



Visual Cafe™ Enterprise Suite

Distributed Development Guide

Symantec Visual Cafe™ Enterprise Suite Distributed Development Guide

The software described in this book is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Copyright Notice

Copyright © 1999 Symantec Corporation.

All Rights Reserved.

Released: 3/99 for Visual Cafe Enterprise Suite Version 3

This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent in writing from Symantec Corporation, 10201 Torre Avenue, Cupertino, CA 95014.

ALL EXAMPLES WITH NAMES, COMPANY NAMES, OR COMPANIES THAT APPEAR IN THIS MANUAL ARE IMAGINARY AND DO NOT REFER TO, OR PORTRAY, IN NAME OR SUBSTANCE, ANY ACTUAL NAMES, COMPANIES, ENTITIES, OR INSTITUTIONS. ANY RESEMBLANCE TO ANY REAL PERSON, COMPANY, ENTITY, OR INSTITUTION IS PURELY COINCIDENTAL.

Every effort has been made to ensure the accuracy of this manual. However, Symantec makes no warranties with respect to this documentation and disclaims any implied warranties of merchantability and fitness for a particular purpose. Symantec shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples herein. The information in this document is subject to change without notice.

Trademarks

Symantec Visual Cafe, Symantec, and the Symantec logo are U.S. registered trademarks of Symantec Corporation.

Other product names mentioned in this manual may be trademarks or registered trademarks of their respective companies and are the sole property of their respective manufacturers.

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

SYMANTEC LICENSE AND WARRANTY

The software which accompanies this license (the "Software") is the property of Symantec or its licensors and is protected by copyright law. While Symantec continues to own the Software, you will have certain rights to use the Software after your acceptance of this license. Except as may be modified by a license addendum which accompanies this license, your rights and obligations with respect to the use of this Software are as follows:

- You may:
 - (i) use one copy of the Software on a single computer;
 - (ii) make one copy of the Software for archival purposes, or copy the software onto the hard disk of your computer and retain the original for archival purposes;
 - (iii) use the Software on a network, provided that you have a licensed copy of the Software for each computer that can access the Software over that network;
 - (iv) after written notice to Symantec, transfer the Software on a permanent basis to another person or entity, provided that you retain no copies of the Software and the transferee agrees to the terms of this agreement; and
 - (v) if a single person uses the computer on which the Software is installed at least 80% of the time, then after returning the completed product registration card which accompanies the Software, that person may also use the Software on a single home computer.
- You may not:
 - (i) copy the documentation which accompanies the Software;
 - (ii) sublicense, rent or lease any portion of the Software;
 - (iii) reverse engineer, decompile, disassemble, modify, translate, make any attempt to discover the source code of the Software, or create derivative works from the Software; or
 - (iv) use a previous version or copy of the Software after you have received a disk replacement set or an upgraded version as a replacement of the prior version, unless you donate a previous version of an upgraded version to a charity of your choice, and such charity agrees in writing that it will be the sole end user of the product, and that it will abide by the terms of this agreement. Unless you so donate a previous version of an upgraded version, upon upgrading the Software, all copies of the prior version must be destroyed.

- Sixty Day Money Back Guarantee:

If you are the original licensee of this copy of the Software and are dissatisfied with it for any reason, you may return the complete product, together with your receipt, to Symantec or an authorized dealer, postage prepaid, for a full refund at any time during the sixty day period following the delivery to you of the Software.

- Limited Warranty:

Symantec warrants that the media on which the Software is distributed will be free from defects for a period of sixty (60) days from the date of delivery of the Software to you. Your sole remedy in the event of a breach of this warranty will be that Symantec will, at its option, replace any defective media returned to Symantec within the warranty period or refund the money you paid for the Software. Symantec does not warrant that the Software will meet your requirements or that operation of the Software will be uninterrupted or that the Software will be error-free.

THE ABOVE WARRANTY IS EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHER RIGHTS, WHICH VARY FROM STATE TO STATE.

- Disclaimer of Damages:

REGARDLESS OF WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE, IN NO EVENT WILL SYMANTEC BE LIABLE TO YOU FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT OR SIMILAR DAMAGES, INCLUDING ANY LOST PROFITS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF SYMANTEC HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

IN NO CASE SHALL SYMANTEC'S LIABILITY EXCEED THE PURCHASE PRICE FOR THE SOFTWARE. The disclaimers and limitations set forth above will apply regardless of whether you accept the Software.

- U.S. Government Restricted Rights:

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c) (1) and (2) of the Commercial Computer Software-Restricted Rights clause at 48 CFR 52.227-19, as applicable, Symantec Corporation, 10201 Torre Avenue, Cupertino, CA 95014.

Language Addendum

If the Software is a Symantec language product, then you have a royalty-free right to include object code derived from the Symantec component (Java source or class) files in programs that you develop using the Software and you also have the right to use, distribute, and license such programs to third parties without payment of any further license fees, so long as a copyright notice sufficient to protect your copyright in the program is included in the graphic display of your program and on the labels affixed to the media on which your program is distributed. You have the right to make changes to the Symantec components, but only to the extent necessary to correct bugs in such components, and not for any other purpose. You also have a royalty-free right to include unmodified (except as stated in the previous sentence) Symantec component files required by your programs, but not as components of any development environment or component library you are distributing. The Symantec component files that may be redistributed are in the following folder in the Visual Cafe directory—`VisualCafe\redist`. The Java Virtual Machine (VM) or Just In Time (JIT) compiler may not be redistributed.

C O N T E N T S

Preface

Chapter 1 Introduction to Distributed Development

Overview of distributed development	1-1
Creating a servant	1-3
Implementation approach	1-3
Interface approach	1-3
Creating a client adapter	1-4
Running the distributed application	1-4
Background of distributed development	1-6
Distributed Applications	1-6
Client and server development	1-7
Remote Method Invocation	1-7
About RMI	1-8
Development under RMI	1-9
Finding more information	1-9
Common Object Request Broker Architecture	1-9
About the Interface Definition Language of CORBA	1-10
Development under CORBA	1-10
Finding more information	1-11
Introduction to Enterprise Rapid Application Development	1-11
Server application	1-11
Developing servants and client adapters	1-12
Distributed Debugging	1-13

Chapter 2 Developing Servant Classes

About Servants and Server Applications	2-1
About server development	2-3
About server application projects	2-3
Creating a server application project	2-4
Building an RMI server application from a Java class	2-5
Building a CORBA server application from a defined interface	2-6
Using the Servant Class Wizard	2-10
Introduction page	2-12
Choosing a source type	2-13

Creating a servant from an interface	2-14
The repository connections tree	2-15
Managing repository connections	2-17
Completing the Select interface page	2-20
Choosing the implementation approach	2-21
Customizing the implementation class	2-21
Creating a servant from a Java implementation	2-26
Selecting the implementation class	2-27
Customizing output files	2-30
Saving your client adapter	2-31
Selecting the component library folder	2-33
Reviewing your selections	2-33
Inserting the servant into a server application	2-36
Executing the servant	2-37
Local servant execution	2-37
Remote servant execution	2-38
Server Project Templates	2-39
Interface/Adapter Updates	2-39
Server application properties	2-39
Global application property settings	2-40
RMI global application property settings	2-41
CORBA global application property settings	2-42
Servant record application properties	2-44
Servant record application properties (CORBA-specific)	2-44
Special Requirements on Running the RMI Registry	2-45

Chapter 3 Developing Client Adapters

Overview	3-1
Creating a client adapter	3-3
Introduction page	3-3
Selecting the servant	3-4
Using the Select Servant tree view	3-4
Saving the client adapter	3-8
Saving the client adapter in the project	3-8
Saving the client adapter in a JAR file	3-9
Selecting the component library folder	3-11
Reviewing your selections	3-12

Customizing a client adapter	3-13
Customizing RMI client adapters	3-13
Advanced RMI settings	3-14
Customizing OrbixWeb client adapters	3-15
OrbixWeb vendor-specific binding	3-15
Inter-operable ORB reference binding	3-18
Naming Service binding	3-19
Customizing VisiBroker client adapters	3-20
VisiBroker vendor-specific binding	3-21
Inter-operable ORB reference binding	3-24
Naming Service binding	3-25

Chapter 4 Distributed Debugging

Overview of distributed debugging features	4-2
About the distributed debugging executables	4-2
The ddservices executable	4-3
The debugvm executable	4-3
Some distributed debugging scenarios	4-4
Installing Distributed Debugging Services	4-7
Installing on Windows	4-7
Running on Windows	4-8
Installing on UNIX	4-9
Running on UNIX	4-9
Verifying network connections	4-10
Attaching the debugger to a VM	4-11
Socket protocol	4-11
Passwords	4-11
Registering the VM	4-11
Configuring the network and port	4-12
Running on multiple machines on a subnet	4-12
Configuring distributed debugging	4-13
Setting debugging parameters for the project technique	4-13
Specifying remote execution settings	4-15
Specifying security settings	4-16
Specifying distributed debugging options for a project	4-17
General category options	4-17
Exceptions category	4-19
Distributed category options	4-20

Running and debugging remote processes	4-22
Starting a distributed debugging session	4-23
Run in Debugger	4-23
Attach	4-23
Debug in Waiting VM	4-24
Comparison of Distributed Debugging techniques	4-25
Starting a debuggable VM	4-26
Attaching to multiple VMs	4-26
Debugging and restarting	4-26
Stopping processes	4-27
Adding Platform support for unlisted JDK versions	4-28
Process factories	4-28
Redirecting display	4-29
Handling socket factory classes	4-29
Process factory classes	4-30
Adding Visual Cafe machine's IP address to comm.properties	4-31
Using the Seamless Stepping Option (RMI only)	4-32
The comm.properties file	4-33
Address	4-37
Troubleshooting	4-40

Appendix A Environment Options for Distributed Development

Distributed Object tab	A-2
Manage CORBA ORBs dialog box	A-4
Client Adapter Wizard Defaults dialog box	A-6
Servant Class Wizard Defaults dialog box	A-7
Interface to implementation method group	A-8
Implementation to interface method group	A-9
New project location	A-9
Debugging tab	A-10
Network Settings dialog box	A-11

Appendix B Exploring Distributed Application Development

A walk-through of the Servant Class Wizard	B-1
Create the server application	B-2
Create the implementation file	B-2
Prepare the servant	B-3
Run the application	B-4
A walk-through of the Client Adapter Wizard	B-5
Using the client adapter	B-7
Exploring distributed debugging	B-9
Glossary	Glossary-1
Index	Index-1

Preface

Welcome to the Visual Cafe Enterprise Suite. This book contains the information you need to take advantage of the distributed development features of Visual Cafe. This chapter gives you an overview of the documentation for Visual Cafe Enterprise Suite.

Stylistic conventions

This manual uses the following typographic conventions:

- ◆ Names of files, resources, classes, methods, and variables, as well as code fragments and information you type, appear in the `code typeface`. Metanames appear in *italic*. A metaname is a descriptive placeholder for a real name. For example, when referring to a project's `.vep` file, we say *projectname.vep*, rather than specifying a specific project name.
- ◆ Terms that appear in the glossary appear in **bold** type when they're defined in the text.
- ◆ Names of menus, menu items, buttons, and other user interface elements appear in this typeface.
- ◆ The ► character is a shorthand for menu traversal. A sequence such as
File ► Open
means "From the File menu, choose Open".
- ◆ Keys you press at the same time are shown as follows: CTRL-G (press the CTRL and G keys simultaneously). Please note that even though the letter keys are listed in uppercase, you should not hold down the SHIFT key when executing these key combinations unless the SHIFT key is listed as part of the combination.

- ◆ We use the word “program” to refer to whatever you’re creating with Visual Cafe, whether it’s an applet, application, library, or servlet.
- ◆ Wherever possible, we use the term “folder” rather than “directory” in accordance with standard Windows style, except in cases where the Visual Cafe interface uses “directory.” Because Windows also uses the DOS system (which primarily uses the term “directory”), and Visual Cafe makes use of this DOS–Windows relationship, some areas of the product deal with “directories.”

Visual Cafe documentation

Visual Cafe Enterprise Suite comes with extensive printed documentation and Online Help to assist you in the process of developing distributed applications.

- ◆ The ReadMe file is the first document you should read before using Visual Cafe. It contains late-breaking news, work-arounds, and known issues. This file is available for viewing at the end of the installation process, as well as from the Windows Start menu.
- ◆ *Visual Cafe User’s Guide* contains all the information you’ll need to use the rest of the standard (non-databound) parts of Visual Cafe. This book includes information on projects, compiling, and debugging your programs. It also provides troubleshooting information for your Visual Cafe installation and an overview of JFC programming.
- ◆ *Visual Cafe Getting Started* consists of a tour of the main features of Visual Cafe.
- ◆ *Visual Cafe Sourcebook* presents coding samples, in the form of applications and servlets, that were developed in Visual Cafe, and describes them in detail so you can understand how they work and how they were built. You can modify the code in this book to build your own applications, applets, and servlets.
- ◆ *Visual Cafe Database Developer’s Guide* contains all the information you need to take advantage of the powerful databound features of Visual Cafe—both the conceptual and the procedural information for developers of data-aware Java programs using Visual Cafe Database Edition.

- ◆ Visual Cafe has extensive Online Help that describes all of the procedures for building distributed applications in Java. To access the Visual Cafe Help, choose Help Topics from the Help menu. The Online Help is also context-sensitive, which means that you can press F1 in most areas of Visual Cafe and receive information that pertains to your current activity.

You can also access information about Visual Cafe components and review the Java API documentation from Sun Microsystems. You can access information about an individual component by typing the name of the component in the Index tab of the Help window, choosing Components Reference from the Help window's Contents tab, or selecting a component in the Component Library and pressing F1.

Visual Cafe also includes Online Help for the Java macro system. To access this help topic, choose Macro Reference from the Help menu.

- ◆ The *Visual Cafe Enterprise Suite Getting Started* provides a tour that guides you through a development scenario that includes installing, configuring, building and running client- and server-side components, and performing Distributed Debugging on a remote machine.
- ◆ This manual, the *Visual Cafe Enterprise Suite Distributed Development Guide*, may be the document you turn to most frequently as you work with Visual Cafe Enterprise Suite. It contains both conceptual information and step-by-step procedures.

Chapter 1 introduces the history and the current tools of distributed development.

Chapter 2 discusses how to develop servants.

Chapter 3 discusses how to develop client adapters that enable clients to communicate with servants.

Chapter 4 discusses distributed debugging.

Appendix A discusses the Environment Options tabs that were added for Visual Cafe Enterprise Suite.

Appendix B provides a walkthrough of one path through the Servant Class Wizard, the Client Adapter Wizard, and some distributed debugging.

Portable Document Format (PDF) versions of the books are included with your copy of Visual Cafe. These documents require that Adobe Acrobat Reader be installed. Adobe Acrobat Reader is included on the installation CD-ROM of your Visual Cafe product. It's also freely available from Adobe Systems at <http://www.adobe.com>.

System requirements

The *development system* on which you run Visual Cafe Enterprise Suite should satisfy these minimum requirements:

- ◆ Windows 95/98 or Windows NT 4.0 (or greater).
- ◆ 133 MHz (or greater) Pentium or compatible.
- ◆ 96 MB RAM (128 MB recommended).
- ◆ 205 MB hard disk space (540MB recommended).
- ◆ CD-ROM drive.
- ◆ 256 color VGA monitor (16-bit Super VGA recommended).
- ◆ TCP/IP network connectivity, RMI or CORBA.

The *remote system* can operate on a platform different from the development system. Platforms fully supported by Visual Cafe Enterprise Suite at this time include

- ◆ Windows (95, 98, and NT 4.0).
- ◆ UNIX (Solaris, HP-UX, Compaq True 64 Unix).

Introduction to Distributed Development

Visual Cafe Enterprise Suite makes available a new category of Java development tools, defined as enterprise rapid application development tools. These tools facilitate the creation of sophisticated server-based enterprise applications leveraging the Internet and corporate intranets and extranets.

This chapter includes:

- ◆ Overview of distributed development
- ◆ Background on:
 - ❖ Distributed applications
 - ❖ Client and server development
 - ❖ Remote Method Invocation (RMI)
 - ❖ Common Object Request Broker Architecture (CORBA)
 - ❖ Enterprise Rapid Application Development
- ◆ Introduction to Distributed Debugging

Overview of distributed development

Enterprise applications mark the real potential of Java technology as more than just a language for creating dynamically downloaded client software (applets). Server-side Java will proliferate throughout the enterprise and beyond it into the extended enterprise of suppliers, partners, and customers.

The Visual Cafe Enterprise Suite of distributed application development tools is the industry's first Java-based rapid application environment for use in building large-scale distributed applications. It is designed to interoperate with virtually any system on any platform, including non-Java legacy systems. This edition of Visual Cafe includes a unique and sophisticated *Single-View* technology that makes developing distributed applications look as if you're developing for a single machine. You can debug simultaneously against multiple heterogeneous servers from a single console. Other key capabilities that put the inherent power of Java to work for server-based distributed systems include deployment platform independence and distributed applications debugging.

Visual Cafe Enterprise Suite helps you to create distributed applications composed of client adapters and servants—server objects—that communicate using either the Java Remote Method Invocation (RMI) mechanism or the Common Object Request Broker Architecture (CORBA) mechanism.

To create a distributed application, you could use all the tools listed below. More commonly, though, you will use some combination of these operations according to your needs.

- 1 Using Visual Cafe (File ► New Project), create a server application project (CORBA or RMI) in Visual Cafe.
- 2 Using the **Servant Class Wizard** (File ► New Servant Class), either create a servant adapter and an interface definition for an existing servant implementation, or create a servant template from an existing interface. The servant will execute on a remote system.
- 3 Using the **Client Adapter Wizard** (File ► New Client Adapter) and an interface definition, create a client adapter that a client can use to invoke methods on the remote servant—the client invokes methods on the local client adapter, which invokes methods on the servant through RMI or CORBA mechanisms.
- 4 Using the *Single-View* technology of the **Distributed Debugging Services**, use one workstation to debug the client executing locally and the servant executing remotely.

Creating a servant

A **servant** is a class that, when instantiated as a remote object (an object executing on a remote system), is used as part of a distributed computing model, such as RMI or CORBA. You can create servants that communicate with clients that were not created by the Client Adapter Wizard.

You can use the **Servant Class Wizard** to create a servant, through either the **implementation** approach or the **interface** approach.

Implementation approach

In the implementation approach to creating a servant, you start with a class that executes the server logic. You present this class (by selecting the `.class` file) to the Servant Class Wizard, and the wizard creates a **servant adapter**. The server application (see page 1-11) instantiates the servant adapter, which instantiates your class. The servant adapter plus your class constitute the servant. The servant adapter receives requests on behalf of the class and passes them along to the class, then receives its responses and passes them back to the requester.

Interface approach

In the interface approach, you start with an interface definition in one of the following:

- ◆ a CORBA interface in an `.idl` file (CORBA)
- ◆ a CORBA interface defined in an Interface Repository
- ◆ a CORBA interface defined in a Java `.class` file
- ◆ an RMI interface described in a Java `.class` file

You specify an interface and the Servant Class Wizard generates a template with empty method bodies, and (as necessary) the associated classes (stubs, skeletons, helpers, holders, and so on) to allow the class to run on a remote machine. You then provide the code to implement the methods defined in the interface.

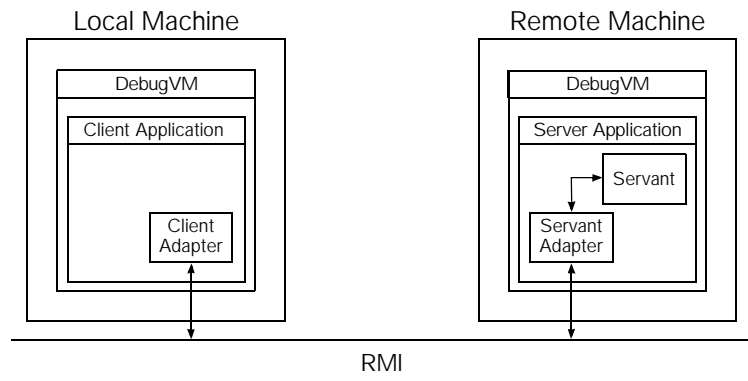
Creating a client adapter

Once you have a servant, you can use the **Client Adapter Wizard** to create a **client adapter**—a proxy class that handles the CORBA or RMI communication with that servant. In this way, you can simply call methods on this local client adapter proxy and need not be concerned with the details of the distributed communication with the servant that actually implements the methods.

