Installation and Administration Guide

VERSION 3.3

VisiBroker™ for C++

Inprise Corporation, 100 Enterprise Way
Scotts Valley, CA 95066-3249

Printed in the U.S.A.

VCA0033WW21000 1E0R798
9899000102-9 8 7 6 5 4 3 2 1
D4

Contents

Tables

Figures

Chapter

1

Preface

The *VisiBroker for C++ Installation and Administration Guide* describes how to install and configure VisiBroker for C++ and its associated components for application development and deployment environments. Also, it describes how to execute client applications and servers, and provides information on the VisiBroker administration tools.

The Preface lists the contents of the *VisiBroker for C++ Installation and Administration Guide*, describes the conventions used throughout the guide, and tells you where to find more information relating to this product.

Overview

VisiBroker for C++ allows you to develop and deploy distributed object-based applications, as defined in the Common Object Request Broker Architecture (CORBA) specification.

Who should read this guide

This guide is intended for system administrators and those responsible for installing, configuring, and administering VisiBroker for C++ for application development and deployment.

Organization of this manual

This manual includes the following sections:

- Chapter 2, “VisiBroker basics” provides a general overview of VisiBroker for C++ and CORBA and a description of the product components.

- Chapter 3, “Installing VisiBroker” describes the VisiBroker Developer for C++ installation processes, by platform.
- Chapter 4, “Configuring VisiBroker” describes how to configure the VisiBroker environment using command-line options and environment settings.
- Chapter 5, “Starting runtime support services” describes the runtime support services provided with VisiBroker for C++.
- Chapter 6, “Using Administration tools” describes the administration tools provided with VisiBroker for C++.
- Chapter 7, “Deploying and executing applications” describes how to deploy and run client programs and server objects that have been developed with VisiBroker for C++.
- Appendix A, “Error and log files” describes the VisiBroker for C++ log files and error messages.

Typographic conventions

This manual uses the following conventions:

Convention	Used for
boldface	Bold type indicates that syntax should be typed exactly as shown. For UNIX, bold type indicates database names, filenames, and similar terms.
italics	Italics indicates information that the user or application provides, such as variables in syntax diagrams. It is also used to introduce new terms.
`computer`	Computer typeface is used for sample command lines and code.
UPPERCASE	Uppercase letters indicate Windows file names.
[]	Brackets indicate optional items.
{}	Curly brackets are used in the more complex syntax statements to show a required item.
...	An ellipsis indicates that the previous argument can be repeated.
|	A vertical bar separates two mutually exclusive choices.
⋮	A column of three dots indicates the continuation of previous lines of code.

Platform conventions

This manual uses the following symbols to indicate that information platform-specific:

W Windows NT and Windows 95

NT Windows NT only

95 Windows 95 only

U All UNIX platforms

Where to find additional information

For more information about VisiBroker for C++, refer to the following information sources:

- *VisiBroker for C++ Programmer's Guide* provides information on developing distributed object-based applications in C++ for Windows and UNIX platforms.
- From your web browser, you can open the **vbcpp.html** file, located in the directory where you installed VisiBroker, to view the release notes and other important information.
- VisiBroker also publishes a variety of white papers concerning distributed-computing and VisiBroker. To obtain copies of white papers, visit Inprise's web site at http://www.inprise.com/visibroker.

For more information about the CORBA specification, refer to the following sources:

- *The Common Object Request Broker: Architecture and Specification*—96-03-04. This document is available from the Object Management Group and describes the architectural details of CORBA. You can access the CORBA specification using the World Wide Web at http://www.omg.org.
- *IDL to C++ Language Mapping 1.1.* This document is available from the Object Management Group and describes the Interface Definition Language mapping for C++.

Contacting Inprise Technical Support

Inprise offers a variety of support options to help you get the most from your Inprise products. For information about these options, see the "Services" section of Inprise's web site at http://www.inprise.com, or contact our Sales Department at 1-800-632-2864.

When contacting VisiBroker Technical Support, be prepared to provide complete information about your environment, the version of the VisiBroker product you are using, and a detailed description of the problem.

Chapter

2

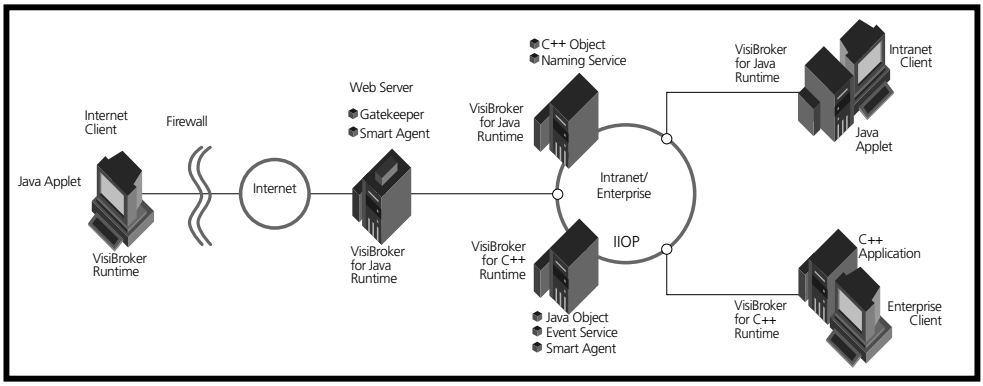
VisiBroker basics

This chapter introduces VisiBroker for C++ basics for System Administrators. After reading this chapter you will be ready to install, configure, and administer VisiBroker for C++. This chapter contains the following major sections:

Overview

VisiBroker for C++ is a complete CORBA 2.0 Object Release Broker (ORB) that supports the development, deployment, and management of distributed C++ applications that are open, flexible, and interoperable across a variety of platforms and operating systems. An example of the VisiBroker architecture is shown in Figure 2.1. Objects built with VisiBroker for C++ are easily accessed by Web-based applications that communicate using OMG's Internet Inter-ORB Protocol (IIOP), the standard for communication between and among distributed objects running on the Internet and intranets. VisiBroker has a native implementation of IIOP, ensuring high-performance, interoperable distributed-object applications for the Internet, intranets, and enterprise computing environments.

Figure 2.1 Architecture of VisiBroker



VisiBroker for C++ components

In addition to VisiBroker for C++ (the ORB), two other components are available with this product. They include the Naming and Event Services.

Naming Service

The VisiBroker Naming Service allows you to associate one or more *logical* names with an object implementation and store those names in a *namespace*. It also lets client applications use this service to obtain an object reference using the logical name assigned to that object. See the *VisiBroker Naming and Event Services Programmer's Guide* for more details on the Naming Service.

Event Service

The VisiBroker Event Service provides a facility that de-couples the communication between objects. It provides a supplier-consumer communications model that allows multiple supplier objects to send data asynchronously to multiple consumer objects through an event channel. See the *VisiBroker Naming and Event Services Programmer's Guide* for more details on the Event Service.

What is CORBA?

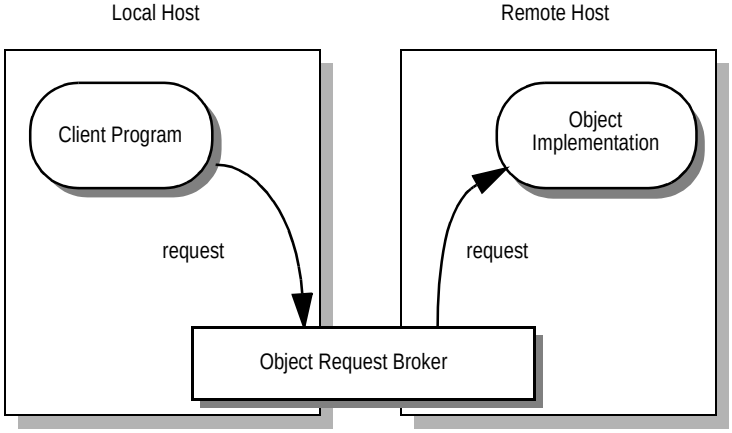
The Common Object Request Broker Architecture (CORBA) specification was adopted by the Object Management Group to address the complexity and high cost of developing distributed applications. CORBA uses an object oriented approach for creating software components that can be reused and shared between applications. Each object encapsulates the details of its inner workings and presents a well defined interface, which reduces application complexity.

The VisiBroker Object Request Broker (ORB) for C++ in Figure 2.2 connects a client application with the objects it wishes to use. The client program does not need to know whether the object resides on the same computer or is located on a remote computer somewhere on the network. The client program only needs to know the object's name and understand how to use the object's interface. The ORB takes care of the details of locating the object, routing the request, and returning the result.

Note An object implementation may itself be a client program for other requests.

The ORB is not a separate process. It is a collection of libraries and network resources that integrates within end-user applications, and allows your client applications to locate and use objects.

Figure 2.2 Client program acting on an object through an Object Request Broker (ORB)



Included with VisiBroker for C++

The items listed below are part of the VisiBroker product line and may be installed with certain VisiBroker products. However, depending on which VisiBroker product contains the particular item, a separate license and install and setup process may be required. The installation of these items may not be required if you intend to deploy, but do not actually develop, VisiBroker applications.

Java Runtime package

A Java Runtime Environment (JRE) is a subset of the complete Java development environment. It is bundled with the VisiBroker development tools, but is only necessary for deployment when using the interface repository.

Runtime support services

VisiBroker ORB for C++ includes several runtime support services. See Chapter 5, “Starting runtime support services” for a complete description.

Smart Agent

The Smart Agent (osagent) is used by applications to locate the objects they wish to use. It is a process that must be started on some host within your network. See the *VisiBroker for C++ Programmer's Guide* for further details on the Smart Agent.

Object Activation Daemon

The Object Activation Daemon (OAD) enables objects to be activated automatically when they are needed by a client application. This reduces overhead by allowing servers that implement objects for client applications to be started on demand, rather than running continuously. See the *VisiBroker for C++ Programmer's Guide* for details on the Object Activation Daemon.

Interface Repository

The Interface Repository is an online database of meta information about object types. Meta information stored for ORB objects includes information about modules, interfaces, operations, attributes and exceptions. Applications that use the dynamic interfaces require that an Interface Repository be available. See the *VisiBroker for C++ Programmer's Guide* for details on the Interface Repository.

VisiBroker development package

VisiBroker for C++ can be used during either the development or deployment phase. The VisiBroker developer product includes the following components:

- Administration and development tools
- C++ header files
- VisiBroker ORB libraries

To add more functionality to your development and testing environment, you might want to include the VisiBroker Manager. See “VisiBroker Manager” on page 2-6 for more information on VisiBroker Manager.

Administration tools

VisiBroker provides a complete set of tools for administering your VisiBroker ORB. These tools can be used within scripts, giving you greater flexibility and control over administration tasks. See Chapter 6, “Using Administration tools” for details on the administrator tools.

