7

## J

### Java

code examples

PrincipalAuthenticator object

Java 5-3

SecurityCurrent object 5-3

## L

Loading CORBA interfaces into the Interface  
Repository 4-1

## M

main window

Services window 3-1

Workstation Views window 3-1

menu options, description of 3-6

methods

TransactionCurrent object 6-3

MIDL files

directory location 4-4

## N

naming conventions

ActiveX 1-3

factories 1-9

## O

objects

on the Application Builder GUI 3-4

ODL files

directory location 4-5

OMG IDL

description 1-4

online help

printing xii

using vii

window viii

options

Edit menu 3-6

File menu 3-6

Help menu 3-9

View menu 3-7, 3-8

---

Windows menu 3-8

## **P**

PDF location

of online help xii

PrincipalAuthenticator object

arguments 5-4

code examples

C++ 5-3

Java 5-3

Visual Basic 5-3

getting the authentication level 5-3

logging on to the domain 5-4

using in client applications 5-3

printing product documentation xii

## **R**

related information xii

relationship to domains 1-6

## **S**

sample applications

Security 5-2

Transactions 6-2

security

configuring 5-1

getting the PrincipalAuthenticator object  
5-3

getting the SecurityCurrent object 5-2

logging off the domain 5-5

logging on to the domain 5-4

obtaining the authentication level 5-3

overview 5-1

supported authentication levels 1-11

SecurityCurrent object

code examples

C++ 5-3

Java 5-3

Visual Basic 5-3

description 1-11

properties

Credentials 1-11

PrincipalAuthenticator 1-11

using in client applications 5-3

Services window 3-3

description 3-1

starting the Application Builder 4-2

support

technical xiii

## **T**

tasks

changing the default directory locations  
4-7

creating

ActiveX views of CORBA objects  
4-3

creating Deployment Packages 4-6

loading CORBA interfaces into the

Interface Repository 4-1

starting the Application Builder 4-2

TOBJ\_APPAUTH

description 1-11

required arguments 5-4

TOBJ\_NOAUTH

description 1-11

required arguments 5-4

TOBJ\_SYSAUTH

description 1-11

required arguments 5-4

toolbar 3-10

Tools menu 3-8

transaction policies

defining in ICF file 6-1

description 1-12

TransactionCurrent object

methods 6-3

transaction policies 1-12

---

transactions  
    getting the TransactionCurrent object 6-2  
    in client applications 6-4  
    overview 6-1

type libraries  
    creating with Application Builder 2-8  
    directory location 2-8, 4-4  
    loading bindings into development tool 2-8  
    naming conventions 2-8

## U

UBBCONFIG file  
    defining  
        security 5-1  
    starting server application for Interface Repository 2-6

## V

View menu  
    options 3-7

views  
    creating 2-7  
    definition 1-2  
    deploying 2-15  
    description 1-2  
    invoking operations on 2-11, 2-12  
    writing declarations for 2-9

Visual Basic 5-5  
    code examples  
        Bootstrap object 2-10  
        factories 2-11  
        FactoryFinder object 2-11  
        invoking operations 2-11, 2-12  
        logging on to the domain 5-5  
        PrincipalAuthenticator object 5-3  
        SecurityCurrent object 5-3  
        TransactionCurrent object 6-2

    transactions 6-4  
    declarations for 2-10  
        Bootstrap object 2-10  
        FactoryFinder object 2-10  
    loading type libraries for bindings 2-8

Visual Basic samples  
    chat room sample 2-13

## W

windows  
    Services 3-3  
    Workstation Bindings 4-4  
    Workstation Views 3-3

Windows menu 3-8

Workstation Bindings window 3-8, 4-4

Workstation Views window 3-3  
    description 3-1