The Client Adapter Wizard packages the client adapter as a Bean so that you can easily customize it, drop it into your client project, and use the other tools of Visual Cafe such as the interaction wizard to manipulate it. Client adapters can be created for any servant interface or any running, published servant. You can create client adapters for servants that were not created by the Servant Class Wizard.

Running the distributed application

When you have a servant and you have a client adapter in your client, you can execute the distributed application. The illustration below shows an RMI implementation, in which a servant adapter interfaces between the client adapter and the servant.



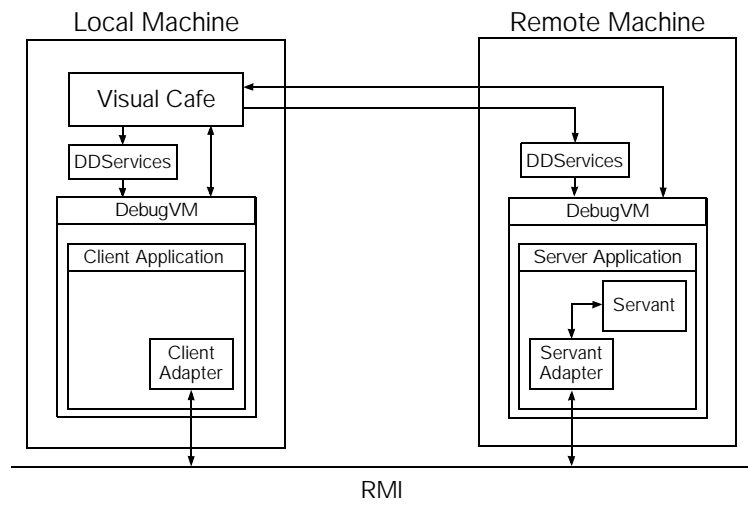
You insert the servant into a server application (either automating this in the Servant Class Wizard or doing it by hand). If you are using a server application generated by Visual Cafe, when you insert the servant its signature information is recorded in the `ServerApp.properties` file. When you execute the server application it uses this `.properties` file,

instantiates any servants listed there, and publishes their availability on a CORBA Object Request Broker (ORB) or an RMI Registry.

You can install the server application on a remote machine that has a Java Virtual Machine (VM), run the client on your local machine, and have the local client communicate with the remote servant.

Using the Visual Cafe Enterprise Suite Distributed Debugging Services, you can launch a VM on a remote machine. This VM will then download the classes for the server application and execute them. Using the *Single-View* technology of the debugger, you can debug the combination of a local client and remote servant while controlling and viewing it all on your local machine.

The working model of the complete distributed RMI application looks like this:



The rest of this chapter examines the technology in somewhat more detail. If you are versed in distributed computing, CORBA, and RMI, you can jump forward to Chapter 2, “Developing Servant Classes.”

Background of distributed development

Before we can discuss the details of the Visual Cafe implementation, it's useful to review the underlying technologies of modern distributed application development. In this section, we discuss RMI (on page 1-7), and CORBA (on page 1-9). We then examine the capabilities of Visual Cafe (on page 1-11) and Distributed Debugging (on page 1-13).

Distributed Applications

Modules are easier to maintain when they are self-contained, and the dependencies and interfaces are few and well-defined. When tasks are localized one to a unit, and communication between units occurs through standardized interfaces, then individual units can be changed without disrupting the larger system, as long as they continue to use the same interface. The separate units can even be distributed across a network, or run on different hardware families and operating systems. Today, objects in object-oriented code make distributed development even cleaner, because objects are units that contain both code and data.

The objects that execute on the machine where the data is stored can handle the tasks involving data input and output as well as the computation done on that data. If this same object is on a remote machine, then it streamlines operations further by sending only the results to the machine where they are needed—rather than sending all the data to the display machine to be operated on there.

Objects on the machine where the user makes requests and views results can be streamlined to handle only the data capture and display layout activities. In this model, objects that make requests are termed **clients** and objects that service those requests are termed **servers**.

The Java programming language is ideal for creating distributed applications that are composed of clients and servers, because it has a rich vocabulary of display mechanisms for clients, and can be run on a large number of platforms. This enables you to focus on a specific programming task and not be concerned with platform-specific details.

Client and server development

A **client** object requests the services of another object. In many cases, a client features a graphical user interface. A **server** object responds by providing a service to the invoking client.

When the server is running on a separate computer, connected to the client's computer through the Internet, the exchange of data occurs through the standard Internet protocol: TCP/IP. This protocol requires the client to open a socket in its operating system which it uses to send data through a network, and the server to open a socket in its operating system through which it accepts data.

Because dealing with all the details of communication through sockets is complex, programming languages have been enriched by mechanisms such as Remote Procedure Calls (RPC). These mechanisms enable a client to call a method on a server as though it were running on the same computer. The RPC takes care of all the details.

Because of the way some early client/server communication was implemented, some vendors advised their customers to use only clients and servers. Now that the infrastructure has evolved and been standardized, it has become completely feasible to have a client call a server that must, in turn, call another server. In this *three-tier system*, the first server is acting as a client with respect to the second server. A whole class of such double-agent programs has arisen and been christened *middleware*. Designers are planning *n-tier* systems with multiple layers of programs acting as clients making requests of other programs acting as servers.

Java's native mechanism for making remote calls is RMI, but it also can use the Common Object Request Broker Architecture: CORBA.

Remote Method Invocation

RMI was developed by Sun Microsystems to extend the pure Java model to the network. It enables objects in a local VM to invoke methods of objects in a remote VM, on the company intranet or out on the Internet.

Remote Method Invocation (RMI) is a programming technology—a Java API. It does not use an open-standard protocol, and is Java-to-Java only. The RMI protocol enables a local Java applet or application to communicate with a remote Java application. Through RMI, your client

application can invoke methods on a server object anywhere on the network.

About RMI

Before a client Java applet or application can call a remote (server) object, the remote object must register its interface with the RMI Registry (a part of the RMI implementation). RMI clients interact with their remote objects exclusively through these published interfaces.

Local Java calls pass parameters by reference, but remote Java calls must pass parameters by value, because references are valid only within the same VM. RMI uses object serialization to pass objects as parameters, sending code and data to the remote VM as a stream of bytes. This enables RMI to pass many built-in Java objects such as vectors, hashtables, and dates. RMI can also pass and return user-created objects between VMs.

In order to pass parameters in RMI, you must generate stubs and skeletons. These objects are specialized to code parameters into transportable objects and then to decode the parameters when they're received.

Stubs and skeletons

To call a method of a remote object, a local object calls a regular Java method that is encapsulated in a local proxy object called a **stub**. The stub packages the parameters. It serializes objects and sends them to the remote system. The process of encoding parameters into a transportable format is called **parameter marshalling**.

On the server side, the parameters are delivered to a **skeleton** object that unmarshals the parameters (translates the encoding) and passes them to the called method. In turn, the skeleton encodes any result that the method is returning, and sends it back to the stub. The skeleton also passes exceptions back to the caller.

Setting up stubs and skeletons requires more preparation than just using local methods. You must run Java programs on both the client and the server. The stubs and skeletons must be created, and there must be a mechanism (RMI Registry) for the client to use to find the objects on the server.

Development under RMI

The following steps represent the typical sequence a developer follows when coding for RMI by hand. Visual Cafe automates much of this for you, so you do not have to handle all these details.

To establish a distributed (client/server) application under RMI:

- 1 Define the remote interface for the server by extending the `java.rmi.Remote` interface.
- 2 Implement the remote interface in a Java server class that derives from `java.rmi.UnicastRemoteObject`.
- 3 Compile the server class.
- 4 Run the `rmic` stub compiler against your class to generate stubs and skeletons for the remote class.
- 5 Start an RMI registry on the host where the server is to run.
- 6 Load the server class and instantiate the servant from it.
- 7 Register the servant with the registry, so that clients can find it.
- 8 Create (write code for) the client; it must invoke the servant by invoking a stub as a proxy for the corresponding servant.
- 9 Compile the client.
- 10 Load the client code and instantiate client objects that can invoke the servant.

Finding more information

There is quite a bit of information about RMI on the Sun Java Web site. To find it, go to the search page <http://www.javasoft.com/cgi-bin/search.cgi> and search on RMI. Also, books such as *Core Java, Volume II - Advanced Features* (Cay Horstman and Gary Cornell), and *Client/Server Programming with Java and CORBA* (Robert Orfali and Dan Harkey), which addresses RMI as well as CORBA, are good resources.

Common Object Request Broker Architecture

Common Object Request Broker Architecture (CORBA) is an architecture developed by the OMG (Object Management Group) to enable a client object to communicate with a server object through a language-independent interface. The server object can be on the same computer, on a different computer on the same intranet, or on a remote computer

anywhere on the Internet. Because the CORBA interface is language independent, the client object could be in Java, while the server object could be implemented in C++ or some other language. Thus, CORBA is an integration technology, designed to facilitate cooperation between objects, regardless of their implementation languages.

About the Interface Definition Language of CORBA

CORBA functions as a communication medium that is language-independent, by requiring that the interfaces between components be described in its own Interface Definition Language, IDL. These definitions must be compiled into the language of the client and the language of the server before they can be used. Vendors provide ORBs that mediate communication between clients and servers. Each vendor also provides compilers that translate interfaces from IDL to target languages, such as Java.

Development under CORBA

The following steps represent the typical sequence a developer follows when using CORBA and coding by hand. Visual Cafe automates much of this for you, so you do not have to handle all these details.

To establish a distributed (client/server) application under CORBA:

- 1 Define the remote interface for the server by writing IDL definitions of the servant, its attributes, its methods, and the parameters of the methods.
- 2 (Optional) Register the interface definition with the Interface Repository.
- 3 Compile the interface definition from IDL to declarations of a particular language such as Java, producing (client) stubs, (server) skeletons, and a template class.
- 4 Add your code to the template class, to create the servant code.
- 5 Compile the servant.
- 6 Register the servant with the Implementation Repository so that clients can find it.
- 7 Load the servant class and instantiate a servant from it.
- 8 Implement the client; it must invoke the servant by calling a stub, which serves as a proxy for the corresponding servant.
- 9 Compile the client.

- 10 Load the client code and instantiate client objects that can invoke the servant.

Finding more information

Because CORBA is the product of the Object Management Group, their Web site (<http://www.omg.org>) is a good place to start a search for online CORBA documentation, including the CORBA 2.1 specification (<http://www.omg.org/pub/docs/formal/97-09-01.pdf>).

Useful books include *Client/Server Programming with Java and CORBA* (Robert Orfali and Dan Harkey), and *Java Programming with CORBA* (Andreas Vogel and Keith Duddy). The principal ORB vendors, such as:

- ◆ Iona: OrbixWeb
- ◆ Inprise: Visibroker

also provide extensive documentation with their products.

Introduction to Enterprise Rapid Application Development

The Symantec Visual Cafe Enterprise Suite provides an enterprise rapid application development tool. By using wizards (filling out a series of forms), you can create RMI or CORBA applications without digesting hundreds of pages of documentation about the powerful technologies involved.

Server application

Servants (server objects) spend much of their time waiting for a client to make a request. However, it would be very wasteful to have the servant start for every request, and terminate after every reply because there are many startup and shutdown activities the operating system must perform. It is much more efficient for the servants to be launched before the client makes a request.

A **server application** is a Java application that launches servers and registers them with a CORBA interface repository or an RMI registry. Once a server application is started and registered, developers can use the server application to construct the client adapters that can communicate with them.

In the Visual Cafe Enterprise Suite system, you start by creating a project that builds either an RMI server application or a CORBA server application. Then you can use a wizard to build a servant—a server object for the server application to instantiate—and add more Servants as needed. When a server application has begun executing, and instantiated one or more servants, you can use another wizard to create a client adapter that can make requests to the servant.

Developing servants and client adapters

The following steps outline the process of using the Visual Cafe Enterprise Suite wizards for creating servants and client adapters, for either RMI or CORBA.

To create a servant:

- 1 If you are using CORBA, set the appropriate Current CORBA Orb value on the Distributed Objects tab in the Environment Options dialog box.
- 2 Create a server application project as a vehicle for instantiating remote servants.
- 3 Launch the Servant Class Wizard.
- 4 Using the wizard, create one or more servants.

The definition for these servants can come from an interface or a Java implementation class.

- 5 Add the servants to the server application. This can be done by hand or by the Servant Class Wizard.

To create a client adapter:

- 1 Create or open a project to manage the client development.
- 2 Run the Client Adapter Wizard to create one or more client adapters that will be used to communicate with the remote servants in the project.
- 3 Add the client adapters to the project.

- 4 Customize each client adapter with respect to its binding methods, related servant, and so on.

To use the client adapter to connect a client to a servant:

- 1 Create or update the source code of a client project to call methods on the client adapter to invoke the servant through its servant adapter.
- 2 Load the client code and instantiate client objects that can invoke the servant.

Distributed Debugging

The Professional and Database Editions of Visual Cafe have always offered integrated debugging features. You can watch your programs execute line by line, and observe how the various components of the program are behaving. You can monitor the values stored in variables, the flow of control in the program, and so on. You can also set breakpoints to control where execution stops, so you can examine variables as you wish. But all of that was within one Java Virtual Machine (VM).

With the Visual Cafe Enterprise Suite and its distributed debugging, you can observe the behavior of the different elements of a distributed application, on different VMs. You can watch the code in the client, and when the client calls the servant, you can either let the servant execute in its entirety and resume watching in the client after the servant is done, or you can set breakpoints in the servant and watch what is happening there as well. One VM on the development system orchestrates all this activity, but separate VMs either on the same machine or on remote machines can also cooperate in the debugging. You have a single source of debugging across multiple computer systems.

Developing Servant Classes

This chapter provides:

- ◆ an explanation of the server application
- ◆ brief outlines of creating a servant from a class and from an interface
- ◆ a discussion of the Servant Class Wizard, examining each page and what it does
- ◆ an explanation of the `ServerApp.properties` file that is used to configure the server applications based on the Visual Cafe ServerApp templates

About Servants and Server Applications

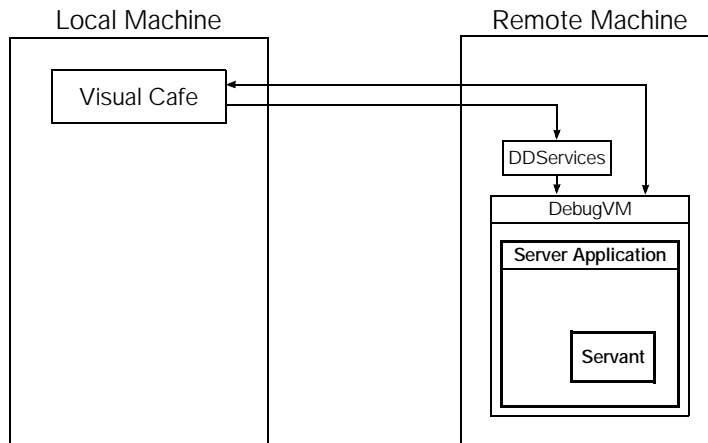
The Servant Class Wizard of Visual Cafe Enterprise Suite enables you to take a Java class or a description of an interface, and develop a Java server object, known as a **servant**. Each servant can communicate with clients. To manage the execution of one or more servants, you can use Visual Cafe to generate a **server application** in Java for either RMI or CORBA. The server application runs on a remote machine in a Java Virtual Machine (VM) that communicates with Visual Cafe.

If you run the servant in a VM equipped for use in debugging, termed a **debuggable VM**, you can use the Visual Cafe debugger on the local development machine to debug the servant on a remote machine.

Distributed Debugging Services (DDServices), which you install on the remote machine, assist in:

- ◆ remote debugging
- ◆ serving class files to other hosts
- ◆ starting remote debuggable VMs
- ◆ keeping a list of available debuggable VMs
- ◆ keeping a list of locations of available DDServices hosts (see Chapter 4)

The following illustration shows the relationship between the local development machine and the related applications, servants, and services that run on the remote machine.



When you create the server application project, you can define a servant as part of that project. That servant, and any subsequent ones you create, can be in the same project. If you choose to organize all your reusable servant logic in one place, you can put all the servants into a separate project to be shared with other development efforts.

When run—either manually or by some activation policy—the server application instantiates one or more servants, making them accessible to clients.

About server development

Server development is similar to library development. The server's objects provide an API that is made available to clients. This API is a set of interfaces, published in an RMI Registry or a CORBA Interface Repository. Although this API specifies how the server can interact with its clients, the real “owner” of the API is the server.

The client deals with whatever API the server implements and exports. For this reason, IDL or Java object APIs can be incorporated in the following ways:

- ◆ as part of the server project
- ◆ through Beans
- ◆ as source code included in the project

At the point where a “remotable” object interface—one describing an object that can be adapted to execute remotely—has been identified, it can be inserted directly into the server project.

About server application projects

A server application project generates a Java application that, when run either manually or by some activation policy, instantiates a server process (the server application) that publishes one or more servants. To build a server application in Visual Cafe, you start by creating a server application project. Visual Cafe Enterprise offers two new project types:

- ◆ CORBA Server
- ◆ RMI Server

Creating the servants begins with a choice between “Building an RMI server application from a Java class” on page 2-5 and “Building a CORBA server application from a defined interface” on page 2-6.

Creating a server application project

No matter whether your servant uses RMI or CORBA, no matter whether it was generated from an implementation or an interface, your servant executes in either an RMI server application or a CORBA server application. After you create the server application project, you can create one or more servants by using the Servant Class Wizard. Next, you insert an instance of the servant in the server application, and then launch the server application to run under RMI or CORBA, for clients to use.

An RMI server application checks for the presence of an RMI Registry. If it does not find one, it starts one. A CORBA server application relies on the presence of an ORB.

To create an RMI or CORBA server application project:

- 1 Choose File ► New Project.
- 2 In the New Project dialog box, select either RMI server or CORBA server, depending on your choice of distributed processing protocols.
- 3 Click OK.

Visual Cafe creates the server application file, `ServerApp.java`, that provides the basic code required for connecting to an ORB or an RMI registry, and activating a set of servants. Visual Cafe also creates a `ServerApp.properties` file that contains a set of parameters to configure the server application, as well as the list of servants to be created. See “Server application properties” on page 2-39.

The next two sections provide a conceptual overview of the steps involved in building two kinds of server application: an RMI server application built from an existing class, and a CORBA server application built from an interface.

Building an RMI server application from a Java class

To build an RMI server application, create a new RMI server application project, and save it as explained in “Creating a server application project” on page 2-4. This brief discussion is only to give an idea of the ease with which you can create the server. Details are discussed later.

If you are starting from a `.java` file that contains an executable class (as contrasted with an interface), it must be declared as a `public` class.

Either add the `.java` file to the project and compile it, or add the class to the project.

To add a Java file to the project:

- 1 Open the source file by choosing File ► Open.
- 2 Add the `.java` file to the project by choosing Project ► Add *filename*.
- 3 Compile it by choosing Project ► Compile.

To add a class file to the project:

- ◆ Insert the `.class` file to the project by choosing Insert ► Files into Project.

To create an RMI servant from the implementation class:

- 1 Launch the Servant Class Wizard by choosing File ► New Servant Class.
Visual Cafe displays the Introduction page.
- 2 Click Next to display the Choose source type page.
- 3 Click the Java implementation radio button to indicate that you are starting from a Java class file.
- 4 Click Next to display the Select implementation class page.
- 5 Click Browse beside the Implementation class text field.
Visual Cafe displays the Select Class dialog box.
- 6 Locate the class file, select it, and click Open.
The file name appears in the Implementation class text field.
- 7 Click Next.
Visual Cafe displays the Customize output files page.

8 Click Next.

Visual Cafe displays the Review options page where you can examine the values and go back to make any necessary adjustments by using the Back button.

9 Click Finish to direct the wizard to complete its task.

The Servant Class Wizard then creates the servant adapter class. If a project is selected, and the Insert servant checkbox is selected, the servant adapter properties are then saved in the `ServerApp.properties` file.

To use your Java program as a server, you must move the three files (`.class`, servant adapter, and server application) to the Java VM on the target machine and execute the servant adapter on that Java VM. Executing the servant adapter causes the server application to be executed (if it is not already executing), publishes (binds) the server to the RMI Registry, and launches the server.