The following tools are used to administer the VisiBroker ORB during development.

Tool	Purpose
`irep`[1]	Used to manage the Interface Repository.
`oad`	Used to manage the Object Activation Daemon (OAD).
`oadutil list`[1]	Lists ORB object implementations registered with the OAD.
`oadutil reg`[1]	Registers an ORB object implementation with the OAD.
`oadutil unreg`[1]	Unregisters an ORB object implementation with the OAD.
`osagent`	Used to manage the Smart Agent.
`osfind`	Reports on objects running on a given network.

1. Written in Java

Development tools

VisiBroker provides a complete set of tools, including an IDL compiler, for developing C++ applications that use distributed objects. See the Chapter 2, "Programmer tools" in the *VisiBroker for C++ Reference* for details on the development tools.

The following tools are used during the development phase:

Tool	Purpose
`idl2ir`[1]	This tool allows you to populate an interface repository with interfaces defined in an IDL file.
`idl2cpp`[1]	This tool generates C++ stubs and skeletons from an IDL file.

1. Written in Java

Deploying VisiBroker applications

The deployment phase occurs when a developer has created client programs or server applications that have been tested and are ready for production. At this point a system administrator is ready to deploy the client programs on end-users' desktops or server applications on server-class machines.

The VisiBroker ORB must be installed on every client and server machine. If Smart Agent capability is required, at least one Smart Agent must be running on the network. See "Starting the Smart Agent" on page 5-2, for details.

VisiBroker ORB

The VisiBroker's ORB libraries for C++ enable client and server applications to use and provide distributed objects. Included in the VisiBroker ORB are the runtime support services.

VisiBroker CORBA compliance

VisiBroker for C++ is fully compliant with the CORBA specification (version 2.1) from the Object Management Group (OMG). For more details, refer to the CORBA specification on OMG's web site at http://omg.org.

Not included with VisiBroker

The following items are not included with VisiBroker for C++, but are needed to develop or deploy applications.

C++ compiler

In the C++ development environment, a C++ compiler is required to compile the C++ code. For more information on the system requirements for VisiBroker, refer to "Installing VisiBroker for C++ and its associated components" on page 3-3.

Java-enabled web browser

The HTML files can be viewed in any Java-enabled web browser—such as Netscape Navigator or Microsoft's Internet Explorer. The README.htm file is the high-level file shipped on the CD and meant to be viewed first. You can obtain these Java-enabled web browsers by navigating to one of the following URLs:

- http://www.netscape.com
- http://microsoft.com/ie

VisiBroker Manager

VisiBroker Manager is a separate product that provides intuitive graphical tools for visualizing, monitoring, and managing distributed objects from a central location. VisiBroker Manager provides an interactive graphical interface to your network's naming service, interface repositories, implementation repositories, and Smart Agents. For more information, see http://www.inprise.com/visibroker/.

Interoperability with VisiBroker for Java

C++ applications created with VisiBroker for C++ can communicate with object implementations developed with VisiBroker for Java, which is a separate product. Simply use the same IDL you used to develop your C++ application as input to the IDL compiler supplied with VisiBroker for Java. You may then use the resulting Java skeletons to develop the object implementation in Java. See the *VisiBroker for C++ Programmer's Guide* for details.

Object implementations written with VisiBroker for C++ will also work with clients written in VisiBroker for Java. In fact, a server written with VisiBroker for C++ will work with *any* CORBA 2.0 compliant client; a client written with VisiBroker for C++ will work with *any* CORBA 2.0 compliant server.

Interoperability with other ORB products

CORBA 2.0 compliant software objects communicate using the Internet Inter-ORB Protocol (IIOP) and are fully interoperable, even when they are developed by different vendors who have no knowledge of each other's implementations. VisiBroker's use of IIOP allows client and server applications developed with VisiBroker for C++ to be used with a variety of ORB products from other vendors.

Chapter

3

Installing VisiBroker

This chapter explains how to install VisiBroker Developer for C++ and its associated components on the desired platform, describes the system requirements, and also provides you with guidelines on compatibility with earlier versions. This chapter contains the following major topics:

Introduction

By following the instructions in this chapter you will be able to install or uninstall VisiBroker for C++ and its associated components. However, before you begin the installation, make sure that you read the instructions in this chapter and in Chapter 4, “Configuring VisiBroker.” Both chapters will provide you with the information you need to correctly set your environment settings and configure VisiBroker.

In addition to the product, the VisiBroker distribution also includes online documentation files. For instructions on how to copy the documentation to your

machine or to view the documentation from the CD, refer to the section “Viewing VisiBroker documentation” on page 3-16.

VisiBroker system requirements

The following information describes the system requirements for the UNIX and Windows versions of VisiBroker for C++ and the disk space requirements.

UNIX systems requirements

The following information describes the system requirements for installing VisiBroker for C++ on a UNIX workstation.

For all UNIX versions

- A CD ROM drive or connection to the VisiBroker website

For Solaris

- Solaris 2.5, 2.5.1, or 2.6 operating system
- Sun SparcWorks C++ compiler, version 4.0.1 or greater

Windows systems requirements

The following information describes the system requirements for installing VisiBroker for C++ on a Windows workstation.

- Operating systems—Windows NT 3.5.1 or 4.0 or Windows 95
- A CD ROM drive or connection to the VisiBroker website
- Microsoft Visual C++, version 4.1 or greater

Disk space requirements

The following table describes the disk space requirements for VisiBroker for C++ and its associated components.

Table 3.1 VisiBroker for C++ free disk space requirements

Package	Space
VisiBroker for C++ 3.3 Developer Package	17 Mb
VisiBroker for C++ 3.3 Runtime Package	15 Mb
VisiBroker Naming Service 3.3 Developer Package	3.5 Mb
VisiBroker Naming Service 3.3 Runtime Package	2 Mb
VisiBroker Event Service 3.3 Developer Package	2 Mb
VisiBroker Event Service 3.3 Runtime Package	1.5 Mb

Checking the Release Notes

Make sure that you read the Release Notes. This file contains important product information that was not available at the time that this guide was published. The Release Notes are located in /vbroker/doc/vbcrel.html.

Installing VisiBroker for C++ and its associated components

The following instructions tell you how to install VisiBroker for C++ or the VisiBroker components (Naming and Event Services) on your specific platform. The CD shipped to you contains the distribution package. To install VisiBroker you need an authorized license key or you can choose to install the product for evaluation.

Windows-specific installation points

During the Windows install process, you will be asked to set or accept the default values for the OSAGENT_PORT and VBROKER_ADM *registry values*. These options can later be set as environment variables. If set later, these variables will override the registry settings. You may want to review the information on the environment variables in Chapter 4, "Configuring VisiBroker," before you begin the install process. In addition, the VisiBroker Registry Edit tool gets installed in the VisiBroker program group and allows you to change these settings.

If you plan to install on Windows NT you should log in as Administrator before beginning the installation.

If you are installing the Windows version of VisiBroker, make sure that you close all active Windows applications before you begin the installation.

UNIX-specific installation points

During the UNIX install process, the OSAGENT_PORT and VBROKER_ADM environment variables are added to automatically generated shell files. You may want to review the information on the environment variables in Chapter 4, "Configuring VisiBroker" before you begin the install process.

If you plan to install VisiBroker on the system in a location that only root or superuser would have write access to, you must login as root or superuser before beginning the installation.

Solaris specific

If you install the UNIX version of VisiBroker on a system that is running Volume Manager, you will not need to mount the CD. If the system is not running Volume Manager or if Volume Manager is turned off, follow the UNIX installation instructions provided in this chapter to mount the CD.

Installing on UNIX platforms

The following instructions describe how to copy the distribution package from the VisiBroker CD to your hard drive and then complete the installation process.

Installing VisiBroker on a UNIX platform

1 Login as superuser.

If you want to install VisiBroker in the general system area, you must log in as root or superuser to have the correct privileges.

2 If you are installing a version of the product downloaded from the Inprise website, use the following command to extract the contents of the tar file to a temporary directory and then proceed to Step 14.

```
tar xvf vbcpp32.tar
```

The address of the VisiBroker website is http://www.inprise.com/visibroker.

3 Insert the VisiBroker CD in the CD ROM drive.

4 View the Readme.htm file before continuing for additional information on how to access product documentation and how to install the product or the evaluation software.

The Readme.htm file is at the top-level of the directory structure on the CD.

Follow steps 5 through 8 to mount the CD for Solaris systems

5 Enter the following command to determine if the Volume Manager is running:

```
/usr/bin/ps -ef | grep vold
```

Look for a response similar to the following to see a list of all running processes:

```
root 247 1 0 July 30? 0:/00 /usr/sbin/vold
```

6 If the Volume Manager is running, then proceed to Step 12.

7 If the Volume Manager is not running, enter the following commands to mount the CD:

```
mkdir /cdrom/vb_prod_cd
/usr/sbin/mount -f hsfs -r /dev/dsk/c0t6d0s2 /cdrom/vb_prod_cd
```

8 Enter the following command to change to the CD ROM directory:

```
cd /cdrom/vb_prod_cd
```

Follow steps 9 through 13 to mount the CD for HP-UX systems

9 Enter the following command to determine if the Portable File System daemons are running:

```
/usr/bin/ps -ef | grep pfs
```

Look for a response similar to the following to see a list of all running processes:

```
root 4159 4158 0 15:55:58 ttypl 0:00 pfsd.rpc
root 4158 4149 0 15:55:58 ttypl 0:00 pfsd
root 4150 4149 0 15:55:53 ttypl 0:00 pfs_mountd
root 4151 4150 0 15:55:53 ttypl 0:00 pfs_mountd.rpc
```

10 If the Portable File System daemons are running, then proceed to Step 12.

11 If the Portable File System daemons are not running, enter the following commands to start the daemons:

```
pfs_mountd &
pfsd &
```

12 Enter the following command to mount the CD:

```
pfs_mount /dev/dsk/c0t0d0 /cdrom
```

13 Enter the following command to change to the CD ROM directory:

```
cd /cdrom
```

Follow the remaining steps to complete the installation.

14 Begin installation.

Type the following command to execute the installation program:

```
./installvb
```

The VisiBroker installation script begins and displays the license agreement.

15 Review the license agreement.

VisiBroker prompts you to agree to the terms of the license agreement. The following message and prompt appears:

```
Do you agree to the terms of the license agreement [y/n]? (q to quit): y
```

Note You must agree to the terms of the license agreement to continue the install process.

16 Enter the registration information.

VisiBroker prompts you to enter your name, company name, and product license key. The following messages and prompts appear:

```
Please Enter Your Name (q to quit): Your Name
Please Enter Your Company's Name (q to quit): Company Name
Please Enter Your License Key (q to quit): AAAAAAAA-AAAAAAAA-AAAAAAAA-AAAA
```

Note You do not need a License Key to install an evaluation copy or version.

If you are installing a licensed version of the product, make sure that you enter the Company name and license key exactly as provided by Inprise.

Verify that your registration information is correct. The following message and prompt appears:

```
License Key Has Been Validated.
User Name: Your Name
Company Name: Company Name
License Key:AAAAAAAA-AAAAAAAA-AAAAAAAA-AAAA
Is this information correct [y/n]? [Default: y] (q to quit):
```

17 Select the products you want to install from the list of VisiBroker products by choosing the product numbers at the following prompt:

```
Enter Your Choice [press enter to continue] (q to quit):
```

The product options may appear as either:

```
VisiBroker for C++ (Licensed)
[1]    * Developer Package
[2]    * Runtime Package
VisiBroker Event Service for C++ (Evaluation)
[3]    * Developer Package
[4]    * Runtime Package
VisiBroker Naming Service for C++ (Licensed)
[5]    * Developer Package
[6]    * Runtime Package
```

or

```
VisiBroker for C++ (Licensed)
[1]    * Debug Runtime Package
VisiBroker Event Service for C++ (Evaluation)
[2]    * Debug Runtime Package
VisiBroker Naming Service for C++ (Licensed)
[3]    * Debug Runtime Package
```

Note If you installing the Debug Runtime Package, the only options that appear in the product list are the Debug Runtime libraries, as shown in the second part of the above example.

Note An asterisk (*) next to the product name indicates that it is selected for install. If you do not have a License Key for a specific product, the word (Evaluation) will appear next it in the product list. You have the option to preview the product by installing the evaluation version. If you do not want to install the evaluation version, unselect it from your choices.

18 To select either the Developer or Runtime Package or the Debug Runtime Package for your product, enter the appropriate number at the prompt.

For example, to install only the VisiBroker for C++ Runtime Package instead of both the Developer and Runtime Packages, enter 1 to deselect the Developer package when prompted for your choice.

Note The Runtime Packages are subsets of the Developer Packages and can be installed on machines that will not be used for development. The Debug Runtime Package is a separate installation package.

19 Enter the installation path for VisiBroker. The following message and prompt appears:

```
Please enter the path for this installation.
[Default: /usr/local/vbroker] (q to quit):
```

Note All VisiBroker products must share the same installation directory. If the directory you selected does not exist, you will be prompted to create it, use the default, or quit.

VisiBroker creates the installation directory and than displays the list of product options.

Follow steps 20 and 21 when installing the core ORB product

20 Enter the OSAGENT_PORT value. The following message and prompt appears:

```
Enter the Visigenic Smart Agent Port [Default: 14000] (q to quit):
```

Note Entering this value sets OSAGENT_PORT in the shell script that is generated during this install.

VisiBroker uses this value to find VisiBroker's Smart Agent, which is the service that locates objects. You can override this value by resetting it to a different port number. Setting OSAGENT_PORT to the same value, allows all systems (on your subnet) to communicate with one another. Setting OSAGENT_PORT to a unique value on your subnet means that objects on the system will not be available through the Smart Agent to other systems.

21 Enter the VBROKER_ADM value. The following message and prompt appears:

```
Enter the VisiBroker adm directory [Default: /usr/local/vbroker/adm] (q to quit):
```

Note Entering this value sets VBROKER_ADM in the shell script that is generated during this install.

VisiBroker uses this value to find the directory where the repository information and optional configuration files are located. By default, this should be the set to the adm directory in the installation. You can override this value by resetting it to a different directory.

22 Verify that you want the install the components. The following message and prompt appears:

```
The installation is now prepared to copy product to /usr/local/vbroker.
Do you wish to continue [y/n]? [Default: y] (q to quit):
```

VisiBroker will install the components and generate the following shell scripts:

- vbroker.sh
- vbroker.csh

The following message displays when the installation is complete:

```
VisiBroker for C++ (Solaris) 3.3 Installation Complete.
A transcript of the installation process is located at: /usr/local/vbroker/installvb.log1
To start using VisiBroker, open the file
      /usr/local/vbroker/vbcpp.html
in your web browser. The user's environment cam be set using the scripts generated
during installation.
If you wish to uninstall some or all of the components, run "removevb" from within /usr/
local/vbroker/uninst.
Hit return to exit...
```

Note See the following two topics for information on setting up your environment and uninstalling VisiBroker.

Setting up the environment on UNIX

1. Use the following command to change to the installation directory.

   ```
   cd <installation dir>
   ```

2. Use the following command to set up your environment to work with this installation of VisiBroker:

   - for Bourne shell users, enter:

     ```
     . <installation dir>/vbroker.sh
     ```

   - for C-shell users, enter:

     ```
     source <installation dir>/vbroker.csh
     ```

Use the following instructions to unmount the CD from the CD ROM drive once the installation is complete.

Follow step 1 to unmount the CD on Solaris.

1. Enter the following commands to unmount the CD:

   ```
   umount /cdrom/vb_prod_cd
   ```

   Use the above command only if the Volume Manager is not running, otherwise use the following command.

   ```
   eject cdrom
   ```

Follow step 2 to unmount the CD on HP-UX.

2. Enter the following commands to unmount the CD:

   ```
   pfs_umount /cdrom
   ```

3. Use the following command to end the session.

   ```
   exit
   ```

   Remove the CD from the CD ROM drive.

Uninstalling VisiBroker on Solaris

Before you install a new version of VisiBroker or upgrade from an evaluation copy, you should uninstall the existing VisiBroker product. However, if you are running more than one VisiBroker product on your system, make sure you do not choose to unregister services or delete shared binaries that are required for your remaining VisiBroker products. See the following instructions.

To uninstall VisiBroker for C++:

1. To uninstall some or all of the components, execute the following command from within /usr/local/vbroker/uninst:

   ```
   ./removevb
   ```

2. Select the number for the component you want to uninstall.

   During the uninstall process, VisiBroker prompts you to choose the components to be uninstall from the following menu.

   ```
   Select A Component to Uninstall:
   1: VisiBroker for C++ 3.3 (Developer Package)
   2: VisiBroker for C++ 3.3 (Runtime Package)
   3: VisiBroker for C++ Naming Service 3.3 (Developer Package)
   4: VisiBroker for C++ Naming Service 3.3 (Runtime Package)
   5: VisiBroker for C++ Event Service 3.3 (Developer Package)
   6: VisiBroker for C++ Event Service 3.3 (Runtime Package)
   Enter the # of your choice (0=quit):
   ```

Installing on Windows platforms

The following instructions tell you how to copy the distribution package to your hard drive and then complete the installation process.

Before beginning the VisiBroker installation on Windows, close any other Windows applications currently running.

Copying from a CD to a Windows Platform

1. Log into Windows as the Administrator if you are installing VisiBroker on Windows NT.
2. If you are installing a version of the product downloaded from the VisiBroker website, execute the following command to run the executable and then proceed to Step 6.

   ```
   vbcpp32.exe
   ```

   The address of the VisiBroker website is http://www.inprise.com/visibroker.
3. Insert the VisiBroker CD in the CD drive.

4 View the Readme.htm file before continuing for additional information on how to access product documentation and on how to install the product or the evaluation software.

The Readme.htm file is at the top-level of the directory structure on the CD. Use Windows Explorer to view the contents of the CD. Double-click the CD ROM drive icon and select the Readme.htm file.

5 In the Windows Explorer, navigate to that location of the installer and Execute the setup program by double clicking on the setup.exe in to execute the installer.

The VisiBroker for C++ Welcome screen displays.

6 Review the instructions on the VisiBroker for C++ Welcome Screen and then click Next to continue.

This screen displays a message reminding you, if not already done, to close any active Windows applications before you continue.

7 Click Next.

The Software License Agreement displays.

8 Read the License Agreement and click YES to continue the install procedure. If you click NO, a dialog displays where you may choose to resume the install process or to exit Setup.

Note You must accept the License Agreement to continue the install process.

9 At the User Information Screen, enter your name, company name, and the VisiBroker for C++ product license number and click Next to continue.

You must enter a valid license key for the product that you want to install. If you enter an incorrect key, the following message will display:

```
Warning invalid key entered. Please reenter key. Company name and license key must be
exactly like those provided by Visigenic.
```

Enter the correct license key to continue. Make sure that you enter all of the dashes in the license key sequence.

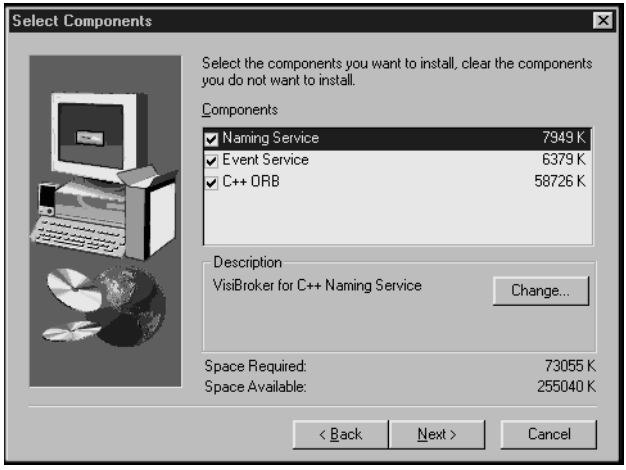
Note You must enter the company name and license key exactly as provided by Inprise. Otherwise, leave the Serial field blank to install the evaluation versions of all components.

10 At the Choose Destination Location dialog, enter the name of the desired directory where the setup process will install VisiBroker and click Next to continue.

You can accept the default options, click the Browse button to locate the desired directory, or click the Cancel button exit the install process. The default directory path is: C:\Visigenic\vbroker.

Note VisiBroker requires you to install all VisiBroker products and services in the same directory.

11 At the Select Components Screen, select the VisiBroker for C++ components that you want to install.



If you do not want to install a selected component, click the check box next to the item to remove the check. If you want to install a sub-component, refer to the following step. Otherwise, click Next to continue and proceed to step 12.

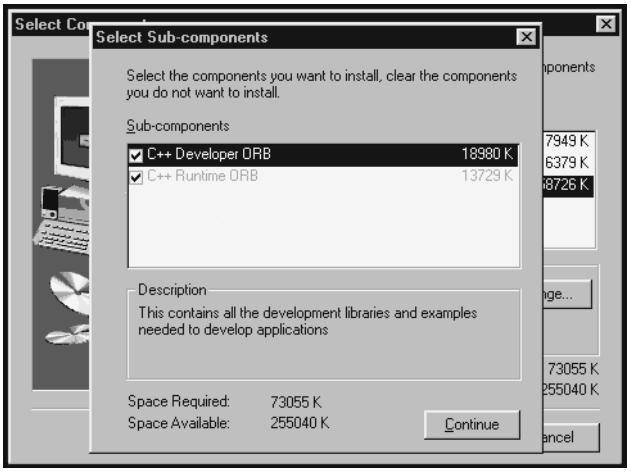
To choose sub-components:

12 At the Select Components Screen, shown in the previous step, select a VisiBroker for C++ component from the list and click the Change button in the Description field to view a list of the sub-components for that option in the Select Sub-component Screen.