Building a CORBA server application from a defined interface

When you have an interface in the form of an `.idl` file, or an interface registered with either a CORBA Interface Repository or an RMI Registry, or a `.class` file that contains a Java interface, the Servant Class Wizard can create a template implementation class from that interface.

You must choose between two approaches, known as delegation and inheritance:

In the **delegation** approach Visual Cafe creates two classes. One is called the **tie** class because it does the binding and receives the method calls from the client. Then the tie class makes method calls on the other the **delegate** class, which does the work.

In the **inheritance** approach Visual Cafe generates the template class known as the **ImplBase** class. You then write a new class that extends the

`ImplBase` class, so that this new class can receive the method calls from the client and do the work.

To build a CORBA server application, create a new CORBA server application project, and save it as explained in “Creating a server application project” on page 2-4.

To create a CORBA servant:

- 1 Launch the Servant Class Wizard by choosing File ► New Servant Class.
Visual Cafe displays the Introduction page.
- 2 Click Next to display the Choose source type page.
- 3 Click the Interface radio button to specify that you are starting from an interface.
- 4 Click Next to display the Select interface page.
This includes a tree of all interface sources, such as `.idl` files, registered interfaces, and `.class` files of interfaces.
- 5 Select the interface you want:
 - a If your interface is visible, select it; if it is not, explore the tree until the interface is visible, and select it.
 - b If it is not in the displayed tree, you must either add the interface to one of the displayed repositories (Interface Repository, RMI Registry, or folder of interface files) or add the repository that contains the interface to the tree. You do this by using the Manage Repository Connections dialog box, accessible through the Manage Repositories button.
 - c Click Next to display the Choose implementation approach page.
- 6 Choose between delegation and inheritance approaches, and click the corresponding radio button.

7 Click Next.

If you selected Delegation

The Customize implementation file page displays, offering you the opportunity to specify:

- the implementation class name,
- the object from which it extends, and
- the project where it will be stored.

If you selected Inheritance

The Customize implementation class page displays, offering you the opportunity to specify:

- the implementation class and
- the project where it will be stored.

You can also choose at this point whether to insert an instance of the servant into the current server application, and whether to create a client adapter for the servant. Of course, the servant must match the server application—both are RMI, or both are CORBA for the same vendor.

8 Click Next.

If you chose to create a client adapter

A Save Client Adapter page appears. Choose whether to save the adapter in a project, in a JAR file or both, whether to include a serialized instance (and whether to customize that instance), and click Next.

If you chose not to create a client adapter

No Save Client Adapter page appears.

The Review options page appears.

- 9 Examine the values and go back to make any necessary adjustments by using the Back button, then returning to the Review page by repeated pressing of the Next button.

The Servant Class Wizard then creates a number of classes. If your initial interface was called `Store`, for example, the Servant Class Wizard generates:

- ◆ `Store` The Java interface class
- ◆ `_StoreImplBase` The abstract Java class for the inheritance approach

If you followed the delegation approach, these two classes are generated:

- ◆ `StoreDelegate` The delegation template class
- ◆ `_tie_Store` The connector to the delegation class

If you followed the inheritance approach, this class is generated:

- ◆ `StoreImpl` The inheritance template class

The ORB you configured in the Environment Options dialog box also generates a number of stub, skeleton, and other classes.

The next portion of this chapter describes the pages of the Servant Class Wizard in closer detail.

Using the Servant Class Wizard

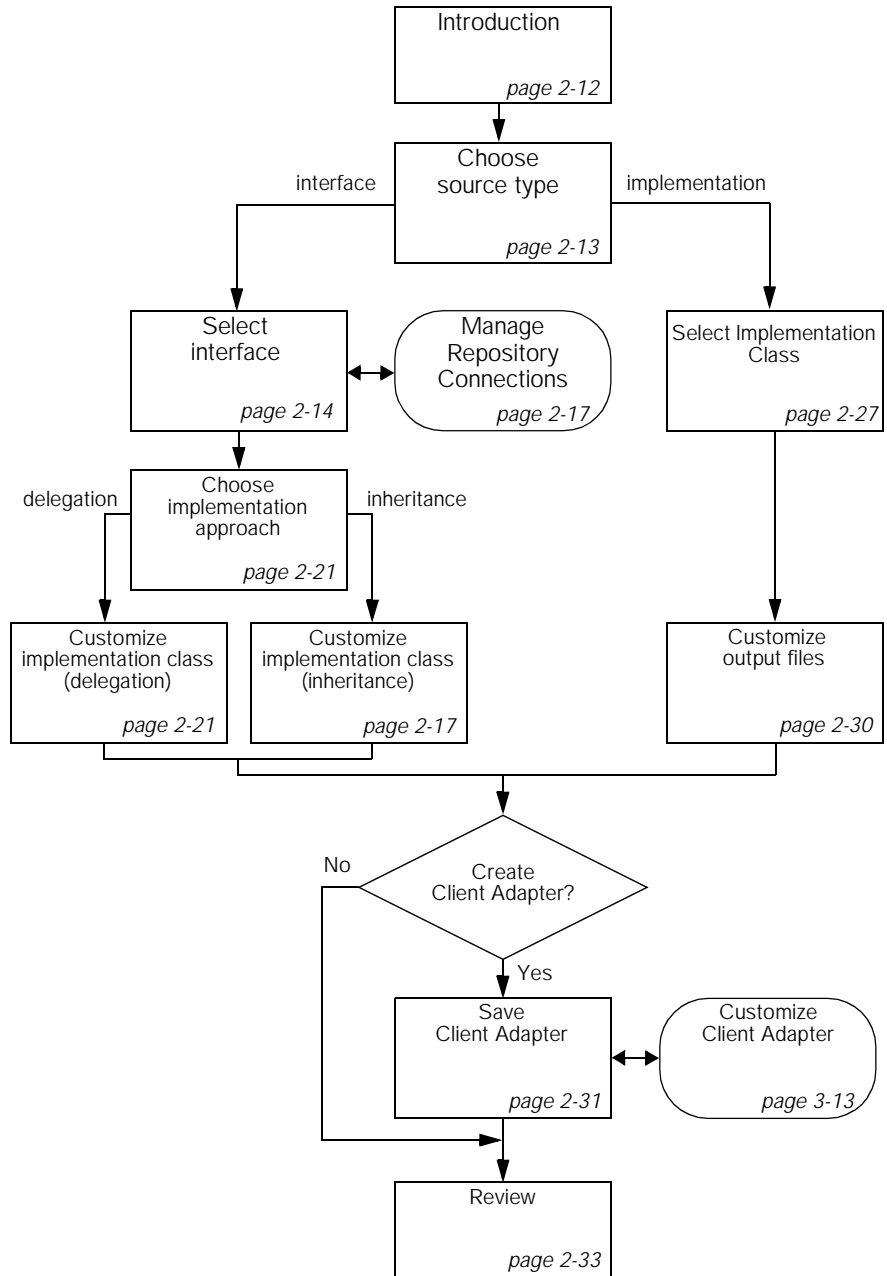
You use the Visual Cafe Servant Class Wizard to create a servant that performs tasks for a client and handles the communication with that client.

This wizard performs one of two general functions:

- ◆ If you give the Servant Class Wizard a remote interface—from a file, a CORBA Interface Registry, or an RMI Repository—the wizard creates a group of files, one of which is a code template for you to complete. You must write the source code to implement the (stubbed-out) methods in the template. The collection of classes embodied in the files constitute the servant.
- ◆ If you give the Servant Class Wizard a Java class (an implementation), the wizard generates a servant adapter class that is a wrapper for the original class, plus a remote interface file. The original class plus the adapter class constitute the servant.

Creating each of these involves a number of decisions, but the Servant Class Wizard does almost all of the routine work for you. You'll need to implement the business logic, but the wizard generates all the classes that manage the RMI or CORBA interactions for you.

The following figure gives a high-level view of the Servant Class Wizard. Each rectangle represents a page of the wizard and shows where in this chapter that page is discussed. Choices you make on these pages can take you down different paths. The ovals represent optional configuration activities.

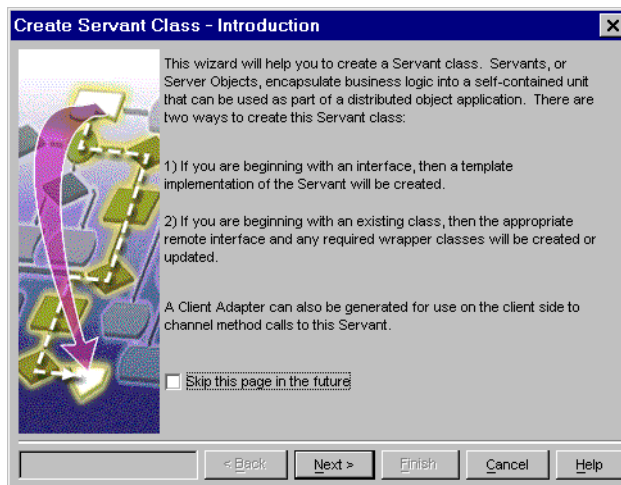


You can then use a client adapter created by the Client Adapter Wizard—see “Developing Client Adapters” on page 3-1—or a hand-written client, to pass method calls from a client applet or application to the servant.

Now that you have had a quick preview of the Servant Class Wizard, let’s explore it in more detail. Select File ► New Servant Class to start the Servant Class Wizard.

Introduction page

The first time you start the Servant Class Wizard, it displays the Introduction page.



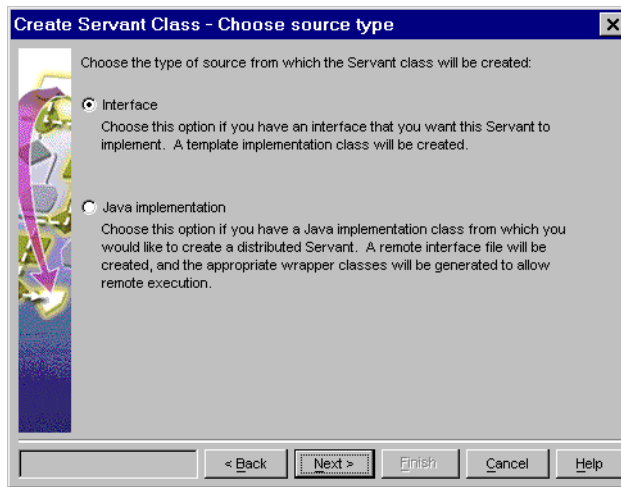
Once you have read and understood it, you can select the Skip this page in the future checkbox to instruct Visual Cafe not to display it every time you use the wizard. To reset this option, you can click the Back button on the Choose source type page.

When you have finished with the Introduction page, click Next to proceed to the Choose source type page, where you can specify a source for the servant.

Choosing a source type

You use the Choose source type page to tell the Servant Class Wizard whether the source of your servant is a remote interface or a Java implementation.

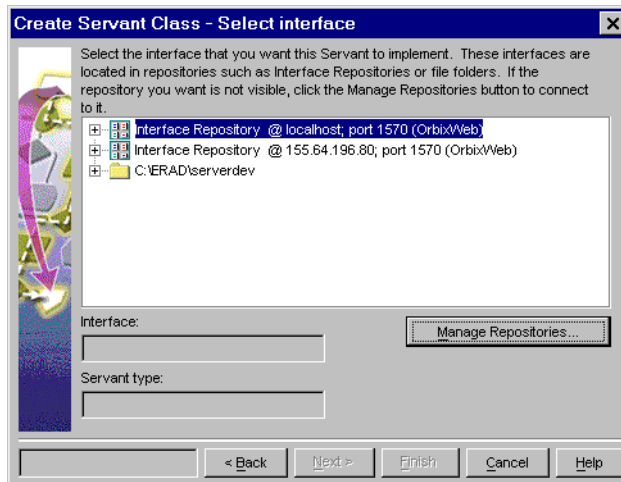
A remote interface can be described by an `.idl` file (CORBA only), a Java interface `.class` file, or an interface that has been registered in a CORBA interface repository. A Java implementation is simply a class that you want to use as a servant, so that it can run remotely.



Select either Interface or Java implementation and click Next. The next section discusses the Interface option. For the Java implementation discussion, see “Creating a servant from a Java implementation” on page 2-26.

Creating a servant from an interface

If you selected Interface as your source type, the Servant Class Wizard displays the Select interface page. This page presents a tree containing known interfaces from which you select the one you want the servant to implement, as shown in the following illustration.



The interface displays a hierarchical tree of repository connections, displayed as containers and sources.

Containers include:

- ◆ Folders (folder icon with + sign)
- ◆ Modules or packages (folder icon with + sign)
- ◆ .class files (a file icon with + sign)
- ◆ .idl files (an idl file icon with a + sign)
- ◆ CORBA Interface Repository (its icon with a + sign)

Sources include:

- ◆ CORBA interfaces in Interface Repositories
- ◆ .class files
- ◆ .idl files (Interface icon. with + sign)

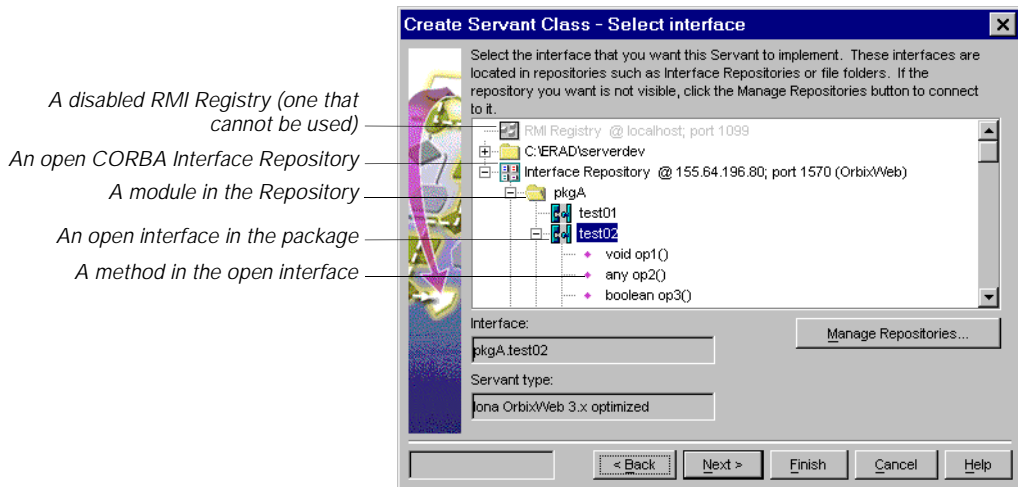
Each repository is a CORBA interface repository, an RMI registry, or a file folder where the wizard can find interface definitions.

You can click any plus (+) icon to expand a collapsed branch of the tree, or click any minus (-) icon to collapse its branch. You can add other interface sources by using the Manage Repositories button.

You can expand sources to display their methods (method icon).
















The repository connections tree

The visible part of the tree in the image below illustrates a number of typical icons :



Beside each icon in the tree, an identifying word or phrase appears. The meanings of the icons are shown in the following table.

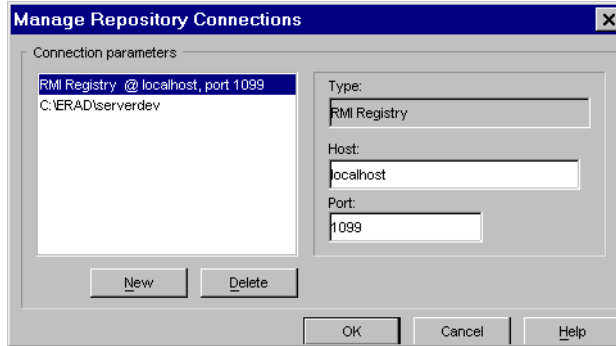
Icon	Description
	A CORBA Interface Repository that can be used
	A CORBA Interface Repository that Visual Cafe is connecting to
	A CORBA Interface Repository that Visual Cafe can't connect to
	A CORBA Interface Repository that cannot be used (Visual Cafe is working on an RMI server application)
	An RMI Registry that can be used

Icon	Description
	An RMI Registry that Visual Cafe is connecting to
	An RMI Registry server that Visual Cafe can't connect to
	An RMI Registry that cannot be used for the Servant Class Wizard—it already contains a servant
	A folder (directory) in a file system or a module in a CORBA interface repository
	A folder or module that is no longer available
	An object
	A disabled object (no stub is available)
	An interface
	An IDL file
	An invalid IDL file
	A Java interface file
	An invalid Java interface file
	A class file (might be used as an implementation file)
	An invalid class file
	A method

The order of the display of repositories in the hierarchy is determined by the sequence in which they were defined.

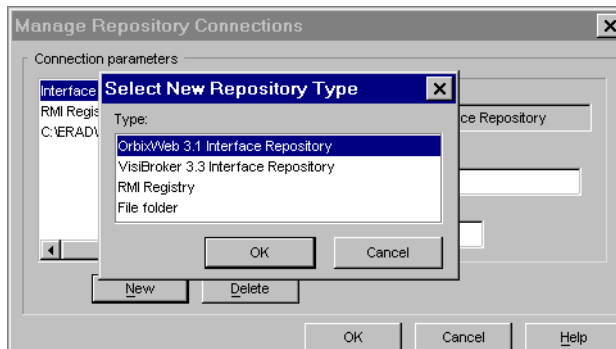
Managing repository connections

If the repository that you need has no connection defined, you can click Manage Repositories to open the Manage Repository Connections dialog box.



The Manage Repository Connections dialog box lists the available repositories. It allows you to add, edit, and delete all types of repositories.

When you click New to add a new repository connection, the Select New Repository Type dialog box appears, and you must select a type:



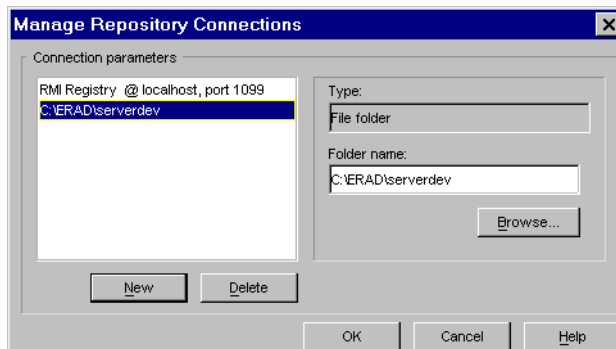
The four repository types are:

- ◆ OrbixWeb 3.1
- ◆ VisiBroker 3.3
- ◆ RMI Registry
- ◆ File folder

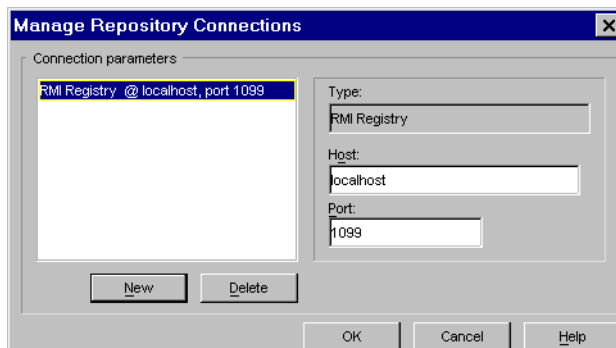
Once you have selected a type, the Type field becomes read-only so that only the other fields can be changed.

When you select a connection in the list box, the controls for that connection appear in the frame to the right. Enter any parameters needed to connect to the desired repository.

When you select a file folder, as shown in the following figure, you can click Browse to open a dialog box that allows you to select any folder.



After you have created a repository type, the entry displayed in the list box is “untitled”. You must complete the remaining fields in the right half of the dialog box.



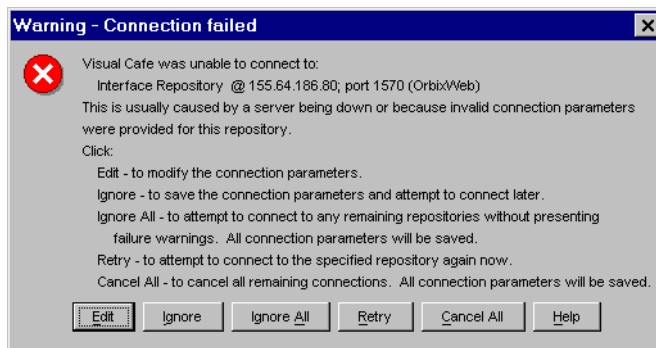
The installation default port number is 1099 for RMI. You can set different default port numbers for RMI and for CORBA in the Distributed Objects tab of the Environment Options dialog box (see “Distributed Object tab” on page A-2).