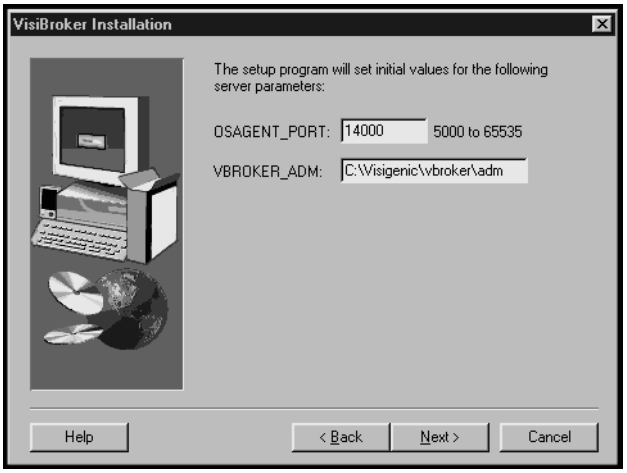
Note The Runtime package is a sub-set of the Developer package.

13 At Select Sub-component Screen, select the desired sub-component by clicking the check box next to the option.

Once you have made your choice, click the Continue button to return to the Select Components Screen.



14 At the VisiBroker Installation screen, enter the OSAGENT_PORT and VBROKER_ADM registry values and click **Next**.



These registry values are set in the system registry. The default values are:

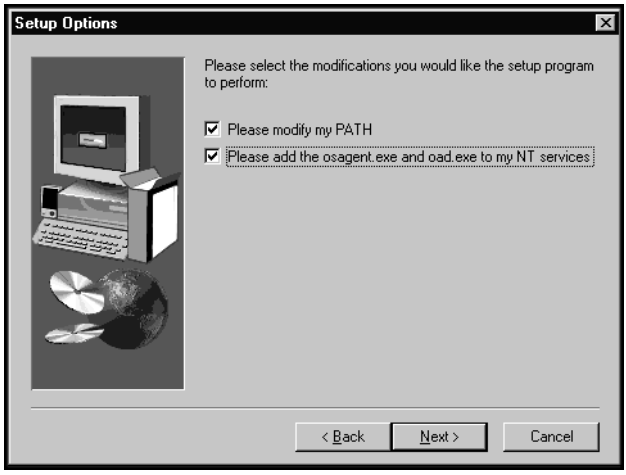
- OSAGENT_PORT is 14000
- VBROKER_ADM path is C:\Visigenic\vbroker\adm

If you do not want to use the defaults, enter your own values for both options. However if you later set the OSAGENT_PORT and VBROKER_ADM environment variables, those settings will override these registry value settings.

Note For more information on setting environment variables see Chapter 4, “Configuring VisiBroker.”

15 At the Setup Options screen, update the system by setting the following options as appropriate and click Next:

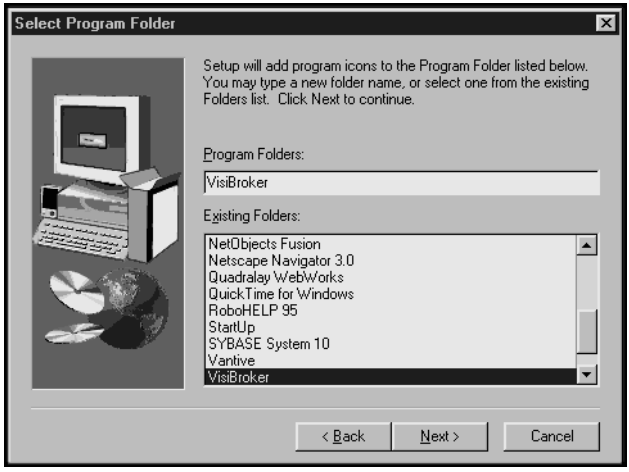
- Please modify my PATH
- Please add the osagent.exe and oad.exe to my NT services (NT platform only)



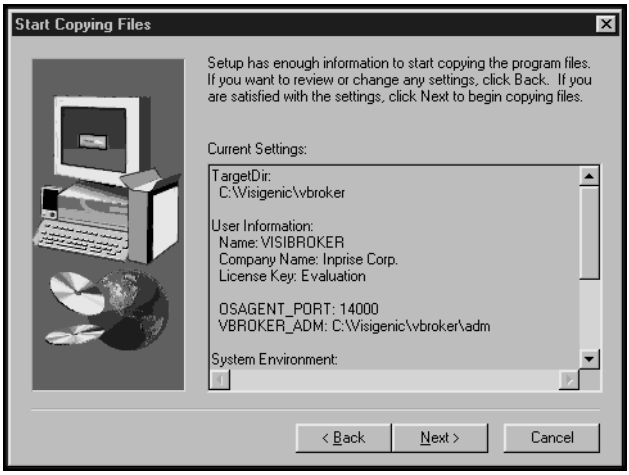
Note If you are installing the Windows NT version, you can register the ORB services as NT services. This way you can start the ORB services from the NT Service Window.

16 At the Select Program Folder screen, enter the name of the Program Folder that displays in the Windows Start Menu or use the default name and then click Next to continue.

The default name is VisiBroker.



17 At the Start Copying Files screen, review the options that you set and click Next to continue the install.



VisiBroker copies all necessary files to your destination directory and creates a VisiBroker Program Group. For Windows 95 users, if you choose to modify your path, the following message displays in an information dialog:

```
Your old autoexec.bat has been saved to autoexec.X.
Click OK to continue.
```

18 If you installed NT Services or redirected your PATH in Windows 95, you will see a screen asking you to select to restart your computer now or later. Otherwise, you will see the following Setup Complete screen.

At the Setup Complete dialog, click Finish to complete the setup.

19 Click Finish to complete the install.

Finishing the installation

After you have installed VisiBroker for C++, your web browser will display the Welcome to VisiBroker page. Be sure to read the release notes, which contain the latest information on VisiBroker for C++.

Uninstalling VisiBroker on Windows

Before you install a new version of VisiBroker or upgrade from an evaluation copy, you should uninstall the existing VisiBroker product.

To uninstall VisiBroker for C++ use one of the following methods:

1. Go to the Add/Remove Programs Properties dialog in your Windows' Control Pane, select VisiBroker for C++ from the list of installed software, and click Add/Remove or choose the Uninstall VisiBroker item from the VisiBroker Program Group.

   If you registered ORB services as NT services or installed shared libraries, you will be prompted to unregister them during the uninstall process.

Note If you have more than one VisiBroker product installed you should not unregister ORB services or uninstall the shared binaries, as they can be used by the other VisiBroker product.

2. Click YES when prompted to uninstall the VisiBroker Smart Finder. Uninstall prompts you to specifically unregister each of the services.
3. Click YES or click YES TO ALL, when prompted, to delete the shared binaries.

Note Click NO if you have VisiBroker for Java installed.

Viewing VisiBroker documentation

In addition to the printed documentation shipped with the product, an electronic version of the documentation is included on the VisiBroker CD. The following instructions tell you how to either view the documentation from the CD or to copy it to your system.

To view the documentation directly off the CD:

1. Insert the VisiBroker CD in the CD ROM drive.
2. Select the Readme.htm file in the top level directory of your CD and view the file in your web browser.
3. Scroll down to the section called Accessing Product Documentation and double-click the file you want to view.

Note You can view a .pdf file in Adobe Acrobat Viewer and .htm files in your web browser.

You can download a free copy of Acrobat Reader from the following Adobe website: http://www.adobe.com.

To copy the documentation from the CD to your system:

1. Insert the VisiBroker CD in the CD ROM drive.
2. Open the top-level docs directory.
3. Copy the file that you want onto your system.

Chapter

4

Configuring VisiBroker

This chapter describes how to configure your VisiBroker environment using command-line options and environment variables. It contains the following sections:

Overview

VisiBroker is designed to be easily configured with environment settings, regardless of whether you are developing applications, deploying those applications, or simply administering a VisiBroker system.

W When you install VisiBroker, the Windows registry will be updated with your configuration information. You may use the environment variables described in this chapter to override any registry settings that may have been made.

Many of the environment variables discussed in this chapter may be overridden through the use of command-line arguments, discussed in Chapter 7, “Deploying and executing applications.”

Basic environment settings

This section discusses the basic environment variables that must be set to run applications developed with VisiBroker, to use the developer and administration tools, and to execute the runtime support processes.

U Under UNIX, the necessary environment settings are added to the vbroker.sh or vbroker.csh shell script file which is generated by VisiBroker during the installation.

W Under Windows, the necessary environment settings are added to the Windows registry when you install VisiBroker. You may use the environment variables described in this chapter to override any registry settings that may have been made.

PATH

The PATH environment variable should be set to include the bin directory which contains the VisiBroker distribution. The bin directory is where the VisiBroker tools for developers, administrators, and users are located.

95 You can set the PATH environment variable with the following DOS command, substituting <installation_location> with the location where you installed VisiBroker for C++. Alternatively, you may want to set the PATH in your autoexec.bat file.

```
prompt> set PATH= <installation_location>\bin;%PATH%
```

NT Although the DOS set command can be used to set environment variables in Windows NT, you may find it easier to use the System control panel to automatically set the PATH. Open the System Control Panel, choose “PATH” as the variable to edit, and add the following definition to PATH, substituting <installation_location> with the location where you installed VisiBroker for C++:

```
<installation_location>\bin;
```

Any changes made to your environment variables with System Control Panel will not be reflected in currently running applications. All subsequently launched applications and DOS prompts should see the new settings.

U If you are using csh, you can update the PATH environment using the following command, substituting <installation_location> with the location where you installed VisiBroker for C++:

```
prompt> set path = (<installation location>/bin $path)
```

U If you are using the Bourne shell, you can update the PATH environment variable using the following commands, substituting <installation_location> with the location where you installed VisiBroker for C++:

```
prompt> PATH=$PATH:<installation_location>/bin
prompt> export PATH
```

VBROKER_ADM

The VBROKER_ADM environment variable defines the administration directory where important configuration information for VisiBroker's interface repository, object activation daemon, and Smart Agent are stored.

95 You can set your VBROKER_ADM environment variable with the following DOS command, substituting <installation_location> with the location where you installed VisiBroker for C++. Alternatively, you may want to set the VBROKER_ADM in your autoexec.bat file.

```
prompt> set VBROKER_ADM=<installaton_location>\adm
```

NT Although the DOS set command can be used to set environment variables in Windows NT, you may find it easier to use the System control panel to automatically set the PATH. Open the System Control Panel, choose "VBROKER_ADM" as the variable to edit, and add the following definition, substituting <installation_location> with the location where you installed VisiBroker for C++:

```
<installation_location>\adm
```

Any changes made to your environment variables with System Control Panel will not be reflected in currently running applications. All subsequently launched applications and DOS prompts should see the new settings.

The VBROKER_ADM environment variable is automatically set in the Windows registry when you install VisiBroker. You can change the registry settings by using the `vregedit` tool.

You can override the registry by setting the VBROKER_ADM environment variable. Assuming you want your own directory C:\my\adm to be used, you could set the VBROKER_ADM environment variable as follows:

```
prompt> set VBROKER_ADM= <installation_location>\adm
```

U If you are using csh, set the VBROKER_ADM environment variable as follows:

```
prompt> setenv VBROKER_ADM <installation location>/adm
```

U If you are using the Bourne shell, set the VBROKER_ADM environment variable as follows, substituting <installation_location> with the location where you installed VisiBroker for C++:

```
prompt> VBROKER_ADM=<installation_location>/adm
prompt> export VBROKER_ADM
```

Smart Agent environment variables

These following environment variables control communication with the Smart Agent (osagent).

W Under Windows, the necessary environment settings are added to the Windows registry when you install VisiBroker. You may use the environment variables described in this chapter to override any registry settings that may have been made.

OSAGENT_ADDR

A Smart Agent (osagent) is usually located through the use of a broadcast message, sent on behalf of a VisiBroker application. You may also explicitly specify the IP address or host name of the host that is executing the Smart Agent that you want your VisiBroker applications to use by setting the OSAGENT_ADDR environment variable.

You may override this environment variable by using the ORBagentAddr command-line argument, described on Chapter 7, "Deploying and executing applications."

Note If a Smart Agent cannot be contacted at the address you specify, broadcast messages will then be used to attempt to locate a Smart Agent.

W You can set the OSAGENT_ADDR with the following DOS command. Alternatively, you may want to set the OSAGENT_ADDR in your autoexec.bat file.

```
prompt> set OSAGENT_ADDR=199.99.99.1
```

U If you are using csh, you can set the OSAGENT_ADDR environment using the following command:

```
prompt> setenv OSAGENT_ADDR 199.99.99.1
```

U If you are using the Bourne shell and you have installed VisiBroker in /usr/local/vbroker, you can update the PATH environment variable using the following commands:

```
prompt> OSAGENT_ADDR=199.99.99.1
prompt> export OSAGENT_ADDR
```

OSAGENT_PORT

This environment variable defines the port number on which a Smart Agent (osagent) will listen for requests when it is started. When starting a VisiBroker application or tool, this environment variable specifies the port number that will be used for communicating with an osagent.

If you need to run multiple osagent processes on one host, you may assign each a different port number. See "Smart Agent configuration" on page 4–6 for more information on using this environment variable.

You may override this environment variable by using the -ORBagentPort command-line argument, described on Chapter 7, "Deploying and executing applications."

Note If the osagent_port is not set, a default port number of 14000 will be used.

W OSAGENT_PORT is automatically set in the Windows registry when you install VisiBroker. You can change the registry setting by using the vregedit tool or you may override the Windows registry by setting the OSAGENT_PORT environment variable. If you want the osagent to use port number 10000, you could set the OSAGENT_PORT environment variable as follows:

```
prompt> set OSAGENT_PORT=10000
```

U If you are using csh and you want the osagent to use port number 10000, set the OSAGENT_PORT environment variable as follows:

```
prompt> setenv OSAGENT_PORT 10000
```

U If you are using the Bourne shell and you want to use the port number 10000, set the OSAGENT_PORT environment variable as follows:

```
prompt> OSAGENT_PORT=10000
prompt> export OSAGENT_PORT
```

OSAGENT_ADDR_FILE

This environment variable specifies the name of a file that contains the IP addresses of other Smart Agents. These addresses represent Smart Agents executing on hosts located outside the local network with which the osagent is to communicate. For information on the use and the contents of this file, see "Connecting Smart Agents on different local networks" on page 5–4 of this guide.

W To set the OSAGENT_ADDR_FILE file on a Windows system, you could use the following command:

```
prompt> set OSAGENT_ADDR_FILE=C:\vbroker\admin\lcladdr.txt
```

U If you are using csh, set the OSAGENT_ADDR_FILE environment variable as follows:

```
prompt> setenv OSAGENT_ADDR_FILE /vbroker/admin/localaddr
```

U If you are using the Bourne shell, set the OSAGENT_ADDR_FILE environment variable as follows:

```
prompt> OSAGENT_ADDR_FILE /vbroker/admin/localaddr
prompt> export OSAGENT_ADDR_FILE
```

OSAGENT_LOCAL_FILE

This environment variable specifies the path to a file that contains network interface information for a Smart Agent (osagent) running on a multi-homed host. For information on the use and the contents of this file, see "Running the Smart Agent on multi-homed hosts" on page 5-4 of this guide.

W To set the OSAGENT_LOCAL_FILE file on a Windows system, you could use the following command:

```
prompt> set OSAGENT_LOCAL_FILE=C:\vbroker\admin\agntaddr.txt
```

U If you are using csh and you want the Smart Agent, set the OSAGENT_LOCAL_FILE environment variable as follows:

```
prompt> setenv OSAGENT_LOCAL_FILE /vbroker/admin/agentaddr
```

U If you are using the Bourne shell, set the OSAGENT_LOCAL_FILE environment variable as follows:

```
prompt> OSAGENT_ADDR_FILE /vbroker/admin/agentaddr
prompt> export OSAGENT_ADDR_FILE
```

Using ORB and BOA command-line arguments

The ORB and BOA command-line arguments allow you to override many environment variable settings and control advanced aspects of VisiBroker applications at the time you start your application. See Chapter 7, “Deploying and executing applications” for more details on using command-line arguments.

Configuring a deployment environment

When you are ready to deploy applications that have been developed with VisiBroker for C++, you must first create a deployment environment on the host where you plan to execute the application. Chapter 7, “Deploying and executing applications,” describes how create a deployment environment.

Smart Agent configuration

For complete information on how to configure the Smart Agent, see “Starting the Smart Agent” on page 5–2.

Chapter

5

Starting runtime support services

This chapter describes the runtime support services provided with VisiBroker for C++, and includes the following sections:

Command usage

The use of the VisiBroker runtime support services described in this chapter differ, depending on whether you have a UNIX or a Windows environment.

U On a UNIX system, you may view command-line arguments accepted by a service by entering the command name as follows:

```
osfind -\?
```

W On a Windows system, you would enter:

```
osfind -?
```

Passing command-line arguments

VisiBroker for C++ 3.0 has modified the syntax for command-line arguments to allow for more consistent parsing. In the past, arguments were passed using the following syntax:

```
parameter=value
```

This has changed with release 3.0 to the following syntax:

```
parameter<space>value
```

The new syntax is preferred, but passing parameters with the equals sign has been maintained for compatibility. The following command lines are equivalent in the 3.0 release.

Code Sample 5.1 2.0 argument syntax

```
server -ORBagentaddr=201.34.53.83 -OAport=12000
```

Code Sample 5.2 New argument syntax

```
server -ORBagentaddr 201.34.53.83 -OAport 12000
```

Starting the Smart Agent

The `osagent` command starts the VisiBroker Smart Agent, which provides ORB object location and failure detection services. You must run at least one Smart Agent on a host in your network. You can automate the start-up of an osagent by adding it to your platform's start-up file.

The osagent command has the following syntax:

Syntax

```
osagent [options]
```

The oasgent command accepts the following command-line arguments:

Table 5.1 oasgent command-line arguments

Option	Description
-C	Allows the osagent to run in console mode if it has been installed as NT service.
-p *UDP_port*	Overrides the setting of OSAGENT_PORT and the registry setting.
-verbose	Turns verbose mode on, which provides information and diagnostic messages during execution.

The following example of the osagent command starts the agent in verbose mode and listens on UDP port 11000 (instead of the default 14000).

```
osagent -v -p 11000
```

Verbose output

U On a UNIX platform, the verbose output is sent to stdout.

W On a Windows system, the verbose output is written to a log file stored at <installation_location>/log/osagent.log or to the directory specified by the VBROKER_ADM environment variable.

Environment variables

The following list of environment variables may be set for this command. For information on these environment variables, see Chapter 4, "Configuring VisiBroker."

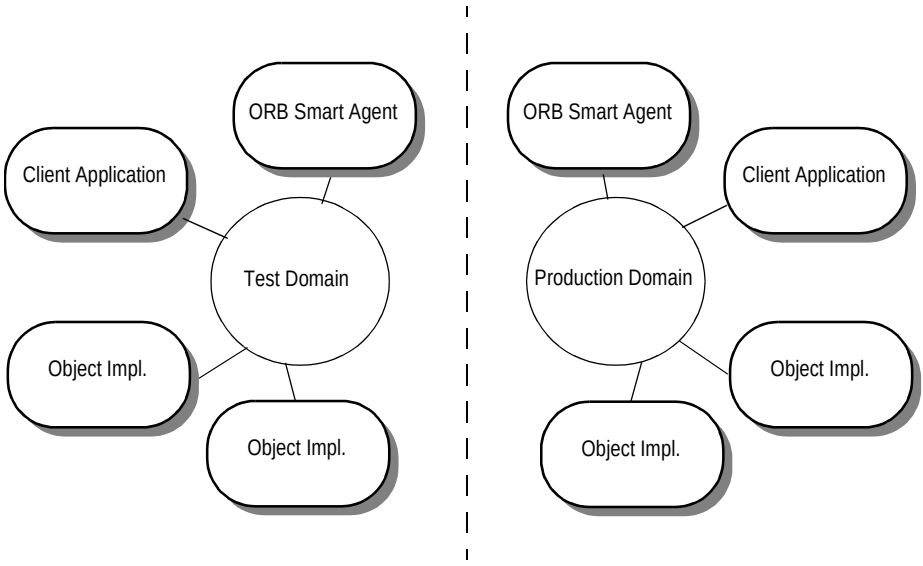
- OSAGENT_ADDR_FILE
- OSAGENT_PORT
- VBROKER_ADM

Note For more information about how environment variables and runtime properties affect the osagent, refer to "The Smart Agent," in the *VisiBroker for C++ Programmer's Guide*.

Configuring ORB domains

It is often desirable to have two or more separate ORB domains running at the same time. One domain might consist of the production versions of client applications and object implementations while another domain might be made up of test versions of the same clients and objects that have not yet been released for general use.