When you click OK, the wizard attempts to connect to the repository. If the connection fails, the Connection failed dialog box (see “Understanding failed connections” on page 2-19) appears.

The location of the properties file that defines this set of repository connections is set in the Environment Options dialog box, allowing multiple versions of Visual Cafe to read from the same file so that it can be administered centrally.

Understanding failed connections

A connection can fail for various reasons. Perhaps no RMI Registry is running, or no CORBA Interface Repository is running, or the folder path defined in the tree may no longer exist, or a network connection has failed. If the connection fails, the following dialog box appears:



You can respond to this dialog box by clicking the following buttons:

Edit

Closes the warning and opens the Manage Repository Connections dialog box with the failing connection selected. Edit is the default button.

Ignore

Closes the Warning dialog box, keeps the connection in the properties file and displayed in the tree, and sets the icon to “disconnected”.

Ignore All

Attempts to connect to those it can, and does not provide any feedback about those it cannot connect to. Closes the Warning dialog box, keeps the settings of all repositories, and updates the display of all defined repositories.

Retry

Closes the Warning dialog box and attempts to connect to the repository.

Cancel All

Makes no attempt to establish any further connections. Closes the Warning dialog box, and cancels the current connection, keeping all connections in the properties file.

Completing the Select interface page

When you select an interface from the tree, the Interface text box displays its name, and the Servant type text box displays the default servant type for that interface, as determined by the characteristics of the interface selected and the current ORB.

Once you have selected an interface, the Next and Finish buttons are enabled.

If you click Finish the wizard uses the default values for implementation approach, customization, and whether or not to create a client adapter, and generates the template file and supporting files.

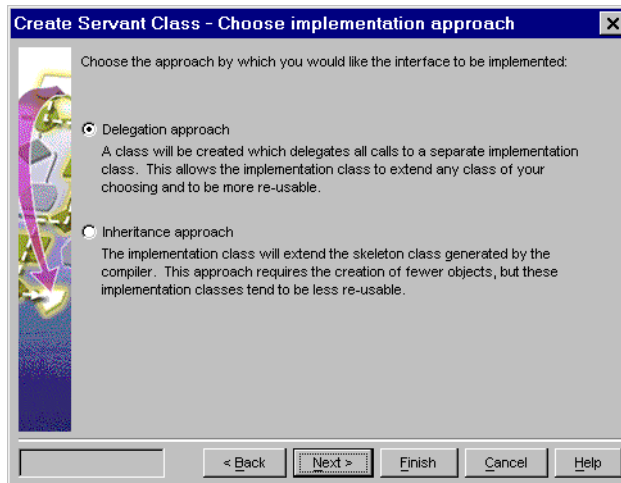
If you click Next, the Choose implementation approach page appears.

Choosing the implementation approach

There are two methods by which a selected interface can be implemented: delegation and inheritance. When **delegation** (the default) is used, a tie class is created which does the binding. The benefit of this approach is that the implementation class can extend any class.

In the **inheritance** approach, a template for an implementation class is generated which inherits from the `ImplBase` class.

On the Choose implementation approach page, you select the approach to use.



The selection made on this page determines the appearance of the Customize implementation class page.

Customizing the implementation class

The Customize implementation class page displays the attributes of the implementation class that the wizard creates. After the wizard completes its task, you must add your code guided by comments in the generated code. The **implementation class** is the one you will add code in order to implement the methods that were defined in the selected interface.

The appearance and functionality of the Customize implementation class page depends on your choice of approach. Before examining the specifics of the two variants of the Customize implementation class page, we must understand the default values. The figures below show the core portion of the delegation and inheritance versions of the Customize implementation class page for an interface named `test02`.

Delegate approach

Inheritance approach

The default values for the Implementation class field depend on the approach you use. The wizard adds a suffix to the name of the interface.

For the delegation approach, the installed suffix is `Delegate`.

For the inheritance approach, the installed suffix is `Impl`.

You can change these default suffixes, as well as the default project name and the default project location, in the Environment Options dialog box (see “Servant Class Wizard Defaults dialog box” on page A-7).

By default, the wizard prepares to put the template implementation class into the current project.

If there is no current project, Visual Cafe creates a new project. If, for example, your Visual Cafe installation folder is `D:\VisualCafe3`, and your interface is `test02`, the default location for the new project is:

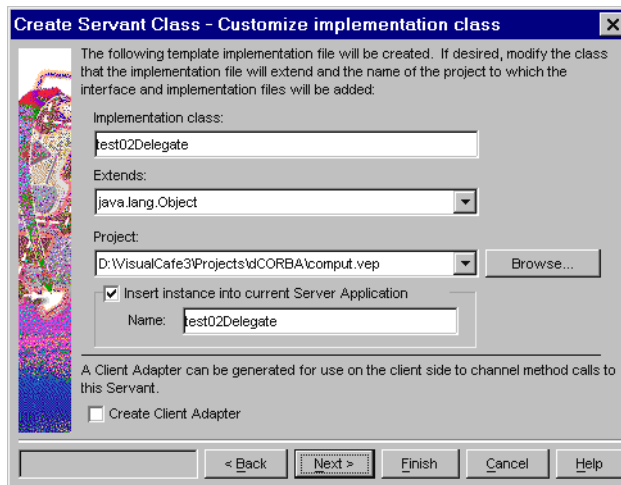
The wizard creates a folder with the same name as your interface, and puts the `.vep` file in there.

The Project combo box contains the name of any open project as well as the “new project” default location shown above.

You can also browse to any existing project—by clicking Browse—and select it, or you can select the new project and Visual Cafe will create it for you.

Using the delegation approach

If you chose the delegation approach, a Customize implementation class page like the one below appears. From this page, you can change the name of the template implementation class and the name of the class it extends from, and specify the project into which it is being created.



To customize the name and location of the implementation class:

- ◆ Type a different name in the Implementation Class text box.

To select the class this implementation class extends:

- ◆ Either select a class from the Extends drop-down list or type the appropriate class name.

The default Implementation base class is the class that the template implementation should extend. The default list contains only `java.lang.Object`. To change or add to the list, you use an entry in the Environment Options dialog box (see “Distributed Object tab” on page A-2).

To select the project to save this implementation class into:

- ◆ Select a project in the Project text field, using the Browse button as necessary.

To insert the implementation class into the current server application whose name is displayed:

- ◆ Select the Insert instance into current server application checkbox.

The Insert instance checkbox is disabled if the current project is not a server application, or if there is no current project (as in the case when you have browsed to a closed project, because the wizard can't tell if it is a server application project).

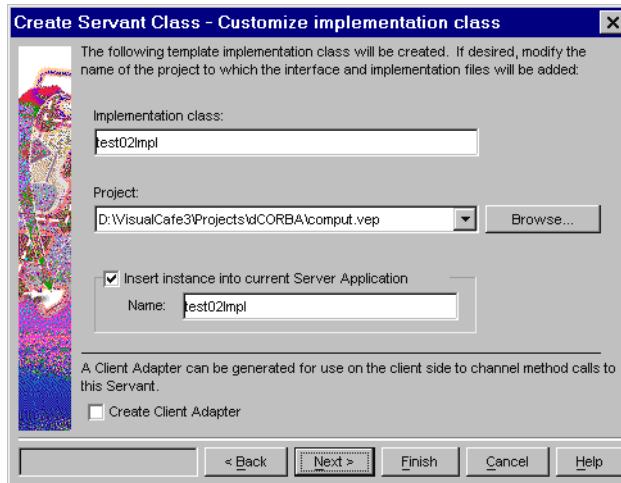
To create a client adapter for the implementation class:

- ◆ Select Create Client Adapter Bean.

If you have chosen to create a client adapter, you will see the Save Client Adapter page next (page 2-31). If not, you will see the Review options page next (page 2-33).

Using the Inheritance approach

If you chose the inheritance approach, the Customize implementation class page shown below appears. From this page, you can change the name of the template implementation file and specify the project into which it is being created.



The idea is to configure the application denoted by Project in order to instantiate an instance of the Implementation class at run time, and bind it to the name specified in Name.

To customize the name and location of the implementation class:

- ◆ Type a different name in the Implementation class text box.

To select the project to save this implementation class into:

- ◆ Select a project in the Project text field, using the Browse button as necessary.

To insert the implementation class into the current server application whose name is displayed:

- ◆ Select the Insert instance into current Server Application checkbox.

Insert instance is disabled if the current project is not a server application, or if there is no current project (as in the case when you have browsed to a

closed project, because the wizard can't tell if it is a server application project).

To create a client adapter for the implementation class:

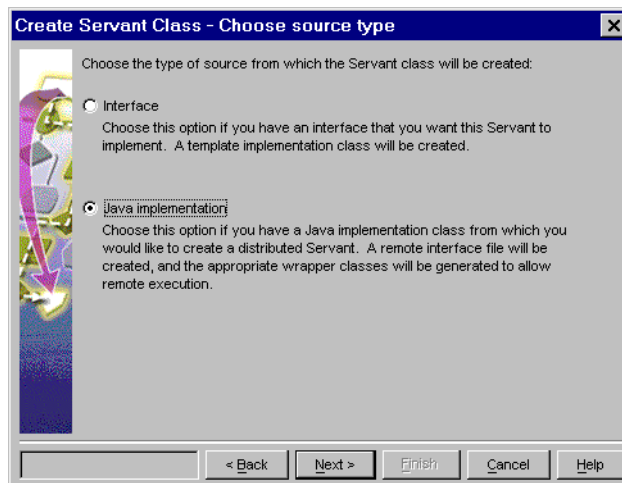
- ◆ Select the Create Client Adapter Bean checkbox.

If you chose to create a client adapter, you will see the Save Client Adapter page next (page 2-31). If not, you will see the Review options page next (page 2-33).

Creating a servant from a Java implementation

“Choosing a source type” on page 2-13 offers the choice between creating a servant from an interface and creating a servant from a Java implementation. This section of the chapter discusses the latter choice.

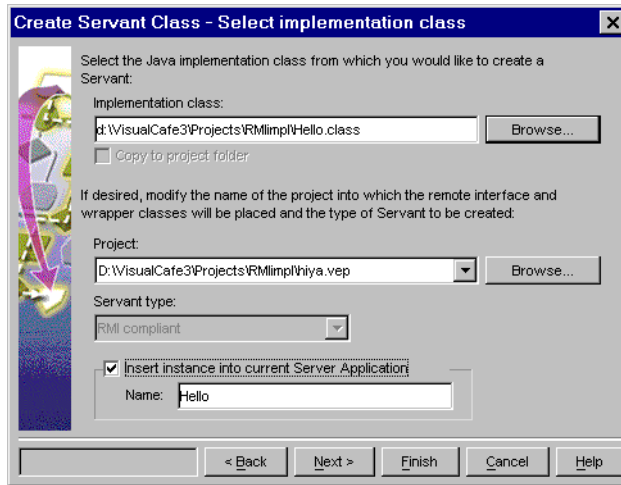
To start creating a servant class from an existing Java implementation, select Java implementation in the Choose source type page and click Next. This takes you to the Select implementation class page. The Servant Class Wizard will produce the appropriate classes and servant adapter, and (if you choose) a client adapter.



Click Next to display the Select implementation class page.

Selecting the implementation class

In the Select implementation class page, you select the implementation class for Visual Cafe to package as a servant class.



Implementation class

The Browse button to the right of this text field opens the Select class dialog box. Navigate to the file you want to use as your implementation class, select it, and click OK. You can also just type the value in.

Project

When you select an implementation class, the Servant Class Wizard displays the name of the project. By default, it puts the source for the implementation file and the other generated files (servant adapter and client adapter) into the current project.

If there is no current project, the wizard creates a new project. The default location is:

```
<install folder>\projects\  
<selected implementation without extension>\  
<selected implementation without extension>.vep
```

You can change the default implementation file name, the default project name, and the default project location in the Environment Options dialog box. See “Servant Class Wizard Defaults dialog box” on page A-7.

Servant type

The Servant type specifies what type of servant you are adapting for remote execution (“remotifying”). The behavior of this UI element depends on the value in the Project field. If that value identifies an open server project, it can be one of three types. The effects are as follows:

RMI: The Servant type field is set to `RMI Compliant` and is read-only.

OrbixWeb: The Servant type field is set to `OrbixWeb` and is read-only.

VisiBroker: The Servant type drop-down list offers two selections: `RMI Compliant` or `OrbixWeb`.

If the value in the Project field does not identify an open server project, the Servant type drop-down list offers two selections: `RMI Compliant` or `OrbixWeb`.

Insert instance into current Server Application

Select this checkbox to specify that you want to insert an instance of the class into the server application.

If the Project field identifies an open server project, the content and enablement of the checkbox depends on the type of the server project and the Servant type field.

If both are RMI, or both are OrbixWeb, this checkbox is enabled. Otherwise, it is disabled.

If the Project field does not identify an open server project, this checkbox is disabled.

Name

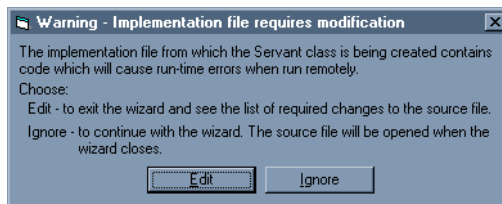
If the Project field identifies an open server project, the content and enablement of the text field depend on the type of the server project and the Servant type field.

If both are RMI, or both are OrbixWeb, the text field is enabled and displays the default name for the server application. Otherwise, it is disabled. If the field is enabled, you can type a name in place of the default.

If the Project field does not identify an open server project, this text field is disabled.

Note: The server project type is determined at its creation by the ORB setting, established in the Distributed Object tab of the Environment Options dialog box (see “Distributed Object tab” on page A-2).

When you click Next, the wizard checks whether the selected implementation file contains method calls that will cause run-time errors when run remotely. If an error occurs, the wizard displays information about the error in its own dialog box:



You can respond to this dialog box by clicking the following buttons:

Edit

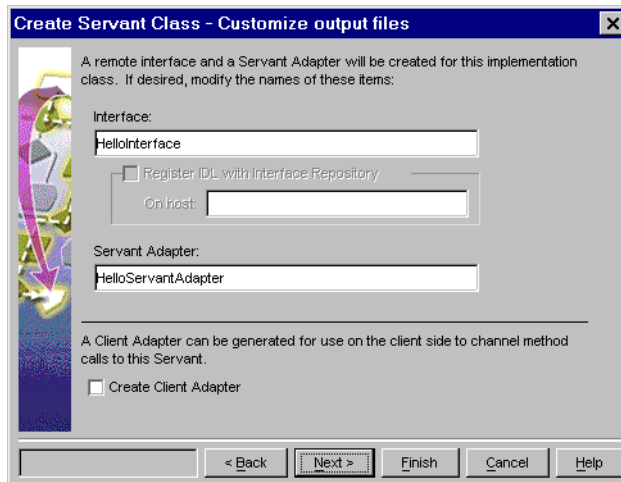
The wizard terminates, and the source file is opened in the current project.

Ignore

The Warning dialog box closes, and you can continue working with the wizard.

Customizing output files

The Customize output files page presents the names of the output files to be created and allows you to modify them. You can change the default naming conventions in the Environment Options dialog box. See “Servant Class Wizard Defaults dialog box” on page A-7.



Interface

The name to be used for the generated interface.

Register IDL with Interface Repository

This checkbox is enabled for a CORBA servant and is disabled for an RMI servant. If selected, it indicates that you want to register the generated IDL for the interface with a CORBA Interface Repository.

On host

If Register IDL with Interface Repository is checked, you must enter the name or IP address of the target host.

Servant Adapter

The name to be used for the generated servant adapter.

Create Client Adapter

By default, the Create Client Adapter checkbox is not selected. If you select it, the default path is:

```
<install folder>\ClientAdapters\<bean name>.jar
```

Saving your client adapter

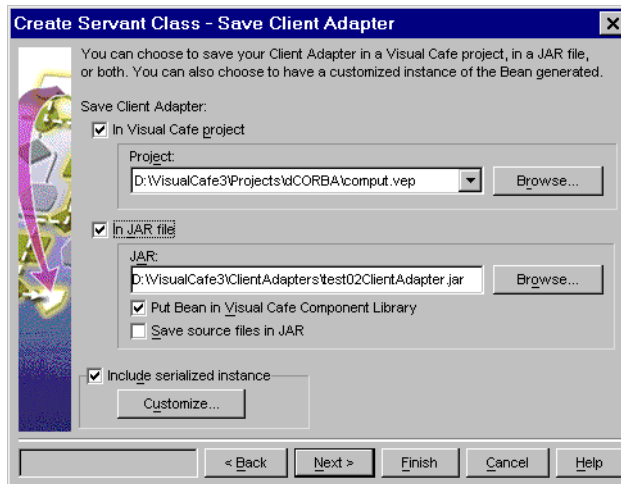
If, on the Customize output files page, you chose to create a client adapter, the Save Client Adapter page appears. Use it to select where you want the client adapter to be saved. The default location is:

<install folder>\ClientAdapters<project name without extension>*.jar*

You can change the location by using the Environment Options dialog box. See “Client Adapter Wizard Defaults dialog box” on page A-6.

Note: The Save Client Adapter page contains the same controls as the In JAR File frame of the Save Options of the Client Adapter Wizard. For full details consult “Saving the client adapter” on page 3-8.

The values in page below represent the interface approach, but the format of the page is the same for interface and implementation.



In Visual Cafe project

If this checkbox is selected, the Project group becomes active and the drop-down list displays the name of the default project.

Project

You can accept the project name that is displayed, select another name from the drop-down list, or use the Browse button to find one that is not displayed.

In JAR file

If this checkbox is selected, the JAR group becomes active, and the JAR text box contains the fully-qualified name of the default JAR file for the client adapter.

JAR

The text box identifies the JAR file. You can accept the default, edit the value, or click Browse to locate the JAR file of your choice.

Put Bean in Visual Cafe Component Library

If this checkbox is selected, the Servant Class Wizard stores the client adapter Bean in the component library (enabling you to fetch it easily by just dragging and dropping).

Save source files in JAR

If this checkbox is selected, the `.java` source files, as well as the compiled `.class` files, are stored in the JAR.

Include serialized instance

You can select to serialize the client adapter so an instance of it is stored—and therefore can be retrieved—with its data and properties intact. Selecting this checkbox makes the Customize button available.

Customize

If you want to modify the properties of the instance of the client adapter that the wizard will create, click Customize to invoke the customizer appropriate to the specified type.

The appropriate customizer will display. (For information about Customizers, see “Customizing a client adapter” on page 3-13.)

Next

If the Put bean in Visual Cafe Component Library checkbox is selected, clicking this button takes you to the Select Component Library folder page.

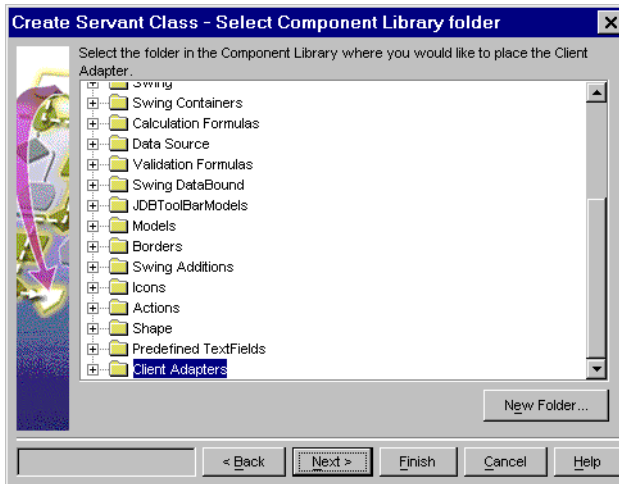
Otherwise, clicking Next takes you to the Review options page (see page 2-33)

Finish

Clicking this button tells the Servant Class Wizard to generate the servant and its related files.

Selecting the component library folder

The default Component Library folder name is `Client Adapters`. If you want to store the client adapter elsewhere, click the folder of your choice, or click `New Folder` (to create a new folder).



This Select Component Library folder page is identical to the Select Component Library folder page in the Client Adapter Wizard (see “Selecting the component library folder” on page 3-11).

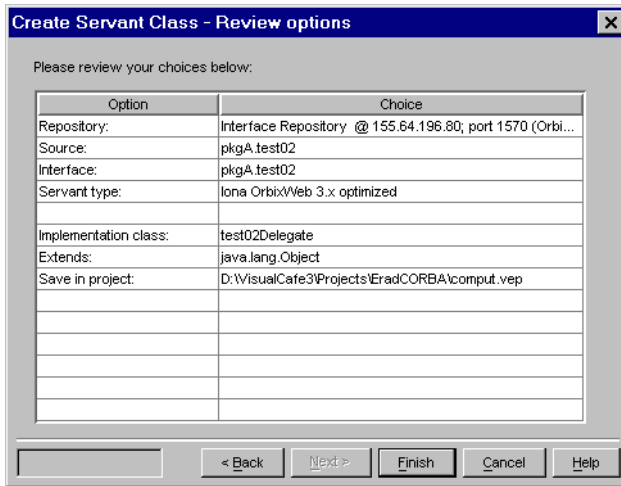
When you are finished with the Select Component Library folder page, click `Next` to go to the Review options page, or `Finish` to tell the Servant Class Wizard to generate the servant and its related files.

Reviewing your selections

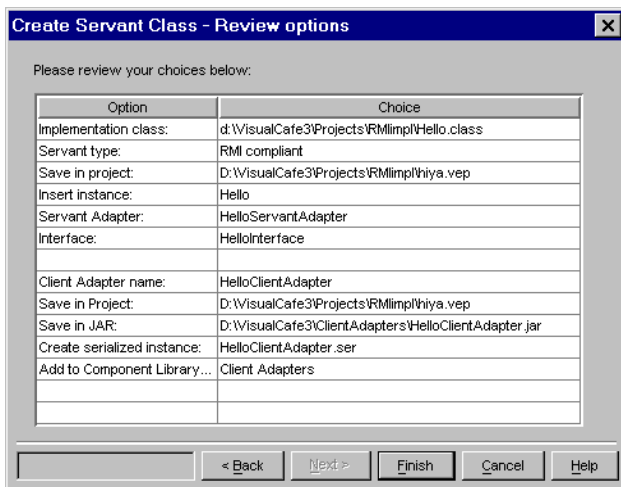
This last page of the Servant Class Wizard allows you to review your chosen options. If you note any that you want to change, use the `Back` button to step back through the wizard pages, make your changes, and

then step forward to this page again using the Next button. The elements of the Review options page are not otherwise editable.

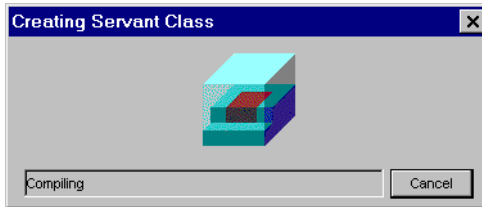
This is an example of the Review options page for the interface approach.



This is an example of the Review options page for the inheritance approach.



When you click Finish on the Review options page, the Creating Servant Class page appears while the project and template implementation file are being created:



Once the Servant Class Wizard finishes generating the servant and its related files, the Wizard terminates. Visual Cafe opens the project to the implementation file.

If you used the interface approach, you need to write the code that implements the methods and compile your servant.

If you used the implementation approach, the servant is ready to use.

You can use the servant in a server application (as described under “About server application projects” on page 2-3).

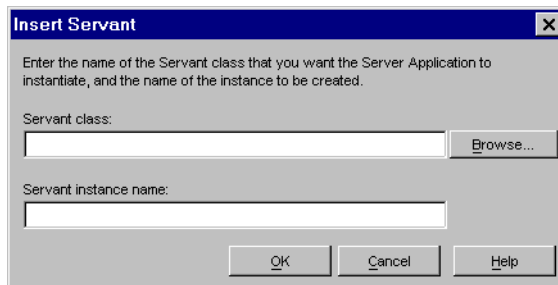
You might see warnings in the Message window about changes to make to the implementation file in order for it to run remotely without errors. You can modify the identified methods using the Source Editor, and compile the servant.

Inserting the servant into a server application

The server application houses and manages one or more servants (server objects). When you create a servant by using the Servant Class Wizard, you can direct the wizard to insert a servant instance into an open server application project.

You might need to insert a servant instance directly into a servant application. This is the case when you did not use the Servant Class Wizard to create the servant class, or when you did use the Servant Class Wizard, but did not insert the servant instance into a server application.

To manually insert a servant, choose the Insert ► Servant Instance menu item. This displays the Insert Servant dialog box:



Servant class

Enter the name for the servant class, or click Browse to open the Select class dialog box, and find the one you want.

Servant instance name

Enter a name for the servant instance or use the default name, which is the name of the servant class.

Executing the servant

Before the servant can be used, it must be instantiated by a server application. Be sure the servant has been recorded in the `ServerApp.properties` file. The Servant Class Wizard records the servant during its processing, but if you are introducing the servant from another source, you will have to construct the `ServerApp.properties` file yourself. An example and discussion of all parameters of the `ServerApp.properties` file can be found beginning on page 2-39.

Having completed the construction of the application, you can run it locally—on the development machine—to verify that it works. Then, you can run it on the remote machine.

Local servant execution

You can run a server application as a separate process on the same system as Visual Cafe. The only relationship between the two is that Visual Cafe is the parent of the server application.

To run the server application locally:

- 1 Be sure the servant has been recorded in the `ServerApp.properties` file.
- 2 Be sure, if your application uses CORBA, that the ORB is running.
- 3 Launch the server application (using Project ► Execute (L)).

The L signifies that the execution will be local.

If your application uses RMI, this starts the RMI Repository if none is active. The server application manages the binding of the name of the servant in the repository or ORB, so that clients (and anyone else) can discover it.

- 4 Your server application opens a new command window and tells you when it launches your servant. Your servant class is now available and visible in an RMI registry which the server application has started.

- 5 You can halt your server application by typing CTRL-C in the command window.

Note: Distributed Development Services must be installed and running on any machine that you hope to find listed in the Remote Execution Setup.

Remote servant execution

You can run a server application as a separate process on the same system as Visual Cafe.

To run the server application remotely:

- 1 Be sure the servant has been recorded in the `ServerApp.properties` file.
- 2 Be sure, if your application uses CORBA, that the ORB is running on both the local system and the remote system.
- 3 Launch the server application (using Project ► Execute (R)).

Visual Cafe starts a process on the remote machine, and your server application is launched in a new command window. You can use the Remote Settings button in the Project options dialog box, and browse to and select any machine running `ddservices`, and run your server application there.)

When you have an executing instance of the servant, you can create a new project (such as AWT template) and create a client adapter with its wizard.

Server Project Templates

To accelerate the development of RMI and of CORBA server applications, Visual Cafe provides two templates: RMI Server and CORBA Server. This is server code that provides reasonable default values that you can modify according to your needs.

The templates provide configuration options like a name server location, RMI registry port, tracing/logging, and so on.

These server applications are not likely to need to be rebuilt when new decisions are made about administration of ORBs, name servers, or other issues arising from using a distributed system in an enterprise.

The server application provides code that locates an existing RMI Registry (creating one if necessary) or connects to an ORB and binds servants to that Registry or ORB.

Interface/Adapter Updates

If you used the Servant Class Wizard to create a servant, you can update the generated interface and wrappers without using the wizard again.

When you right-click a servant object in the Objects tab of the Project window, an Update Servant/Adapters menu displays. Choosing this menu item regenerates the interface file, the servant adapter, and the client adapter using the same options defined in the wizard.

Server application properties

The `ServerApp.properties` file contains the data that your otherwise generic server application needs to instantiate and properly register your servants for use by your client programs. By understanding, inspecting, and even editing the `ServerApp.properties` file, you can leverage the Symantec server application framework without modifying a line of server code. The following sections include descriptions of property settings in the file you may need to understand. They include:

- ◆ Global application property settings, some of which are specific to RMI or to CORBA or to certain implementations of CORBA

- ◆ Servant record application properties, some of which are specific to CORBA

Here is an example of a `ServerApp.properties` file:

```
{
  UseIIOP = "true";
  ApplicationUsesVendor = "Iona OrbixWeb 3.1";
  COSNamingServiceServer = "NS";
  ApplicationServants =
  (
    {
      NamingPath = "EchoDelegate";
      ServantName = "EchoDelegate";
      ExportIOR = "false";
      Classname = "acme.corba.EchoDelegate";
      IORFilePath = "EchoDelegate.ior";
      ExportNamingService = "false";
      TIENAME = "acme.corba._tie_Echo";
    },
  );
  IIOPPort = "1570";
  NamingRepositoryPath = "";
  NamingHost = "localhost";
  ServerTimeout = "-1";
  ServerAppPropertiesFormat = "1.0";
  ApplicationUsesORBOnHost = "localhost";
  ApplicationUsesORBOnPort = "1570";
  NamingPort = "1570";
  NamingIP = "";
  ServerName = "ServerApp";
}
```

Global application property settings

These properties are used in all application servers.

```
ApplicationUsesVendor = "productName"
```

The type of the server application (RMI, OrbixWeb, VisiBroker) is recorded in the properties file for informational purposes only.

Note: This value does not control the actual type of server application. This is actually specified by the base class of the server application's main class.

An example is `ServerApp` inheriting from `IonaServerAppAdapter`. This field denotes the product the server application was targeted for

at the time of creation, such as "Sun RMI 1.1.x" or "Inprise VisiBroker for Java 3.x".

`ServerAppPropertiesFormat = "1.0"`

Used for maintaining backward and/or forward compatibility in properties file. Also used to designate this properties file for inspection and modification by the Visual Cafe Insert Servant Instance functionality.

`ApplicationServants = (servantRecord1, servantRecord2, ...)`

A list of servant records, each described by a sub-properties expression that looks like:

```
{
key1 = "value1";
key2 = "value2";
}
```

Each servant record pertains to an instance of a servant class. The actual keys and values for servant records are documented in "Servant record application properties" starting on page 2-44.

RMI global application property settings

These properties are used only in RMI application servers.

`ApplicationCreatesRMIRegistry = "true"`

If the value is `true` (the default), the server application will attempt to create its own RMI registry in its process. If the creation attempt fails (usually because a registry has already been created), the server application tries to use the previously created registry.

If the value is `false`, the server application only attempts to locate a registry.

`ApplicationUsesRMIRegistryOnPort = "1099"`

The number specified is the port number that the server application uses.

`CodebaseURL = ""`

If no codebase URL was set on the command line to the Java VM, you can use this property to fill in the `java.rmi.server.codebase` property. The value of the codebase URL must end with `/`. See the RMI documentation for more detail.

```
AutoGenerateCodebaseURL = "true";
```

If this property is `true` and the `CodebaseURL` property is empty, then an RMI server application automatically generates a file URL based on the folder of the executed RMI server application. The generated URL is then saved as the `CodebaseURL` and the system property `java.rmi.server.codebase` is set to this value.

```
UseCodebaseOnly = "false";
```

This property can be set to fill in the `java.rmi.server.useCodebaseOnly` property if it is not set on the command line to the Java VM. See the RMI documentation for more detail.

```
LogCalls = "false";
```

This property can be set to fill in the `java.rmi.server.logCalls` property if it is not set on the command line to the Java VM. See the RMI documentation for more detail.

CORBA global application property settings

These properties are used only in CORBA application servers.

```
ApplicationUsesORBOnPort = "14000"
```

```
ApplicationUsesORBOnHost = "localhost"
```

Host machine and port of the ORB. By default, this is the local machine and the port defined for the current ORB product specified in the Environment Options dialog box. See “Manage CORBA ORBs dialog box” on page A-4.

```
COSNamingServiceServer = "CosNaming";
```

The name of the Naming Service server. In VisiBroker, this property corresponds to the `ORBservices` property. In OrbixWeb, this property corresponds to the `IT_NAMES_SERVER` property.

Global application property settings (Visibroker-specific)

These properties are used only in VisiBroker-specific application servers.

```
ORBagentHost = ""
```

The value of this property is set to the Visibroker property `"ORBagentHost"` during the initialization of the ORB. The default value is empty.

ORBagentPort="14000"

The value of this property is set to the Visibroker property "ORBagentPort" during the initialization of the ORB. The default value is 14000.

NamingRootName = "root"

The virtual root of the Visibroker naming service.

Global application property settings (OrbixWeb-specific)

The following properties have equivalents in the OrbixWeb configuration file. They are provided here so that each OrbixWeb server application can be configured independently from the global configuration file. Refer to the Iona OrbixWeb documentation for details.

LocalHostName = ""

The value of this property is set to the OrbixWeb property "IT_LOCAL_HOSTNAME" during the initialization of the ORB. The default value is empty, so the value defined in the OrbixWeb configuration is used.

OrbixdPort = "1570"

The value of this property is set to the value of the OrbixWeb property "IT_ORBIXD_PORT" (Web daemon port) during the initialization of the ORB. The default value is 1570.

IIOPPort = "1571"	Iona IT_ORBIX_PORT property
NamingHost = "localhost";	Iona IT_NS_HOSTNAME property
NamingRepositoryPath = "";	Iona IT_NAMES_REPOSITORY_PATH
ServerTimeout = "-1";	Iona IT_IMPL_IS_READY_TIMEOUT property
NamingPort = "1571";	Iona IT_NS_PORT property
NamingIP = "127.0.0.1";	Iona IT_NS_IP_ADDR property
UseIIOP = "true";	Iona IT_BIND_USING_IIOP property

```
ServerName = "ServerApp"
```

The name of the server application (This is not the same as the name the servants get from the servant records, which is an Iona “marker”).

Important: This name must be registered with either the OrbixWeb Implementation Repository or the `putit` command—unless the orbix daemon (`orbixdj`) is run with the `-u` switch.

Before deployment of the application, the server name should be edited so that it is unique to each Implementation Repository.

Servant record application properties

These properties are used in binding a servant.

```
Classname = "BarServantAdapter"
```

The class that is instantiated to obtain the servant object. The class needs to have a public no-argument constructor. In the CORBA case, see the `TIName` property below.

```
ServantName = "object instance name"
```

The name under which the servant is bound with the RMI Registry or an ORB.

Servant record application properties (CORBA-specific)

These properties are used in binding a CORBA servant.

```
TIName = "_tie_BarInterface"
```

The class of the “tie” object that is created to tie the CORBA requests to the actual servant object.

```
ExportNamingService = "false"
```

Controls whether the object is exported to a naming service or not. Defaults to the setting given in the Environment Options dialog box. For example, `false` means vendor-specific or IOR (interoperable object reference) binding is chosen; `true` means Naming Service binding is chosen. Servants are always accessible through a vendor-specific binding.

`NamingPath = "name1.name2.name3"`

The dot-separated path of names used as the object's name when binding the object to the naming service. Only used when `ExportNamingService` is set to `true`.

`ExportIOR = "false"`

Controls whether the object's IOR string is written to a file or not. Defaults to the setting given in the Environment Options dialog box (that is `false` if vendor-specific or Naming Service binding is chosen, `true` if IOR binding). Servants are always accessible through a vendor-specific binding.

`IORFilePath = "ObjectName.ior"`

The file that is written to with the IOR string upon launch. Note that each time a servant is created on launch a different IOR string is written to this file. Only used when `ExportIOR` is set as `true`.

Special Requirements on Running the RMI Registry

Both the Servant Class Wizard and the Client Adapter Wizard can browse an RMI registry for servants. There are two ways to start the RMI registry. They are:

- ◆ Run the RMI Registry from the command line.

```
rmiregistry [port]
```

The `rmiregistry.exe` executable is located in `\Java\Bin` in your installation folder, and `port` is the port number used by the RMI registry. The port number defaults to 1099 if no value is specified.

- ◆ Start the RMI registry inside your server application code by setting the property `ApplicationCreatesRMIRegistry` to `true`.

Whether you start the RMI registry in a different process or you start the RMI registry inside your server application code, you need to set the property `java.rmi.server.codebase` to a valid URL that Visual Cafe can locate and access. This codebase URL is used by the Client Adapter Wizard to introspect RMI server objects that are bound to the RMI registry. The URL is also used to download class files such as the stub and interface classes and the user-defined classes that are passed as parameter types in remote methods.

If you do not specify a codebase URL in the RMI Registry or in the server code that starts the RMI Registry, then the Client Adapter Wizard does not display this implementation (server) class in its Select Servant page if you select the specified RMI Registry as a source. If the URL is either invalid or inaccessible, or the class files are not found there, the Client Adapter Wizard does not display the implementation (server) classes that are bound to the RMI registry.

If you are starting the RMI registry as a separate process, you must establish the codebase. The `ServerApp.properties` file includes a `CodebaseURL` setting. The value of the codebase URL must end with `/`.

Now that you have seen all the parts of the Servant Class Wizard, you might just try exercising it a bit by walking through the steps of just one of the ways you can generate a servant — see “A walk-through of the Servant Class Wizard” on page B-1. No screen shots are provided—just a quick list of the steps to follow.

Developing Client Adapters

The Client Adapter Wizard helps you to build a client adapter (proxy) for a servant (server object). Using the Client Adapter Wizard lets you stay focused on developing the client-side components as you create a client adapter for use in your client-side source code.

This chapter provides:

- ◆ an overview of the Client Adapter Wizard
- ◆ a walk through of the Client Adapter Wizard, discussing each page and what it does
- ◆ a discussion of customizing the client adapter

Overview

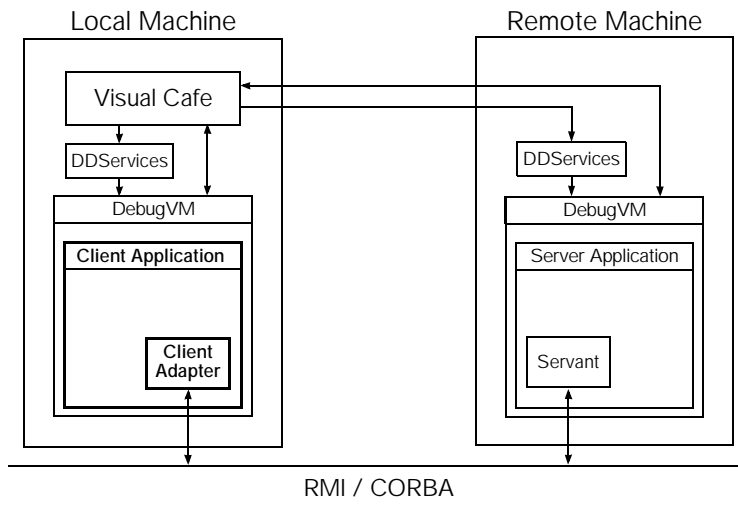
The Client Adapter Wizard offers you a series of pages on which you make the selections. The selections you can specify are:

- 1 The servant you want to communicate with, chosen from a tree of RMI Registries, CORBA Interface Repositories, and folders of files.
- 2 The name for your client adapter.
- 3 The destination for your client adapter: a Visual Cafe project, a JAR file, or both.
- 4 If a JAR file is a destination, you can choose:
 - ❖ to save the client adapter in the Visual Cafe Component Library, and if so, what folder to save it in
 - ❖ to save the source of the adapter in the JAR as well as the class file

- ❖ to include a serialized instance of the client adapter
- ❖ to customize the client adapter for a specific implementation

The Client Adapter Wizard then creates a client adapter that mediates the communication with the servant. All the business of establishing and maintaining the communication is neatly encapsulated in the client adapter—and you just use it without having to manage the details. Once you have set the properties of the client adapter to point to the desired servant, you can make method calls to the client adapter just as you would make method calls to any local class. The client adapter forwards the calls to the servant for execution.

If you add the client adapter to your Component Library, you can drop it onto a project, modify its properties, and connect it to other components with the Interaction Wizard. The following illustration shows the client adapter interacting with the servant through RMI or CORBA.

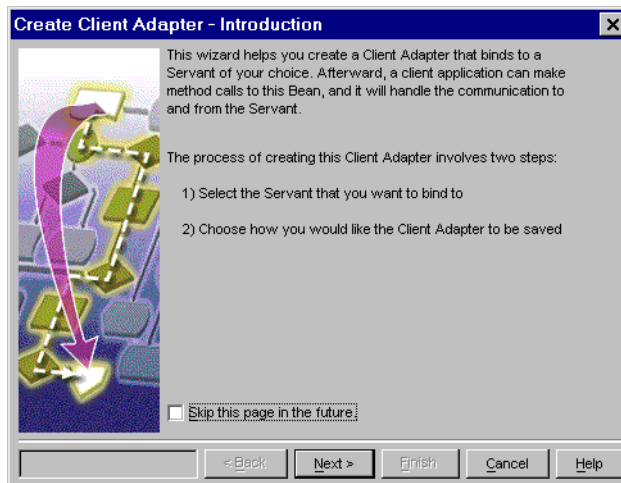


Creating a client adapter

To create a client adapter—a Java Bean that can communicate with a CORBA or RMI object—use the Client Adapter Wizard.