Figure 5.1 Separate ORB domains

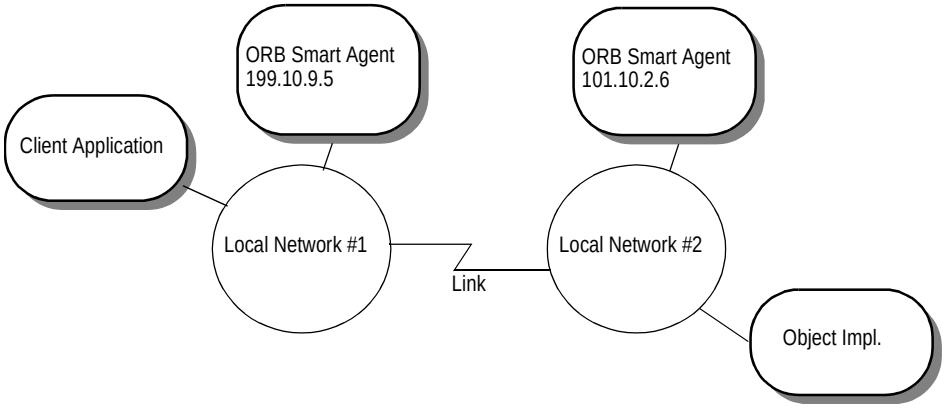


VisiBroker allows you to distinguish between two or more ORB domains on the same network by using a unique UDP port number for the osagents for each domain. The default setting for the osagent port number is 14000. During development, it is useful for each developer to have his or her own port number. The environment variable OSAGENT_PORT must be set on each host running an osagent, an object implementation, or a client application assigned to that ORB domain.

Connecting Smart Agents on different local networks

If you start multiple Smart Agents on your local network, they will discover each other by using UDP broadcast messages. Your local networks are configured by specifying the scope of broadcast messages using the IP subnet mask. Figure 5.2 shows two local networks, connected by a network link.

Figure 5.2 Two osagent processes and their IP addresses, located on separate, connected local networks



To allow the Smart Agent on one network to contact an Smart Agent on another local network, you must make the host name or IP address of the remote Smart Agent available in a file using the IP address. Code Sample 5.3 shows what this file would need to contain to allow the Smart Agent on local network #1 to connect to the Smart Agent on the network #2.

Code Sample 5.3 Content of the agentaddr file for the osagent on network #1

```
101.10.2.6
```

The name of this file may be specified by setting the `OSAGENT_ADDR_FILE` environment variable. If the OSAGENT_ADDR_FILE environment variable is not set, the file must be named agentaddr and must be located in the directory designated by the VBROKER_ADM environment variable.

Running the Smart Agent on multi-homed hosts

Running the osagent on a host with multiple network interfaces (known as a multi-homed host) provides you with a powerful mechanism for bridging objects from separate sub-networks. Since the osagent on a multi-homed host is able to use several network interfaces, clients on one interface's sub-network can use the osagent to locate a server on a different sub-network without having to use multiple osagents and the agentaddr file.

By default, the Smart Agent attempts to bind to all the network interfaces it can find on the local host. If the host has multiple interfaces, the Smart Agent will listen and broadcast on all interfaces that support point-to-point connections or broadcast messages. Each network interface to which the Smart Agent binds successfully is

listed at the beginning of the verbose output (generated by starting `osagent` with the -v option).

For example, on a multi-homed host, `osagent` may start by showing:

```
Bound to the following interfaces:
Address: 216.64.15.10 Subnet: 255.255.255.0 Broadcast: 216.64.15.255
Address: 214.79.98.88 Subnet: 255.255.255.0 Broadcast: 214.79.98.255
. . .
```

This shows the address, subnet mask, and broadcast address for each interface in the machine. This information should match the results from the UNIX command:

```
prompt> ifconfig -a
```

OSAGENT_LOCAL_FILE environment variable

You can specify multiple network interface information in the localaddr file. You also need to set the OSAGENT_LOCAL_FILE environment variable to the full path of this file. The localaddr file should contain a list of the network interfaces that the osagent is to use. Each line should contain the IP address, subnet mask, and broadcast address for a single interface. For example, the following file contains information for two network interfaces:

```
216.64.15.10 255.255.255.0 216.64.15.255
214.79.98.88 255.255.255.0 214.79.98.255
```

U Though the Smart Agent will automatically configure itself to use all available interfaces on a multi-homed host running UNIX, you can use the localaddr file to restrict the network interfaces that the Smart Agent will use. You can display all the available network interface IP address values for your UNIX host by using the following command:

```
prompt> ifconfig -a
```

W You must use the localaddr file with multi-homed hosts running Windows because the Smart Agent is not able to dynamically determine the subnet mask and broadcast address for network interfaces. You can obtain the appropriate network interface values for the localaddr file by accessing the TCP/IP protocol properties from the Network Control Panel. If your host is running Windows NT, the ipconfig command will provide the needed values.

If you do not use the localaddr file with a multi-homed host running Windows, the Smart Agent will use default values which may not be correct for your environment.

Starting the Object Activation Daemon

Object implementations that are registered with VisiBroker's Object Activation Daemon (OAD) are automatically activated when a client requests a bind to the object. Object implementations can be registered programmatically, using the ORB interface to the OAD, or manually, using the oadutil reg command-line interface.

Once an object is registered with the OAD, the object becomes visible to the Smart Agent. When a client application binds to such an object, the Smart Agent and the

OAD cooperate to activate the requested object. This activation process is transparent to the application program.

Note You must execute `oad` before you can use any of the `oadutil` commands.

The `oad` command starts the Object Activation service and has the following syntax:

Syntax

```
oad [options]
```

Note Objects with a local scope are not visible to the Smart Agent or the OAD.

The `oad` command accepts the following command-line arguments:

Table 5.2 oad command-line arguments

Option	Description
-v	Turns on verbose mode.
-f	Stipulates that the process should not fail if another OAD is running on this host.
-t <# of seconds>	Specifies the amount of time the OAD will wait for a spawned server process to activate the requested ORB object. The default time-out is 20 seconds. Set this value to 0 if you wish to wait indefinitely. If a spawned server process does not activate the requested object within the time-out interval, the OAD will kill the spawned process and the client will see a CORBA::NO IMPLEMENT exception. Turn on the verbose option to see more detailed information.
NT -C	Console mode. Allows the osagent to run in console mode if it has been installed as NT service.
-k	Stipulates that an object's child process should be killed once all of its object are unregistered with the OAD.
-?	Displays command usage.

Environment variables

Table 5.3 Environment variables

Environment variable	Description
VBROKER_ADM	Defines the path to the directory where the files in the implementation repository are stored. The default subdirectory is called <VBROKER_ADM>/`impl_dir`. The default may be changed using the VBROKER_IMPL_PATH environment variable.
VBROKER_IMPL_PATH	Specifies the platform-specific directory for storing the implementation repository. If not specified, the default is <VBROKER_ADM>/`impl_dir`.
VBROKER_IMPL_NAME	Specifies the name of the implementation repository. If not specified, the default is `impl_rep`. This environment setting can be overridden using the -filename command-line option.

Starting an interface repository

The irep command is used to start an interface repository server, which manages a database containing detailed descriptions of IDL interfaces. By default a Smart Agent is usually running on a host in your network. If you do not want to run a Smart Agent, you need to set the flag to turn it off or the irep command will not execute. For information on starting a Smart Agent, refer to "Starting the Smart Agent" on 5-2 of this guide.

The interface repository's database includes interface names, inheritance structure, and supported methods and arguments. The interface repository can consist of multiple interface specifications which can be populated using the idl2ir command (refer to the *VisiBroker for C++ Programmer's Guide* for information on using the idl2ir command).

The irep command has the following syntax:

Syntax

```
irep [options] {irep_name} [idl_storage_filename]
```

The irep command accepts the following command-line arguments:

Table 5.4 irep command-line arguments

Option	Description
-console	Runs irep in command-line mode. Otherwise, it is presented as a GUI, as shown in Figure 5.3.
-version	Prints the version of this tool.
irep_name	Specifies the instance name of the interface repository server to load.
idl_storage_filename	Specifies the name of the IDL file to be loaded into the interface repository.

When you use the irep -console command-line option, a text interface with the following commands becomes available:

Table 5.5 irep -console commands

Options	Action
h	(help) Displays the commands recognized by the program.
l [name]	(lookup) Displays the named IR element or all elements if no name is supplied. The name can be a fully qualified IDL name, or a Repository ID.
s	(save) Updates the repository database specified when irep was started, or the file created earlier with an a command.
a {filename}	(save as) Writes the contents of the interface repository to a new repository database file.
r {filename}	(read) Reads the named IDL file, adding its elements to the repository database. If any element in the IDL file matches an element that is already in the repository database, the program rejects the file. Matching is based on fully qualified IDL name or Repository ID.
q	(quit) Exits the program.

The following is an example of how to use the `irep` command:

Example

```
irep -console my_server
```

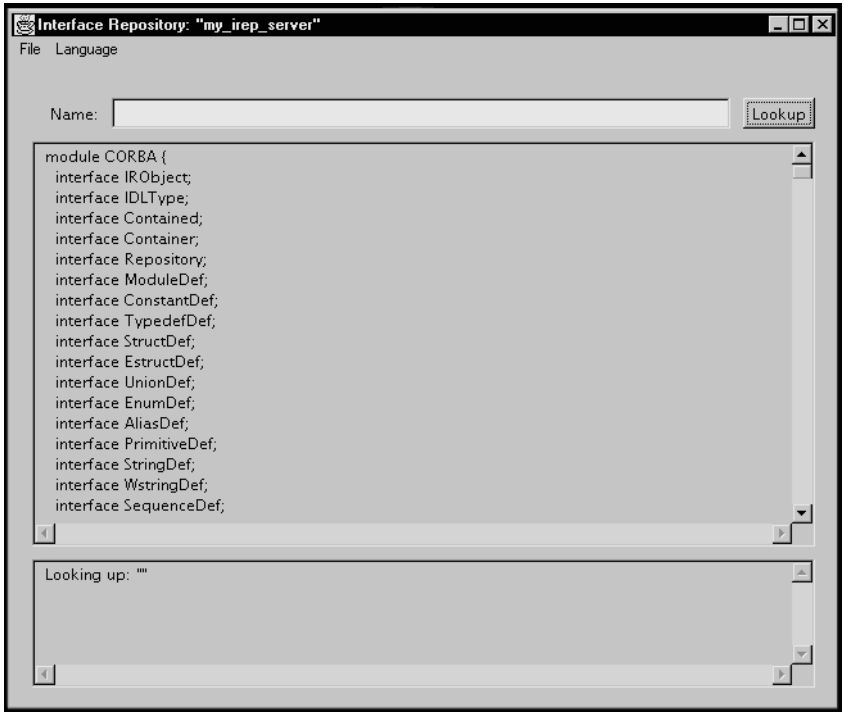
Note For further information on Interface Repositories, refer to the *VisiBroker for C++ Programmer's Guide*.

Using the GUI

The `irep` command's graphical user interface allows you to lookup information about an object by entering its interface name in the field marked "Name" and then clicking on the "Lookup" button.