Introduction page

Start the Client Adapter Wizard by selecting File ► New Client Adapter. The Introduction page appears.



Once you have read and understood this information, you can select the Skip this page in the future checkbox to instruct the Client Adapter Wizard not to display the Introduction page every time you use the wizard. To reset this option, you can click the Back button on the Select Servant page.

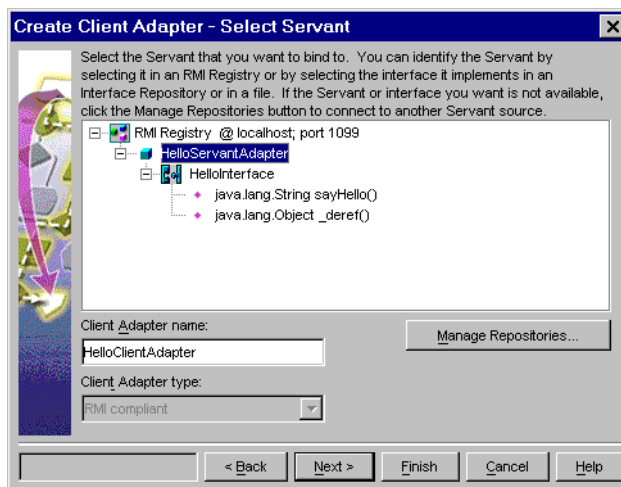
When you have finished with the Introduction page, click Next to proceed to the Select Servant page.

Selecting the servant

Because the Client Adapter Wizard creates a client adapter for a specific interface, you must select an interface (or an RMI servant adapter) before you can generate the client adapter. In the Select Servant page, a tree view displays any repositories that are defined.

This information is recorded in a connection properties file.

Note: The default properties file is `connection.properties`, in the `config` folder in the installation folder. You can establish the location and name of the connection properties file in the Environment Options dialog box, so that multiple developers can use the same set of connections. See “Distributed Object tab” on page A-2.



If no repositories are defined, the Client Adapter Wizard displays the Manage Repository Connections dialog box, and immediately overlays it with a Select New Repository Type dialog box. This dialog box offers you a list of repository types to choose from. For more information, see “Managing repository connections” on page 2-17.

Using the Select Servant tree view

The Select Servant tree view enables you to browse through the defined repositories and to view what is available in them. A **source** is an item

(such as a servant or its interface) that can be used to create a client adapter or a servant.

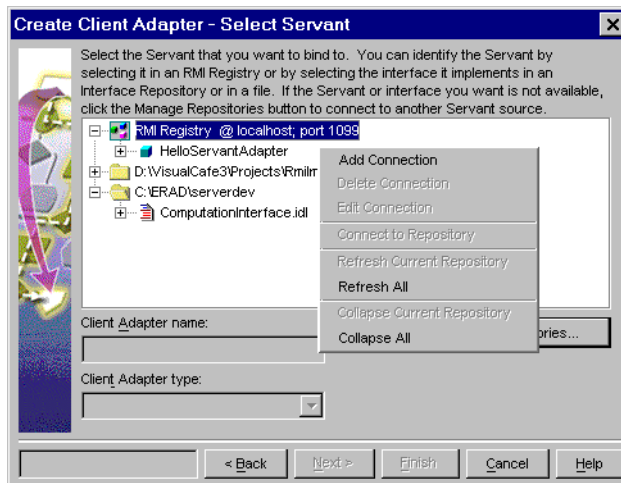
Select the source that contains the interface or object you want to communicate with. Here are the valid types of files that are displayed for a file folder repository:

- ◆ CORBA IDL file
- ◆ CORBA interface class file
- ◆ RMI interface class file (for creating servants only)
- ◆ RMI servant adapter class file
- ◆ RMI stub class file

When you expand a node of the tree, the Client Adapter Wizard attempts to open a connection to the sources associated with that node. The icons show whether a repository is connected, disconnected, in process of connecting, or disabled/invalid. For a complete list of the icons, see “The repository connections tree” on page 2-15.

If you need to connect to a repository that has not yet been defined and so does not appear in the tree, click the Manage Repositories button to open the Manage Repository Connections dialog box.

Another way to add a connection is to right-click in the tree area. A pop-up menu appears, which allows you to perform a variety of tasks.



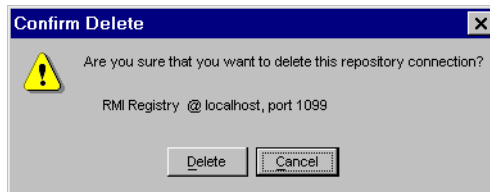
The commands in the Select Servant context menu are:

Add Connection

Opens the Manage Repository Connections dialog box, then opens the Select New Repository Type dialog box on top of it.

Delete Connection

Removes the selected repository from the tree and from the `Connection.properties` file. Displays the following confirmation dialog box:



Click Delete to confirm the deletion, or click Cancel to cancel the deletion.

Edit Connection

Opens the Manage Repository Connections dialog box at the selected source.

Connect to Repository

Connects to a disconnected repository and displays its contents (equivalent to expanding the tree view).

Refresh Current Repository

Updates the display of sources within the repository in which the current selection resides.

Refresh All

Updates the display of sources for all repositories.

Collapse Current Repository

Collapses the entire repository node in which the current selection resides.

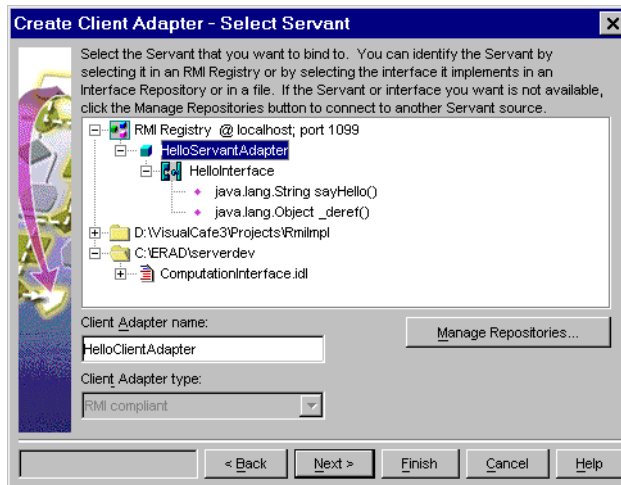
Collapse All

Collapses all nodes.

In the tree controls in both the Client Adapter Wizard and the Servant Class Wizard, all repositories defined in the Manage Repository Connections dialog box are shown as top-level nodes. Repositories that cannot be used in the current wizard session are disabled. For example, if the currently-selected ORB definition's product type is OrbixWeb, any VisiBroker interface repositories that are defined appear in the tree view as disabled.

The items contained under the repository nodes are filtered so that only containers and valid sources are visible. For example, items that are appropriate only for the Client Adapter Wizard are dimmed when the tree is viewed in the Servant Class Wizard.

When you select an interface or a servant adapter, the wizard enters the default name for the client adapter in the Client Adapter name field and enters the default type into the Client Adapter type field:



The default naming convention for a client adapter is:

*sourcename*ClientAdapter

For example, if the source name is `audit`, the default client adapter name is `auditClientAdapter`. You can change the default suffix in the Distributed Object tab of the Environment Options dialog box (see "Distributed Object tab" on page A-2).

The client adapter types are determined by the source selected. The drop-down list shows the valid type options. The Environment Options dialog box contains settings to define the current CORBA ORB.

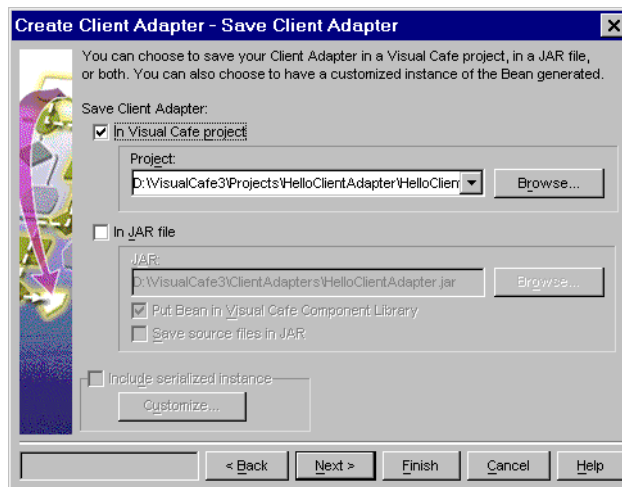
The Next and Finish buttons are not available until you have selected a servant/interface. Once this is done, you can select Finish at any time, and the default settings are used to set any properties remaining.

Saving the client adapter

As you are creating the client adapter, you can save it in a JAR file, in a Visual Cafe project, or in both. The Next button is available when at least one of these options is selected and its parameters are set.

Saving the client adapter in the project

The default option is to save the client adapter in the current project as the following image shows.



To save the client adapter in a project:

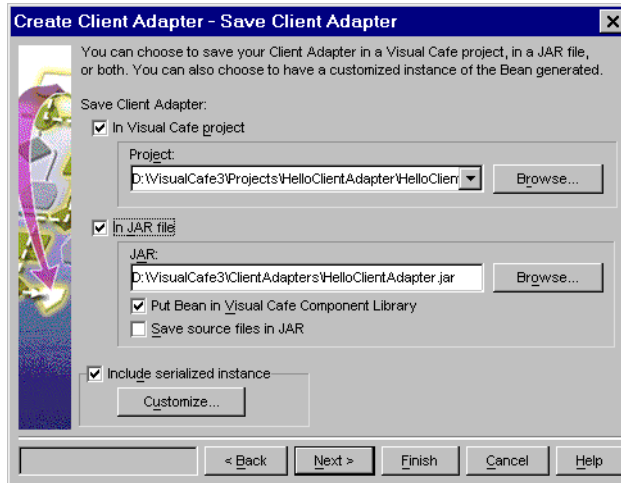
- ◆ Select In Visual Cafe project.

If you choose to save the client adapter in a project, the default file name of the project is:

ClientAdaptername.vcp

Saving the client adapter in a JAR file

If you also select In JAR file, the page appears as follows:



Selecting In JAR file automatically selects Include serialized instance, so if you do not want a serialized instance, deselect that option.

To save the client adapter in a JAR file:

- ◆ Select the file into which you want to save the client adapter, using the Browse button to locate a file other than the displayed one.

To also save the client adapter in the Component Library:

- ◆ Select Put Bean in Visual Cafe Component Library.

To also save the client adapter source file in the JAR:

- ◆ Select Save source files in JAR.

The default name for the JAR is:

ClientAdaptername.jar

The default path for the JAR is:

installationfolder\ClientAdapters

After its creation, the JAR file contains:

- ◆ one or more client adapters
- ◆ one or more `BeanInfo` files
- ◆ any files generated by the IDL or `rmic` compilers
- ◆ one or more stub files
- ◆ any interface file that you selected
- ◆ if you selected Include serialized instance, a `.ser` file defining an instance of the client adapter (described below)

The default folders are created when Visual Cafe is installed, so that the current folder can be displayed in the Browse dialog box. You can change these default paths through the Environment Options dialog box (see “Client Adapter Wizard Defaults dialog box” on page A-6).

To include a serialized instance of the client adapter in the JAR:

- ◆ Make sure that Include serialized instance checkbox is selected.

To modify the properties of the instance:

- a Click Customize.
The appropriate customizer displays.
- b Modify the properties as needed.
- c Click OK to finish with the customizer.
See “Customizing a client adapter” on page 3-13 for details.

Selecting the component library folder

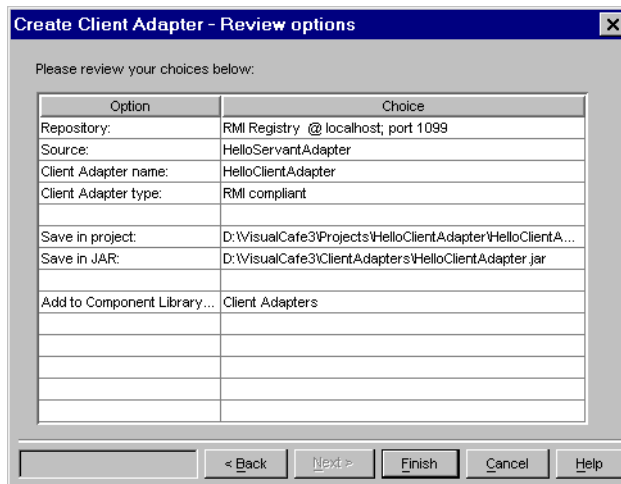
Saving the client adapter in the Component Library enables you to re-use it in other projects. If you chose to put the client adapter in the Component Library, you now see a page that allows you to select a folder in the Component Library. You may choose instead to add a new folder.



The default selection is the `Client Adapters` folder. If you click `New Folder`, a new untitled folder is added to the tree and selected, and a dialog box is opened so that you can easily type a name.

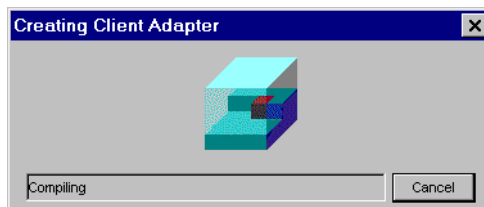
Reviewing your selections

In the final page of the Client Adapter Wizard, you can review your choices. The following image shows that the client adapter is to be saved in a project and a JAR file, but no customization is being performed.



If you want to change any of your selections, click Back to step back through the wizard pages to make your changes, then step forward to this page again using the Next button, or click Finish at any appropriate point along the way.

When you click Finish, Visual Cafe generates and compiles the client adapter and all other required files. While this is happening, it displays an animated dialog box:



If you requested a serialized instance of the client adapter for the JAR, then after the wizard has generated the client adapter class, Visual Cafe creates an instance of the object. The connection parameters used in generating the client adapter, such as Host name and Port of RMI Registry, are used to

define the properties of the client adapter instance. This instance is then serialized and included in the JAR.

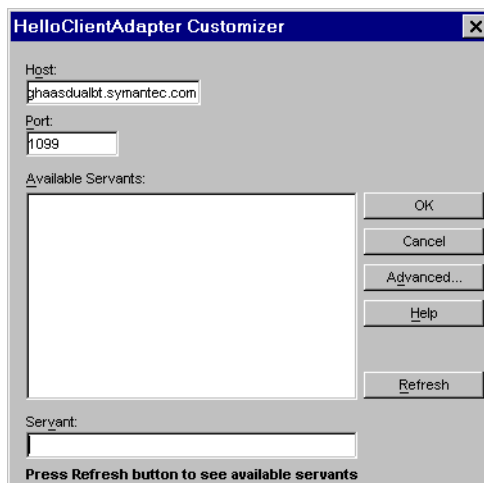
Customizing a client adapter

Customizers enable you to modify the properties of the client adapter Bean. The properties are also available through the Bean's property list to allow experienced users to quickly change individual values. The available properties vary according to the type of client adapter—according to which vendor and which protocol it is specialized for. The three variations are:

- ◆ Customizing RMI client adapters
- ◆ Customizing OrbixWeb client adapters
- ◆ Customizing VisiBroker client adapters

Customizing RMI client adapters

While you are creating a client adapter for an RMI implementation, you can use a customizer to tailor specific instances of it. When you click **Customize** on the **Save Client Adapter** page, the **Customizer** page appears. Here, as an example, is the **Customizer** page for the client adapter named `HelloClientAdapter`:



The elements of the Customizer page and their purposes are:

Host

Specifies the host name (or the IP address) of the machine on which the RMI Registry is running.

Port

Specifies the number of the port through which the RMI Registry communicates.

Available Servants

Displays a list of the names of servants on the specified host machine and port. When you select one, that name appears in the Servant field. If the list is empty, click Refresh.

Servant

Specifies the name of the servant to which this client adapter attaches. Either select it from the list of available servants, or type it.

Advanced

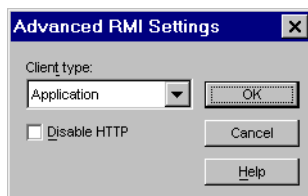
Displays the Advanced RMI Settings dialog box.

Refresh

Populates the Available Servants text area.

Advanced RMI settings

This dialog box lets you make advanced settings.



The elements of the Advanced RMI Settings page and their purposes are:

Client type

Defines the type of client, either application or applet, in which the client adapter will be used.

Disable HTTP

Selecting this option prevents the client adapter from attempting to communicate using HTTP when RMI communication fails.

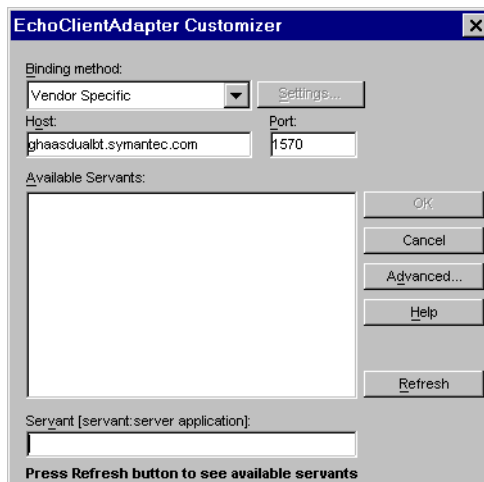
Customizing OrbixWeb client adapters

While you are creating a client adapter for an OrbixWeb implementation, you can use a customizer to tailor specific instances of it. When you click **Customize** on the **Save Client Adapter** page, the **Customizer** page appears. The first field on the page is **Binding method**. The content of the remainder of the page depends on which of three possible values **Binding method** has:

- ◆ Vendor Specific
- ◆ Inter-operable ORB Reference
- ◆ Naming Service

OrbixWeb vendor-specific binding

Here is an example of a **Customizer** page for the client adapter named `EchoClientAdapter` using vendor-specific binding:



The remaining elements of the **Customizer** page and their purposes are:

Settings

This button is not enabled for Orbix vendor-specific binding or Naming Service binding. If you select Inter-operable ORB Reference binding, the **Settings** button becomes enabled, and clicking it opens

the Binding Methods dialog box to the IOR Binding Method (see “CORBA Binding settings” on page 3-17).

Host

The name (or the IP address) of the machine on which `orbixdj` is running.

Port

The port on which the OrbixWeb ORB is running (the default is 1570).

Available Servants

Displays a list of the names of servants on the specified ORB. When you select one, that name appears in the Servant field. If the list is empty, click Refresh. All servers registered in the implementation repository (running or not) are listed. In OrbixWeb, active servants are distinguished by their leading colon (:).

Servant

Specifies the marker name then the server name to which this client adapter attaches. For information about markers, consult the OrbixWeb documentation.

Advanced

Displays the Advanced OrbixWeb Settings dialog box. See page 3-17.

Refresh

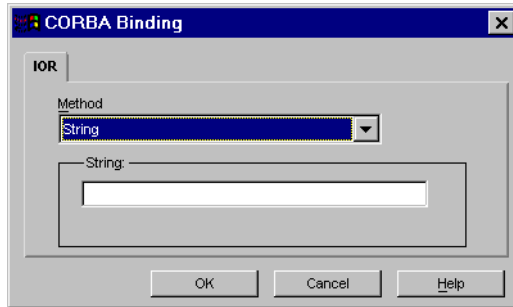
Populates the Available Servants text area.

OK

This button is only enabled when the Servant field is not empty. Click OK to close the dialog box.

CORBA Binding settings

This page offers one tab: IOR (Inter-operable ORB Reference) for binding.



The IOR tab shows you what the parameters are for each method (String, File, or Applet).

String

The IOR string for the servant. For OrbixWeb, this is set by the Wizard.

File

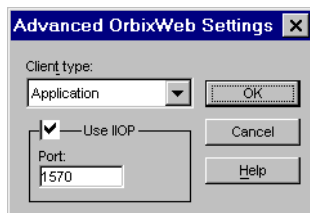
The name of the text file that contains the IOR string for the servant.

Applet

The IOR string that is used in the `Applet Tag` parameter. This option is visible only if the Client type is set to `Applet` in the Advanced OrbixWeb Settings dialog box.

Advanced OrbixWeb settings

This dialog box lets you make advanced settings for an ORB of the type you have selected. Here, as an example, is the dialog box for OrbixWeb:



The elements of the Advanced OrbixWeb Settings page and their purposes are:

Client type

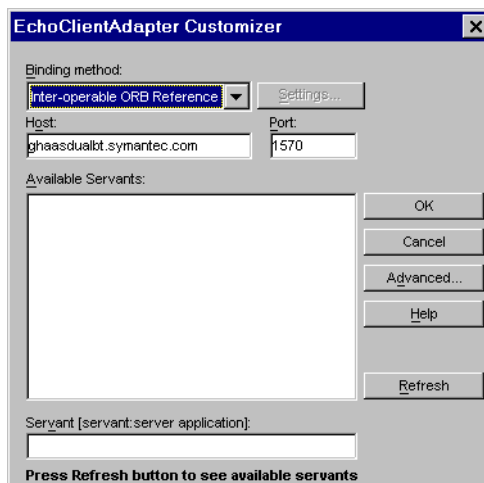
Defines the type of client, either application or applet, in which the client adapter will be used.

Use IIOP

This option governs the choice of port to be used for communication. If it is enabled, the IIOP port (as displayed in Port) is used; if it is disabled, the ORB port is used.

Inter-operable ORB reference binding

Here is an example of a Customizer page for the client adapter named `EchoClientAdapter` using Inter-operable ORB reference binding:



The remaining usable elements of the Customizer page and their purposes are:

Settings

When a servant has been selected from the Available Servants list, the Settings button becomes enabled.

Host

On OrbixWeb the name (or the IP address) of the machine on which `orbixdj` is running.

Port

The port on which the OrbixWeb ORB is running (the default is 1570).

Available Servants

Displays a list of the names of servants on the specified ORB. When you select one, that name appears in the Servant field. If the list is empty, click Refresh. All servers registered in the implementation repository (running or not) are listed.

Servant

Specifies the marker name then the server name of the servant to which this client adapter attaches.

Advanced

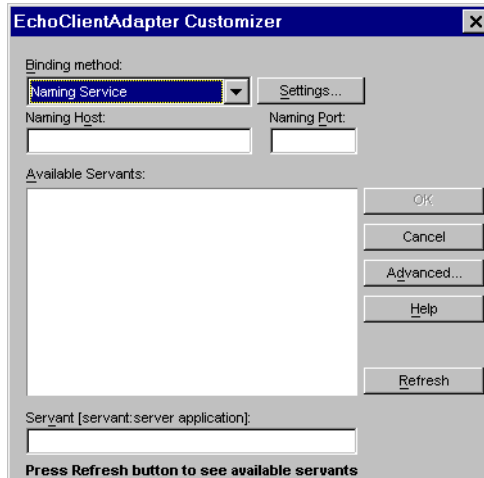
Displays the Advanced OrbixWeb Settings dialog box. See page 3-17.

Refresh

Populates the Available Servants text area.

Naming Service binding

Here is an example of a Customizer page for the client adapter named `EchoClientAdapter` using Naming Service binding:



The remaining usable elements of the Customizer page and their purposes are:

Settings

The Settings button is always enabled for Naming Service binding.

Naming Host

Displays the name (or the IP address) of the machine on which the Naming Service is running.

Naming Port

Displays the port on which the Naming Service is running (OrbixWeb default is 1570).

Available Servants

Displays the naming paths for the servants that are registered to the Naming Service.

Servant

Specifies the marker name then the server name of the servant to which this client adapter attaches.

Advanced

Displays the Advanced OrbixWeb Settings dialog box. See page 3-17.

Refresh

Populates the Available Servants text area.

Customizing VisiBroker client adapters

While you are creating a client adapter for a VisiBroker implementation, you can use a customizer to tailor specific instances of it. When you click **Customize** on the **Save Client Adapter** page, the **Customizer** page appears.

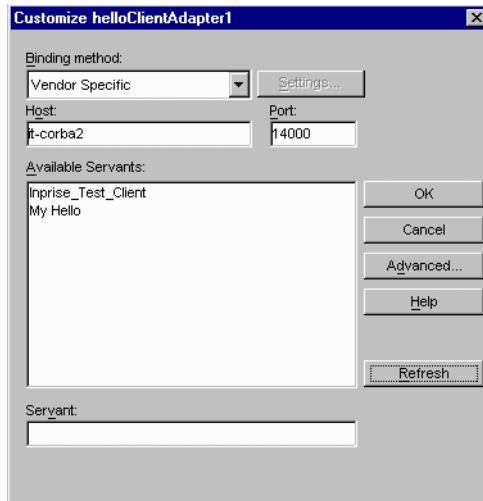
Note: This process is very similar to the procedure for OrbixWeb customization.

The first field on the page is **Binding method**. The content of the remainder of the page depends on which of three possible values **Binding method** has:

- ◆ Vendor Specific
- ◆ Inter-operable ORB Reference
- ◆ Naming Service

VisiBroker vendor-specific binding

Here is an example of a Customizer page for the client adapter named `EchoClientAdapter` using vendor-specific binding:



The remaining usable elements of the Customizer page and their purposes are:

Host

The name (or the IP address) of the machine on which the object activation daemon is running.

Port

The port on which the VisiBroker ORB is running (the default is 14000).

Available Servants

Displays a list of the names of servants on the specified ORB. When you select one, that name appears in the Servant field. If the list is empty, click Refresh.

In Visibroker, only the servers registered with the object activation daemon are listed.

Servant

Specifies the name of the servant to which this client adapter attaches.

Advanced

Displays the Advanced VisiBroker Settings dialog box. See page 3-23.

Refresh

Populates the Available Servants text area.

Settings

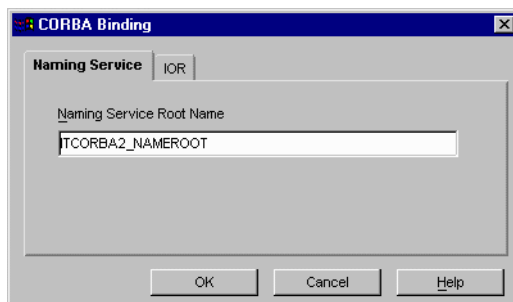
When a servant in the Available Servants list is selected, the Settings button becomes enabled.

Clicking the Settings button opens the Binding Methods dialog box.

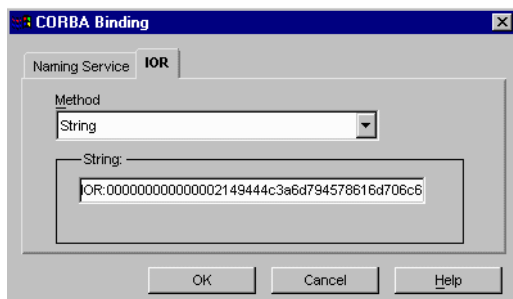
CORBA Binding settings

This page offers two tabs: Naming Service and IOR (Inter-operable ORB Reference).

On the Naming Service tab, you can enter the Naming Service Root Name.



On the IOR tab, you can select a method from the Method drop-down list. The possible methods are String, File, and Applet. The dialog box then displays its value in the corresponding field. The following illustration shows a String value.



The possible values for each method follow.

String

The IOR string for the servant. You can type it in or let the wizard generate it for you.

File

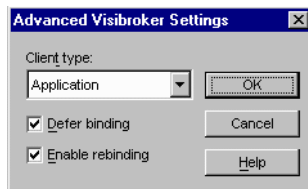
The name of the text file that contains the IOR string for the servant.

Applet

The IOR string that is used in the `Applet Tag` parameter. This option is visible only if the Client type is set to Applet in the Advanced VisiBroker Settings dialog box.

Advanced VisiBroker settings

This dialog box lets you make advanced settings for a selected ORB type. Here, as an example, is the dialog box for VisiBroker:



The elements of the Advanced VisiBroker Settings page and their purposes are:

Client type

Defines the type of client, either application or applet, in which the client adapter will be used.

Defer Binding

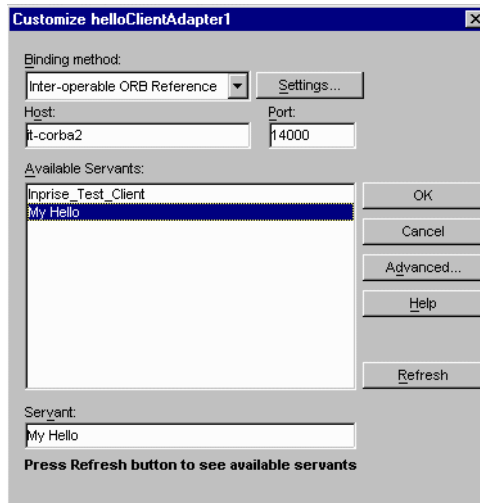
Only binds when a method call is made. Consult vendor documentation for details.

Enable Rebind

Automatically disconnects and reconnects on method calls. Consult vendor documentation for details.

Inter-operable ORB reference binding

Here is an example of a Customizer page for the client adapter named `EchoClientAdapter` using Inter-operable ORB reference binding:



The elements of the Customizer page and their purposes are:

Host

The name (or the IP address) of the machine on which the object activation daemon is running.

Port

The port on which the VisiBroker ORB is running (default is 14000).

Available Servants

Displays a list of the names of servants on the specified ORB. When you select one, that name appears in the Servant field. If the list is empty, click Refresh.

Only the servers registered with the object activation daemon are listed.

Servant

Specifies the name of the servant to which this client adapter attaches.

Advanced

Displays the Advanced VisiBroker Settings dialog box. See page 3-23.

Refresh

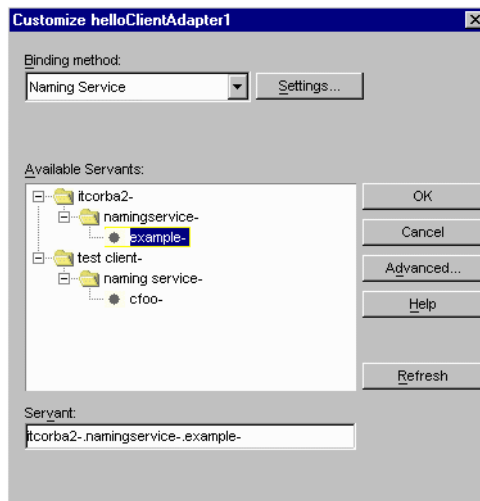
Populates the Available Servants text area.

Settings

When a servant has been selected from the Available Servants list, the Settings button becomes enabled.

Naming Service binding

Here is an example of a Customizer page for the client adapter named `EchoClientAdapter` using Naming Service binding:



The elements of the Customizer page and their purposes are:

Settings

The Settings button is always enabled for Naming Service binding.

Available Servants

Displays the naming paths for the servants that are registered to the Naming Service.

Servant

Specifies the name of the servant to which this client adapter attaches.

Advanced

Displays the Advanced VisiBroker Settings dialog box. See page 3-23.

Refresh

Populates the Available Servants text area.

Now that you have seen all the parts of the Client Adapter Wizard, you might just try exercising it a bit by continuing the walkthrough. See “A walk-through of the Client Adapter Wizard” on page B-5. No screen shots are provided—just a quick list of steps to follow.

Distributed Debugging

Distributed debugging works essentially as the same as non-distributed debugging in previous versions of Visual Cafe. The important differences are that the Distributed Debugging Services of Visual Cafe Enterprise Suite let you execute your project on a remote machine and, during a single debug session, debug Java code executing in multiple Java Virtual Machines (VMs). For details on using the basic debugger functionality, consult the *Visual Cafe User's Guide*.

This chapter discusses:

- ◆ Overview of distributed debugging features
- ◆ About the distributed debugging executables
- ◆ Some distributed debugging scenarios
- ◆ Installing Distributed Debugging Services
- ◆ Configuring distributed debugging
- ◆ Running and debugging remote processes
- ◆ Troubleshooting

Overview of distributed debugging features

The distributed debugging feature allows you to:

- ◆ Start and stop processes on remote host machines
- ◆ Set breakpoints in these processes
- ◆ Step through the running code across multiple host machines

At any time before or during a Visual Cafe debug session, you can select a debug VM (a debuggable VM) or add another debug VM to attach to. The debug VMs are not attached until you have started a debug session.

- ◆ You can control execution and examine program states of multiple debug VMs during a single debugging session from within Visual Cafe.
- ◆ You can attach to, and detach from, debug VMs running on remote machines and seamlessly debug the Java code running on them. The remote debug VMs, even on different operating systems, appear to be local.
- ◆ You can set breakpoints, step, browse variables, and control execution of the remote debug VMs from the Visual Cafe environment while debugging.
- ◆ You can (if you are using the debug VM provided with Visual Cafe Enterprise Suite) perform incremental debugging, making changes to the program while it is paused in the debugger, then continue running with the changes in place.

About the distributed debugging executables

Two executables are associated with Distributed Debugging Services: `ddservices` and `debugvm`.

- ◆ The `ddservices` executable starts Distributed Debugging Services, which tracks debuggable VMs on the network, serves class files to instances of `debugvm` on other hosts, and lets you start processes on remote hosts from Visual Cafe.
- ◆ The `debugvm` bootstrap executable loads debuggable VMs on the remote machine and tells them to register with Distributed Debugging Services. At run time, `debugvm` loads the required classes into the

remote VM so you can use Visual Cafe to debug on your development machine.

The `ddservices` executable

You must execute a copy of Distributed Debugging Services on your development computer first. This is the copy that Visual Cafe on the development computer uses to communicate with remote copies.

- ◆ On computers running the Win32 operating systems, there are two executables:
 - ❖ `ddservicesw` runs on Windows operating systems.
 - ❖ `ddservices` runs on MS-DOS.
- ◆ On computers running the UNIX operating system, `ddservices` is a script that launches executables.

Each computer that has a VM running must also have a copy of Distributed Debugging Services running before that computer can participate in distributed debugging.

If you are debugging on a particular remote computer, it is recommended that you make Distributed Debugging Services a startup process for the remote computer. Distributed Debugging Services runs without a console window.

To stop the execution of Distributed Debugging Services, type:

```
ddservices -stop for the UNIX or MS-DOS version
```

```
ddservicesw -stop for the Windows operating system version
```

at the command line.

The `debugvm` executable

The `debugvm` executable on your development machine starts a debug VM, registers it with Distributed Debugging Services on your development machine, and then executes your Java class. It also handles passing the debug password to Visual Cafe.

Classes that are run by `debugvm` can be debugged from Visual Cafe. The `debugvm` program automatically handles the debug password. If called with no arguments, it starts a VM to which a debugger can later attach.

Syntax

```
debugvm [options] [class]
```

Parameters

options can include:

- name *yourAppName* Specifies a name to identify this application
 - help Prints the full set of java options
- (These can be followed by any other `java.exe` options)

class is the name of the Java class to execute

Some distributed debugging scenarios

Below are a few debugging scenarios to give you a quick idea of the different ways in which the distributed debugging features of Visual Cafe can be used. They include:

- ◆ Client debugging into server code
- ◆ Attaching to a running Virtual Machine
- ◆ Debugging a server whose client is not in debug mode
- ◆ Debugging both server and client remotely

Client debugging into server code

In this instance, you have a project within Visual Cafe that uses servants from a number of remote servers.

To debug multiple remote servants:

- 1 Open your project in Visual Cafe.
- 2 Start your servers on the remote machines using `debugvm`.
- 3 Specify the debug VMs to which you wish to attach (use the Project ► Options ► Debugger-distributed dialog box).
- 4 Start debugging your client by choosing Step Into, or set a breakpoint and choose Project ► Run in Debugger.

Note that each attached VM and its threads can be viewed in the Process view.

Attaching to a running Virtual Machine

You can attach to any debug VM that meets the requirements found at “Attaching the debugger to a VM” on page 4-11. It is not necessary that a project be open.

To attach to a VM that is running:

- 1 Use `debugvm` to start your Java class on your remote host machine.
- 2 Choose File ► Attach to Process.
- 3 From the Available VMs dialog box, expand a host and select a VM.

Note: If you do not have a project open, a default one is created for you before the debug session is started.

If you have a project open, it will be used to begin the debug session. This option can be toggled in the Debugging tab (Tools ► Environment Options takes you to the Debugging tab).

Debugging a server whose client is not in debug mode

You can have your server running in Visual Cafe as a project, and set a breakpoint. When a non-debug client calls, your server will stop at the breakpoint.

Debugging both server and client remotely

You can debug both a server and a client on systems other than your development machine by using the following procedure.

To debug both a server and a client remotely

- 1 Open the `server` project on your development machine.
- 2 Set it to execute remotely:
 - a Choose Project ► Options.
The Project Options dialog box opens.

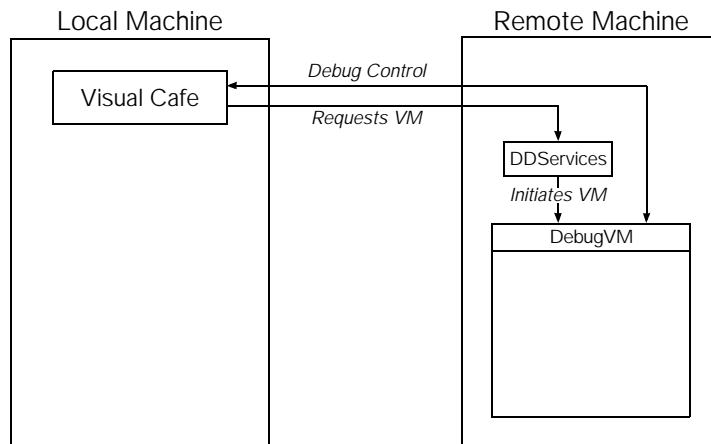
- b Change to Remotely and click Remote Settings to select a remote machine.
- 3 To execute remotely, choose Project ► Execute(R).
- 4 Open the Client project on your development machine.
- 5 Set it to execute remotely as in step 2 above.
- 6 Choose Project ► Options ► Debugger.
- 7 In the Debugger Category drop-down list, choose Distributed.
- 8 In the Attached VMs list, elect the remote machine by selecting the appropriate checkbox for the Server VM to attach to.
- 9 Start a debug session for the client. The server then is also attached.

Installing Distributed Debugging Services

To use the distributed debugging feature of Visual Cafe, you must install Distributed Debugging Services on the remote machine as well as your local development machine.

You must install two executables:

- ◆ The `ddservices` executable starts Distributed Debugging Services, which tracks debug VMs on the network and lets you start processes on remote host machines from Visual Cafe.
- ◆ The `debugvm` bootstrap executable loads debuggable VMs on the remote machine and tells them to register with Distributed Debugging Services. At run time, `debugvm` loads the required classes into the remote debug VM so you can use Visual Cafe to debug on your development machine. See “Executing the servant remotely” on page 3-9.



Installing on Windows

Installing the Visual Cafe Enterprise Suite automatically installs Distributed Debugging Services. If you don't want to install the entire suite on the remote machine, you can install just the Distributed Debugging Services.

To install Distributed Debugging Services on a Windows machine:

- 1 On the remote Windows machine, insert the Visual Cafe Enterprise Suite Disk One CD in your CD-ROM drive.

Normally, this starts the install program.

- 2 If step 1 didn't start the install program, run `Setup.exe`.
- 3 On the Disk One tab, under Visual Cafe Enterprise Suite, choose Symantec Java Runtime. This option automatically installs just the Distributed Debugging Services with the Symantec JDK.

To install on a computer with a third-part VM already present, choose Support for other Java Runtimes. You will be guided to a directory containing files you need to install manually.

- 4 Answer the prompts and click Finish.

The installation wizard installs the files on the remote machine.

Running on Windows

Distributed Debugging Services can be started on the Windows operating system either by launching Visual Cafe or by running the `ddservices` executable.

When started by launching Visual Cafe, the Distributed Debugging Services are part of the Visual Cafe process. When Visual Cafe exits, Distributed Debugging Services are no longer available.

There are two forms of the `ddservices` executable:

- ◆ `ddservices.exe` opens a console. Output appears only if debug logging is on (it is off by default).
- ◆ `ddservicesw.exe` is a non-console version. It appears only in your Task Manager list, and is not visible from the desktop.

The Visual Cafe installer gives you an option to set up Windows so that it launches Distributed Debugging Services whenever you log into Windows.

Note: If you can't run `ddservicesw.exe`, see Appendix A, "Troubleshooting Tips" in *Visual Cafe Enterprise Suite Getting Started*.

Installing on UNIX

If your remote machine is a UNIX platform, follow the instructions below.