The results of the search will appear in the scrollable list in the middle of the window. Status messages, if any, will appear in the scrollable list at the bottom of the window.

Figure 5.3 Interface Repository user interface



Chapter

6

Using Administration tools

This chapter describes the Administration tools provided with VisiBroker for C++, and includes the following sections:

Command usage

The use of the VisiBroker administration tools described in this chapter differ, depending on whether you have a UNIX or a Windows environment.

You may view command-line arguments accepted by a service by entering the command name as follows:

```
oadutil list -help
```

Passing command-line arguments

VisiBroker for C++ 3.0 has modified the syntax for command-line arguments to allow for more consistent parsing. In the past, arguments were passed using the following syntax:

```
parameter=value
```

This has changed with release 3.0 to the following syntax:

```
parameter<space>value
```

The new syntax is preferred, but passing parameters with the equals sign has been maintained for compatibility. The following command lines are equivalent in the 3.0 release.

Code Sample 6.1 2.0 argument syntax

```
server -ORBagentaddr=201.34.53.83 -OAport=12000
```

Code Sample 6.2 New argument syntax

```
server -ORBagentaddr 201.34.53.83 -OAport 12000
```

Using the Object Activation Daemon utilities

The oadutil commands provides a way for you to manually register, unregister, and list the object implementations available on your VisiBroker system. The oadutil commands are implemented in C++ and use a command-line interface. Each command is accessed by invoking the oadutil command, passing the type of operation to be performed as the first argument.

Note An oad process must be started on at least one host in your network before you can use the oadutil commands.

The `oadutil` command has the following syntax:

Syntax

```
oadutil {list|reg|unreg} [options]
```

The options for this tool vary, depending on whether you specify list, reg or unreg.

Converting interface names to repository IDs

When registering or unregistering an object with the OAD, the oadutil commands allow you to specify either an object's IDL interface name or its repository id.

An interface name is converted to a repository ID as follows:

1. Prepend "IDL:" to the interface name.
2. Replace all non-leading instances of "::" with a slash ("/") character.
3. Append ":1.0" to the interface name.

   For example, the IDL interface name:

   ```
   ::Module1::Module2::IntfName
   ```

   would be converted to the following repository ID:

   ```
   IDL:Module1/Module2/IntfName:1.0
   ```

The #pragma id and #pragma prefix mechanisms can be used to override the default generation of repository id's from interface names. If the #pragma id mechanism is used in user-defined IDL files to specify non-standard repository IDs, the conversion

process outlined above will not work. In these cases, you must use -r repository id argument and specify the object's repository ID.

To obtain the repository id of the object implementation's most derived interface, use the method `_repository_id ()` defined on all CORBA objects.

Listing objects with oadutil list

The `oadutil list` command returns all ORB object implementations registered with the Object Activation Daemon. Each OAD has its own implementation repository database where the registration information is stored.

Note An `oad` process must be started on at least one host in your network before you can use the oadutil list command.

The `oadutil list` command has the following syntax:

Syntax

```
oadutil list [options]
```

The `oadutil list` command accepts the following command-line arguments:

Table 6.1 oadutil list command-line arguments

Option	Description
-i <interface name>	Lists the implementation information for objects of a particular IDL interface name. Only one of the following options may be specified at a particular time: -i, -r, or -s.
	Note: All communications with the ORB and BOA reference an object's repository id instead of the interface name. For more information about the conversion performed when specifying an interface name, see "Converting interface names to repository IDs" on page 6-2.
-r <repository id>	Lists the implementation information of a specific repository id. See "Converting interface names to repository IDs" on page 6-2 for details on specifying repository IDs. Only one of the following options may be specified at a particular time: -i, -r, or -s.
-s <service name>	Lists the implementation information for a specific service name. Only one of the following options may be specified at a particular time: -i, -r, or -s.
-o <object name>	Lists the implementation information for a specific object name. You can use this only if the interface or repository id is specified in the command statement. This option is not applicable when the -s argument is used.
-h <OAD host name>	Lists the implementation information for objects registered with an OAD running on a specific remote host.
-verbose	Turns verbose mode on, causing messages to be output to stdout.
-version	Prints the version of this tool.
-full	Lists the status of all implementations currently spawned by the OAD.

Description

The `oadutil list` utility allows you to list all ORB object implementations registered with the Object Activation Daemon. The information for each object includes:

- Interface names of the ORB objects.
- Instance names of the object offered by that implementation.
- Full pathname of the server implementation's executable.
- Activation policy of the ORB object (shared, unshared, or per-method).
- Reference data specified when the implementation was registered with the OAD.
- List of arguments to be passed to the server at activation time.
- List of environment variables to be passed to the server at activation time.

The following is an example of a local list request, specifying an interface name and object name:

Example

```
oadutil list -i Bank::AccountManager -o VisigenicBank
```

The following is an example of a remote list request, specifying a host IP address:

Example

```
oadutil list -h 206.64.15.198
```

Registering objects with oadutil reg

The `oadutil reg` command allows you to register one or more ORB object implementation with the Object Activation Daemon. Registered implementations will be activated automatically by the OAD when a client requests a bind to the object. Once an object implementation has been registered with the OAD, the VisiBroker Smart Agent is notified that the object is available through the OAD. The object's implementation is also added to the OAD's implementation repository.

Note An `oad` process must be started on at least one host in your network before you can use the oadutil reg command.

If an object implementation is started manually as a persistent server, it does not need to be registered with the OAD. Objects with a local scope are not visible to the osagent or the OAD.

The `oadutil reg` command has the following syntax:

Syntax

```
oadutil reg [options]
```

Note If an object implementation is started manually as a persistent server, it does not need to be registered with the OAD.

The options for the `oadutil reg` command accepts the following command-line arguments:

Table 6.2 oadutil reg command-line arguments

Option	Description
Required	
-i <interface name>	Specifies a particular IDL interface name. Only one of the following options may be specified at a particular time: -i, -r, or -s. **Note:** See "Converting interface names to repository IDs" on page 6-2 for details on specifying repository IDs.
-r <repository id>	Specifies a particular repository id. Only one of the following options may be specified at a particular time: -i, -r, or -s.
-s <service name>	Specifies a particular service name. Only one of the following options may be specified at a particular time: -i, -r, or -s.
-o <object name>	Specifies a particular object. You can use this only if the interface name or repository id is specified in the command statement. This option is not applicable when the -s argument is used.
-cpp <file name to execute>	Specifies the full path of an executable file that must create and register an object that matches the -o/-r/-s arguments. Applications registered with the -cpp argument must be stand-alone executables.
-java <full class name>	Specifies the full name of a java class containing a main routine. This application must create and register an Object that matches the -o/-r/-s argument. Classes registered with the -C++ argument will be executed with the command: vbj <full_classname>.
Optional	
-host <OAD host name>	Specifies a specific remote host where the OAD is running.
-verbose	Turns verbose mode on, causing messages to be output to stdout.
-version	Prints the version of this tool.
-d <referenceData>	Specifies reference data to be passed to the server upon activation.
-a arg1 -a arg2	Specifies the arguments to be passed to the spawned executable as command-line arguments. Arguments can be passed with multiple -a (arg) parameters. They will be propagated in order to create the spawned executable.
-e env1 -e env2	Specifies environment variables to be passed to the spawned executable. Arguments can be passed with multiple -e (env) parameters. They will be propagated in order to create the spawned executable.
-p {shared | unshared | per-method}	Specifies the activation policy of the spawned objects. The default policy is SHARED_SERVER. shared—Multiple clients of a given object share the same implementation. Only one client is activated by an OAD at a particular time. unshared—Only one client of a given implementation will bind to the activated server. If multiple clients wish to bind to the same object implementation, a separate server is activated for each client application. A server exits when its client application disconnects or exits. per-method—Each invocation of a method results in a new sever being activated. The server exits upon completion of the method invocation.

Examples

Performing a local registration by specifying repository ID

The following command will register with the OAD the VisiBroker for C++ program factory_r found in the /home/developer/project1 directory. It will be activated upon request for objects of repository ID IDL:ehTest/Factory:1.0 (which corresponds to the interface name ehTest::Factory). The instance name of the object to be activated is ReentrantServer, and that name is also passed to the spawned executable as a command-line argument. This server has the unshared policy, by which it will be terminated when the requesting client breaks its connection to the spawned server.

Example

```
prompt> oadutil reg -r IDL:ehTest/Factory:1.0 -o ReentrantServer \
-cpp /home/developer/Project1/factory_r -a ReentrantServer \
-p unshared
```

Performing a local registration by specifying IDL interface name

The following command will register with the OAD, the VisiBroker for C++ class Server. The specified C++ class must be found in the OAD's CLASSPATH. In this example, the specified C++ class must activate an object of repository ID IDL:Bank/AccountManager:1.0 (corresponding to the interface name IDL name Bank::AccountManager) and instance name CreditUnion. The sever will be started with unshared policy, ensuring that it will terminate when the requesting client breaks its connection.

Performing a remote registration to an OAD on Windows NT

To register an implementation with an OAD on a remote host, use the -h argument to oadutil reg.

The following is an example of how to perform a remote registration to an OAD on Windows NT from a UNIX shell. The double backslashes are necessary to avoid having the shell interpret the backslashes before passing them to oadutil.

Example

```
prompt> oadutil reg -r IDL:Library:1.0 Harvard \
-cpp c:\\vbroker\\examples\\library\\libsrv.exe -p shared -h 100.64.15.198
```

Unregistering objects using oadutil unreg

The `oadutil unreg` command allows you to unregister one or more object implementations registered with the Object Activation Daemon. Once an object is unregistered, it can no longer be automatically activated by the OAD if a client requests the object. Only objects that have been previously registered via the `oadutil reg` command may be unregistered with `oadutil unreg`.

If you specify only an interface name, all ORB objects associated with that interface will be unregistered. Alternatively, you may identify a specific ORB object by its interface name and object name. When you unregister an object, all processes associated with that object will be terminated.

Note An `oad` process must be started on at least one host in your network before you can use the oadutil reg command.

The `oadutil unreg` command has the following syntax:

Syntax

```
oadutil unreg [options]
```

The options for the `oadutil unreg` command accepts the following command-line arguments:

Table 6.3 oadutil unreg command-line arguments

Option	Description
Required	
-i <interface name>	Specifies a particular IDL interface name. Only one of the following options may be specified at a particular time: -i, -r, or -s. **Note:** See "Converting interface names to repository IDs" on page 6-2 for details on specifying repository IDs.
-r <repository id>	Specifies a particular repository id. Only one of the following options may be specified at a particular time: -i, -r, or -s.
-s <service name>	Specifies a particular service name. This option is not applicable if the -s argument is used.
-o <object name>	Specifies a particular object name. You can use this only if the interface name or repository id is included in the command statement. Only one of the following options may be specified at a particular time: -i, -r, or -s.
Optional	
-host <host name>	Specifies the host name where the OAD is running.
-verbose	Enables verbose mode, causing messages to be output to stdout.
-version	Prints the version of this tool.

Example The `oadutil unreg` utility unregisters one or more ORB objects from these three locations:

- The Object Activation Daemon
- The implementation repository
- The Smart Agent

The following is an example of how to use the oadutil unreg command. It unregisters the implementation of the Bank::AccountManager named InpriseBank from the local OAD.

Example

```
oadutil unreg -i Bank::AccountManager -o InpriseBank
```

Reporting all objects and services with osfind

The `osfind` command reports on all VisiBroker related objects and services which are currently available on a given network.

You can use osfind to determine the number of Smart Agent processes running on the network and the exact host on which they are executing. The `osfind` command also reports on all VisiBroker objects that are active on the network. You can use

osfind to monitor the status of the network and locate stray objects during the debugging phase.

The `osfind` command has the following syntax:

Syntax

```
osfind [options]
```

The following options are valid with osfind. If none of the -r/-a/-o options are specified, osfind lists all of the agents, OADs, and implementations in your domain.

Table 6.4 osfind options

Option	Description
-b	Specifies the VisiBroker 2.0 backward-compatible osfind mechanism. **Note:** 3.0 activated objects will be listed as manually started when this flag is set.
-r <repository ID>	Lists all implementations of the given repository ID. **Note:** You can use regular expressions in repository IDs. Example -r "^.*" will list all interfaces. See "Converting interface names to repository IDs" on page 6-2 for details on specifying repository IDs.
-a	Lists all Smart Agents in your domain.
-o	Lists all Object Activation Daemons in your domain.
-LOCdebug <0 or 1>	If you enter a one, this turns on debugging. The default is debugging off. If you've turned on debugging, you can turn it off by entering a zero; for example: -LOCdebug 0.
-LOCverify <0 or 1>	If you enter a zero, this turns off verification. Verification is normally turned on--the Smart Agent does an immediate update and verifies that existence of the object before it returns an object reference to the Location Service. If you turn this feature on it will be slower but will return the most accurate information. To turn verification on, enter a one; for example: -LOCverify 1.

Chapter

7

Deploying and executing applications

This chapter describes how to deploy and run client programs and server objects that have been developed with VisiBroker for C++. It provides information on creating a deployment environment, using command-line arguments, and starting and stopping applications This chapter contains the following topics:

Deploying VisiBroker applications

To deploy applications developed with VisiBroker for C++, you must first set up a run-time environment on the host where the application is to be executed and ensure that the necessary support services are available on the local network.

The runtime environment required for applications developed with VisiBroker for C++ includes these components:

- The VisiBroker C++ libraries, located in the bin sub-directory where the product is installed.
- The availability of the support services required by the application.

VisiBroker ORB libraries

The VisiBroker ORB libraries must be installed on the host where the deployed application is to execute. The location of these libraries must be included in the PATH for the application's environment.

Configuring the environment

If the deployed application is to use a Smart Agent (osagent) on a particular host, you must set the OSAGENT_ADDR environment variable, as described in Chapter 4, "Configuring VisiBroker," before running the application. You can use the ORBagentAddr command-line argument to specify a hostname or IP address, as described on page 7-3.

If the deployed application is to use a particular UDP port when communicating with a Smart Agent (osagent), you must set the OSAGENT_PORT environment variable, as described in Chapter 4, "Configuring VisiBroker" before running the application. You can use the ORBagentPort command-line argument to specify the IP port number, as described in Chapter 4.

Support service availability

A Smart Agent (osagent) must be executing somewhere on the network where the deployed application is to be executed. Depending on the requirements of the application being deployed, you may need to ensure that other VisiBroker runtime support services are available, as well. These services include:

Support services	Needed when
Object Activation Daemon (oad)	A deployed application is a server that implements object which needs to be started on demand.
Interface Repository (irep)	A deployed application uses either the dynamic skeleton interface or dynamic implementation interface. See the *VisiBroker for C++ Programmer's Guide* for a description of these interfaces.

See Chapter 5, "Starting runtime support services" for more details.

Running the application

Now that you have compiled your client program and server implementation, you are ready to run your first VisiBroker application.

Starting the Smart Agent

Before you attempt to run VisiBroker client programs or server implementations, you must first start the Smart Agent on at least one host in your local network. The Smart Agent is described in detail in the *VisiBroker for C++ Programmer's Guide.*

Executing client applications

A client application is one that uses ORB objects, but does not offer any ORB objects of its own to other client applications.

Command-line arguments for clients

The following table summarizes the command-line arguments that may be specified for a client application. These arguments also are applicable to servers.

Table 7.1 Command-line arguments for client applications

Options	Description
`-ORBagentAddr <hostname | ip_address>`	Specifies the hostname or IP address of the host running the Smart Agent this client should use. If a Smart Agent is not found at the specified address or if this option is not specified, broadcast messages will be used to locate a Smart Agent.
`-ORBagentPort <port_number>`	Specifies the port number of the Smart Agent. This option can be useful if multiple ORB domains are required, as described in "Configuring ORB domains" on page 5–3. If not specified, a default port number of 14000 will be used.
`-ORBbackcompat <0 | 1>`	If set to 1, this option specifies that backward compatibility with VisiBroker 2.0 should be provided. The default is 0. See the *VisiBroker for C++ Programmer's Guide* for details on what happens when this option is set to 1.
`-ORBbackdii <0 | 1>`	If set to 1, this option specifies that support for the 1.0 IDL-to-C++ mapping should be provided. If set to 0 or not specified at all, the new 1.1 mapping will be used. The default setting is 0. If -`ORBbackcompat` is set to 1, this option will automatically be set to 1.
`-ORBir_name <ir_name>`	Specifies the name of the Interface Repository to be accessed when the `Object::get_interface()` method is invoked on object implementations.
`-ORBir_ior <ior_string>`	Specifies the IOR of the Interface Repository to be accessed when the `Object::get_interface()` method is invoked on object implementations.

Table 7.1 Command-line arguments for client applications (continued)

Options	Description
-ORBnullstring <0 | 1>	If set to 1, this option specifies that the ORB will allow C++ NULL strings to be streamed. The NULL strings will be marshalled as strings of length 0—as opposed to the empty string ("") which is marshalled as a string of length 1, with the sole character of "\0". If set to 0, attempts to marshal out a NULL string will throw CORBA::BAD_PARAM. Attempts to marshal in a NULL string will throw CORBA::MARSHAL. The default setting is 0. If -ORBbackcompat is set to 1, this option will automatically be set to 1.
-ORBrcvbufsize <buffer_size>	Specifies the size of the TCP buffer (in bytes) used to receive responses. If not specified, a default buffer size will be used. This argument can be used to significantly impact performance or benchmark results.
-ORBsendbufsize <buffer_size>	Specifies the size of the TCP buffer (in bytes) used to send client requests. If not specified, a default buffer size will be used. This argument can be used to significantly impact performance or benchmark results.
-ORBshmsize <size>	Specifies the size of the send and receive segments (in bytes) in shared memory. If your client program and object implementation communicate via shared memory, you may use this option to enhance performance. This option is only supported on Windows platforms.
-ORBtcpnodelay <0 | 1>	When set to 1, it sets all sockets to immediately send requests. The default value of 0 allows sockets to send requests in batches as buffers fill. This argument can be used to significantly impact performance or benchmark results.

Executing server applications

A server application is one that offers one or more ORB objects to client applications. A server application may be started manually with the **vbj** command, or it may be activated dynamically by the Object Activation Daemon (oad).

Command-line arguments for servers

The following table summarizes the command-line arguments that may be specified for a server application.

Table 7.2 Command-line arguments for server applications

Options	Description
`-OAipAddr <hostname | ip_address>`	Specifies the hostname or IP address to be used for the Object Adaptor. Use this option if your host has multiple network interfaces and the BOA is associated with only one of those interfaces. If no option is specified, the host's default address is used.
`-OAport <port_number>`	Specifies the port number to be used by the object adapter when listening for a new connection.
`-OAsendbufsize <buffer_size>`	Specifies the size of the buffer, in bytes, used to send messages. If not specified, a default value will be used.
`-OArcvbufsize <buffer_size>`	Specifies the size of the buffer, in bytes, used to receive messages. If not specified, a default value will be used.
`-OAid <TPool | TSession>`	Specifies the thread policy for multithread servers, to be used by the BOA. The default is TPool unless you are in backward compatibility mode; if you are in backward compatibility, the default is TSession.
`-OADThreadStackSize <stack_size>`	Specifies the maximum thread buffer size in bytes when `-OAD TPool` is selected.
`-OAthreadMax <#>`	Specifies the maximum number of threads allowed when `-OAid TPool` is selected. If you do not specify or you specify 0, this selects unlimited number of threads or, to be more precise, a number of threads limited only by your system resources.
`-OAthreadMin <#>`	Specifies the minimum number of threads available in the thread pool. If you do not specify, the default is zero. You can specify this only when `-OAid TPool` is selected.
`-OAthreadMaxIdle`	Specifies the time which a thread can exist without servicing any requests. Threads that idle beyond the time specified can be returned to the system. By default, this is set to 300 seconds.
`-OAconnectionMax <#>`	Specifies the maximum number of connections allowed when `-OAid TSession` is selected. If you do not specify, the default is unlimited.
`-OAconnectionMaxIdle`	Specifies the time, in seconds, which a connection can idle without any traffic. Connections that idle beyond this time can be shutdown by VisiBroker. By default, this is set to 0 which means that connections will never automatically time-out. This option should be set for Internet applications.

Appendix

A

Error and log files

This chapter describes the log and error files used by VisiBroker for C++, and contains the following section:

Logging output

Many VisiBroker tools offer a verbose mode that displays information about the tool as it executes. In addition, any application that is linked with the VisiBroker library may also produce output. On UNIX systems, this output is written to the console. On Windows systems, this output is written to one of several log files.

W Table A.1 summarizes the names of the various log files that may be produced on Windows.

Table A.1 Summary of log file names

File name	Description
`oad.log`	Produced by the Object Activation Daemon when started with the `-v flag`.
`osagent.log`	Produced by the Smart Agent when started with the `-v flag`.
`visout.log`	Contains any output to `cout` that is produced by a client or server.
`vislog.log`	Contains any output to `clog` that is produced by a client or server.
`viserr.log`	Contains any output to `cerr` that is produced by a client or server.

The location of these files is determined by the following rules:

1 An attempt will be made to write the file to the log directory within the directory pointed to by the `VBROKER_ADM` environment variable. If the application or tool does not have write permission for this directory, go to step 2.

2. An attempt will then be made to write the file to the directory /vbroker/log on the drive from which the application or tool was started. If this fails, go to step 3.
3. An attempt will then be made to write the file to the current directory.

Index

U

V

W