To use the Distributed Debugging Services on the UNIX platform, you need to install from a `.tar` file that includes the debug wrapper (the cross-platform debugging support files), and three other files:

- ◆ `ddservices`, a shell script equivalent to `ddservices.exe`.
- ◆ `debugvm`, a shell script equivalent to `debugvm.exe`.
- ◆ `.debugvmrc`, a function definition file used by the `debugvm` shell script to set the environment variables.

To install Distributed Debugging Services to a UNIX machine from a Windows workstation:

- 1 On the remote Windows machine, insert the Visual Cafe Enterprise Suite Disk One CD in your CD-ROM drive.

Normally, this starts the install program.

- 2 If step 1 didn't start the install program, run `Setup.exe`.
- 3 From the Disk One tab, under Visual Cafe Enterprise Suite, choose Support for other Java Runtimes to install just the Distributed Debugging Services without installing the Symantec JDK.

You will be guided to a directory containing files you need to install manually.

The details of this installation process for direct installation to a UNIX machine are available in the `Platforms.txt` file in the `Platforms` directory on the Visual Cafe Enterprise Suite Disk One CD.

Running on UNIX

Log on to the remote host machine and type one of these commands:

- ◆ `>ddservices`
Runs with the console window open. This is the recommended way, so you can monitor the process.
- ◆ `>ddservices &`
Runs with the console window open, but as a background task.

To stop Distributed Debugging Services on a UNIX machine:

Log on to the remote computer and type the following command:

◆ `>ddservices -stop`

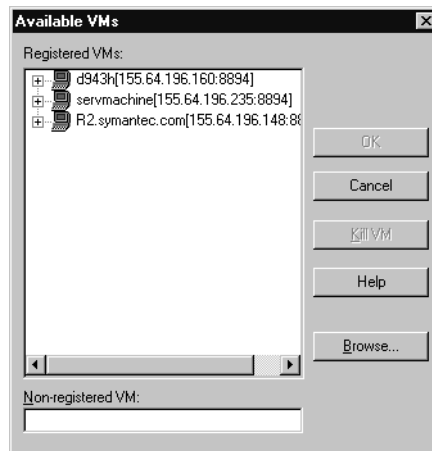
Verifying network connections

With Distributed Debugging Services installed and running on the remote machine, you should now verify the network connections.

To verify the network connections:

- 1 Start Visual Cafe on your development machine, if it is not already started.
- 2 Choose File ► Attach to Process.

The Available VMs dialog box shows the machines that have Distributed Debugging Services running, as in the example below.



If the remote machine you are going to debug on is listed, you are ready to go.

- 3 Click Cancel to close the dialog box.

Note: If the remote machine is not listed, see Tip C, in Appendix A, “Troubleshooting Tips” in *Visual Cafe Enterprise Suite Getting Started*.

Attaching the debugger to a VM

To use Distributed Debugging Services, you must use the right protocol and passwords, and use a VM that meets certain requirements. The requirements for attaching the debugger to a VM follow.

Socket protocol

The protocol used to communicate by TCP/IP between the development machine and the Distributed Debugging Services and the debug VMs must be Sun version 1.1.6. If the combination of your VM and your Java Runtime Environment does not use this protocol, you must use a Symantec class that translates between the 1.1.6 protocol and the one your VM and Java Runtime Environment use.

Passwords

When it is running in debugging mode, the VM provides you with a password. This password is used to uniquely identify the VM instance, and to provide the port number that the debugger should use to contact the VM.

If you have this level of compliance, you can at least type in the IP address and password in the Non-registered VMs list of the Available VMs dialog box, and attach to the VM. See page 4-10.

Registering the VM

Before you can register the VM, the Symantec classes (in the ZIP files in the platform packages) must be in place. This is necessary because either the VM must load the `Agent` class, or when the VM is available for debugging the user or the VM implementor must call:

```
com.symantec.itools.distdbg.agents.registry.DebugEnabler.  
enableDebugging(String password, Int timeout)
```

The package of the `Agent` class depends on the VM you use. The two agent packages are:

- ◆ JDK 1.1 (Sun VM): `sun.tools.debug.Agent`
- ◆ JDK 1.1 (Symantec modified VM): `symantec.tools.debug.Agent`

Configuring the network and port

Visual Cafe and the Distributed Debugging Services use the `comm.properties` file to determine which computers to communicate with and how long to wait for connections. The Distributed Debugging Services generate a default `comm.properties` file when first launched. When any one of Visual Cafe, Distributed Debugging Services, or `debugvm` is first run, it will attempt to create this file. The `comm.properties` file is located in the Visual Cafe `install` directory in:

```
java\com\symantec\itools\distdbg\agents\comm\
```

If you do not have write access to this directory, and a `comm.properties` file cannot be created, the default settings are used without creating this file. On a machine not running Visual Cafe, such as remote UNIX platforms, Distributed Debugging Services attempts to create the file (and its directory structure) in the Java home directory. If that creation is not possible, the default settings are used. In Visual Cafe on the development computer, choose Tools ► Environment Options ► Debugging and click the Network Setting button on the Debugging tab.

For other platforms, you must edit the file. If the `comm.properties` file could not be created on disk, it can be extracted from the ZIP file containing the Symantec debug support (such as `generic-jdk-1.1.7.zip`) and placed on the class path or put back into the ZIP file.

Note: The default `comm.properties` file should work for you in most cases.

Running on multiple machines on a subnet

The Distributed Debugging Services and the `debugvm` program use a pair of ports for communication. To make an independent system dedicated to a specific set of computers, you will need to alter the port numbers in their `comm.properties` files.

You can choose your own port numbers in order to have separate debugging systems running on your network. The port numbers must match on each machine on which you wish to do distributed debugging, otherwise VMs that are started by `debugvm` will not be readily visible from

the Visual Cafe environment. In addition, you must specify ports that are not otherwise in use. Usually, ports above 2000 are safe to use.

Here are some example port numbers:

```
BroadcastPort = "2001";  
CommPort = "2002";
```

`BroadcastPort` is where `ddservices` attempts to locate the Distributed Debugging Services.

`CommPort` is the communication port over which the Distributed Debugging Services and `debugvm` pass information.

Configuring distributed debugging

Several elements of Visual Cafe are involved in configuring distributed debugging. These are handled in:

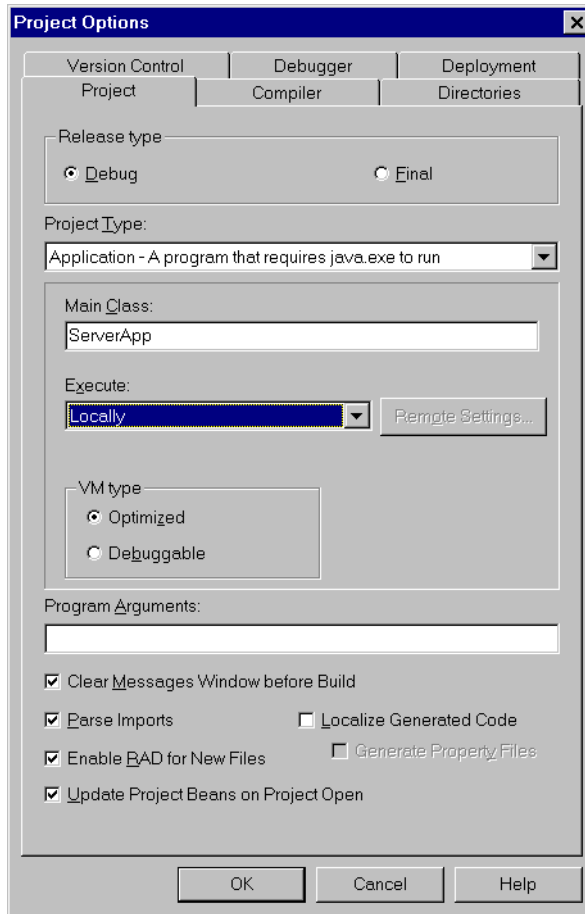
- ◆ Network Settings dialog box, which is accessible from the Debugging tab of the Environment Options (see “Network Settings dialog box” on page A-11).
- ◆ Debugger tab of the Project Options dialog box. See page 4-17.
- ◆ Project tab of the Project Options dialog box, including its Remote Execution Settings dialog box.

Setting debugging parameters for the project technique

If you are using the Project technique of distributed debugging, you must set the required parameters in the Project Options dialog box. These settings reside in the Project and Debugger tabs of this dialog box.

This section covers the options you set in the Project tab. For information about setting options on the Debugger tab, see “Specifying distributed debugging options for a project” on page 4-17.

The settings in the Project tab determine where and how the project runs and stops:



The Execute option has two settings:

Locally

The project is run on the local machine.

Remotely

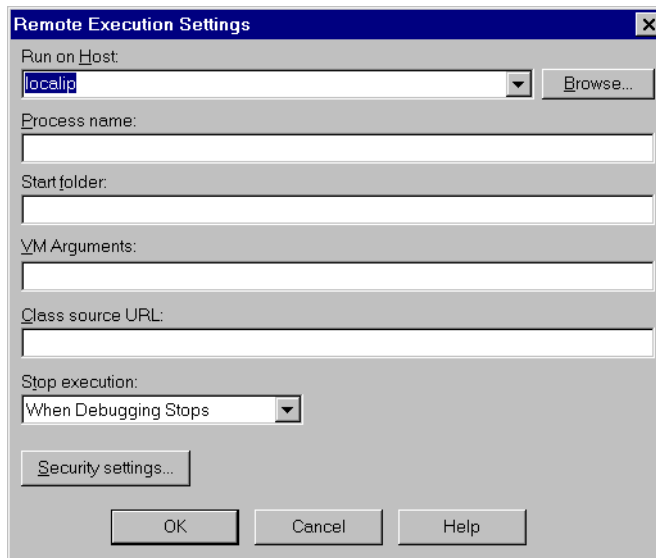
The project is run according to the parameters defined in the Remote Execution Settings dialog box.

For Win32 projects, the Execute option is disabled and set to Locally.

The Remote Settings button opens the Remote Execution Settings dialog box, as discussed in the next section.

Specifying remote execution settings

You specify settings for remote execution by using the Remote Execution Settings dialog box. To access this dialog box, click Remote Settings in the Project tab of the Project Options dialog box.



The required settings for remote execution are:

Run on host

This defines the IP address or host name of the machine on which the project is to be run. The default value is `localhost`. The drop-down list box will contain all host machines known to the Distributed Debugging Services.

Alternatively, you can type in an IP address. If a connection to this host machine is established, it appears in the Available VMs dialog box (described below). The Browse button opens the Available VMs dialog box.

Process name

This is the name that appears in the Available VMs view. The default value is the project name. It has no effect on execution, but serves

solely to identify the process in contexts such as the Available VMs dialog box.

Start folder

This is the folder on the remote host machine where the process is going to run. The default value is `null`.

VM arguments

These are the command-line arguments that are sent to the VM when it is started. The default value is `null`.

Class Source URL

This is the URL from which the Class Loader pulls updated class files. This value is sent as an argument to the remote VM. The default location is the development machine. The URL defines the protocol used for the class transfer. A proprietary protocol is used by default, but you can also use HTTP or FTP by typing in a different protocol in the URL.

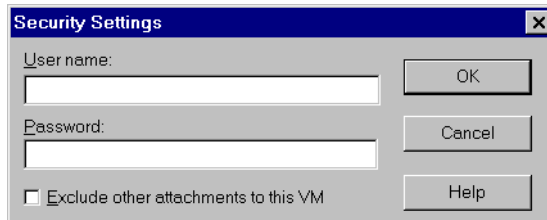
Stop execution

This defines when the remote process should be stopped. The possible values are:

- When Debugging Stops
- When Visual Cafe Exits (default)
- Never

Specifying security settings

The remote machine may require users to authenticate themselves by user name and password before they can execute a process. The Security Settings dialog box provides you a means to configure a user name and password. In the Remote Execution Settings dialog box, you click Security Settings to open the Security Settings dialog box, as shown here:



The options are described below:

User name

This is the user name under which the process is started on the remote machine. The default value is the current User ID on the development machine.

Password

This is the password for the user name.

Note: The process factory provided with Visual Cafe ignores the User name and Password fields. If you want to use this information, you must create your own process factory.

Exclude other attachments to this VM

If this option is selected, the VM is run in Protected Mode, preventing others from attaching to it. This prevents someone who is browsing from a different development environment from the VM.

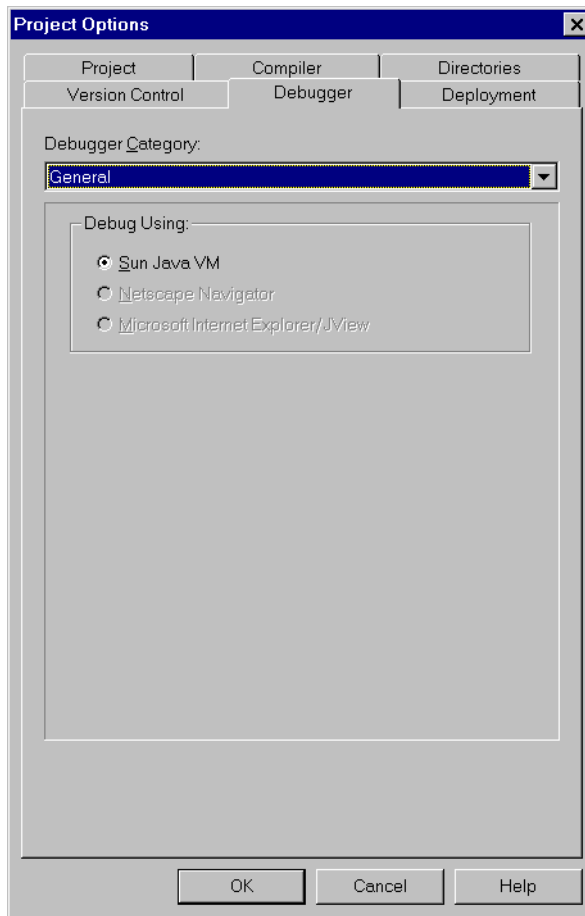
Specifying distributed debugging options for a project

Use the Debugger tab of the Project Options dialog box (choose Project ► Options), to specify debugging-specific options for your projects that are enabled for distributed debugging. These options are divided into three categories: General, Exceptions, and Distributed. This section discusses all three categories. (The Project Options dialog box is discussed more completely in the *Visual Cafe User's Guide*.)

General category options

In the Project Options dialog box, on the Debugger tab, choose General from the drop-down list. Visual Cafe then displays the Debug Using group,

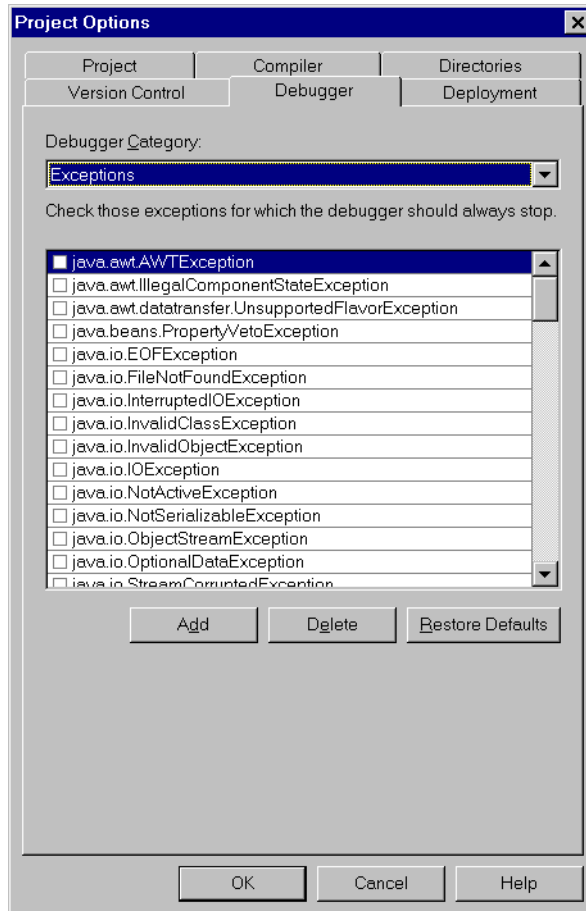
containing Sun Java VM, Netscape Navigator, and Microsoft Internet Explorer/JView radio buttons.



If you are debugging a Java application, the Debug Using group shows only Sun Java VM enabled. If you are debugging a Java applet, only Netscape Navigator and Microsoft Internet Explorer/JView are enabled. Select the appropriate Web browser.

Exceptions category

Use the Exceptions category to select the exceptions for which the debugger must stop. To display the Exceptions debugger category, click the Debugger tab of the Project Options dialog box. Then in the Debugger Category drop-down list, choose Exceptions.



To select an exception on which the debugger must stop:

- ◆ Check the box to the left of an exception name.

(Optional) To add an exception to the Exception scroll box:

- 1 Click Add.
- 2 Type in the name of the exception.

(Optional) To delete an exception from the scroll box:

- 1 Select the exception.
- 2 Click Delete.

(Optional) To restore the default setting:

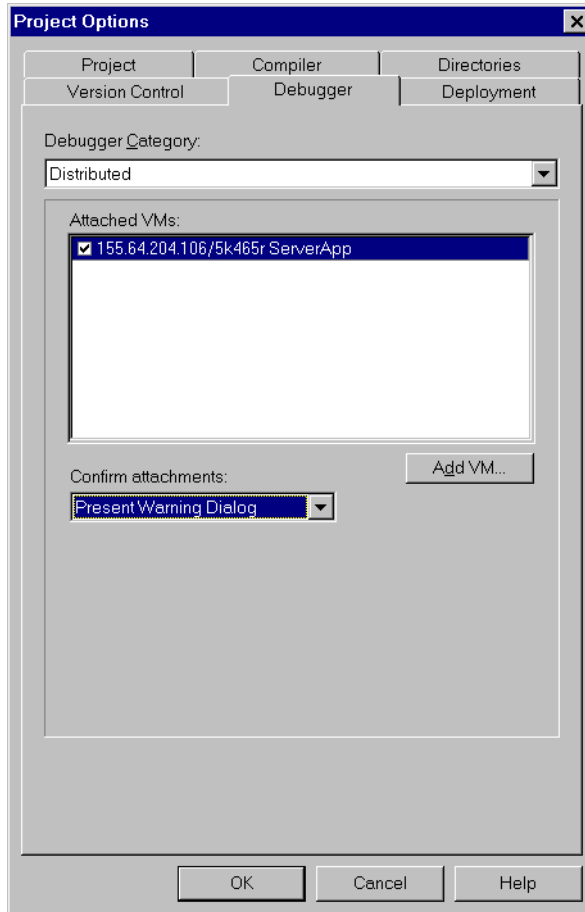
- ◆ Click Restore Defaults.

Distributed category options

In the Debugger Category drop-down list, choose the Distributed category to manage the settings that determine the VMs to which this project can attach.

To display the Distributed debugger category:

- 1 Click the Debugger tab of the Project Options dialog box.
- 2 In the Debugger Category drop-down list, choose Distributed.



The Distributed options are:

Attached VMs

The Attached VMs list enables you to manage the list of VMs that appear in the Processes View. This is a “working” list of VMs which you have previously attached to from this project. While the VMs are running and connected, they are kept in this list so that you can choose whether to attach to them in subsequent runs of the project by. If a VM stops running, it disappears from the list.

Add VM

Click Add VM to open the Available VMs dialog box, where you can select one or more VMs to which to attach. Once they are selected, these VMs then appear in the Available VMs list.

Confirm Attachments

You must choose one of the following options:

Present Warning Dialog

The VMs Detached warning dialog box appears if a VM on the Attached VMs list cannot be re-attached. This is the default setting.

Show Available on Debug

The Available VMs dialog box appears at the beginning of every debugging session. Use this option when the project has no `Main` class and you must attach to a running VM each time the project is run.

Never

The Available VMs dialog box is only visible if you explicitly choose it from the File ► Attach to VM menu. If a previously-attached VM is not available on subsequent project runs, Visual Cafe makes no attempt to locate it.

Running and debugging remote processes

Using distributed debugging, you can start and stop processes on the local computer and remote computers, set breakpoints in these processes, and step through the running code across multiple computers. Distributed debugging includes these major features:

- ◆ cross-platform debugging
- ◆ remote debugging
- ◆ remote execution
- ◆ multiple virtual machine debugging
- ◆ seamless debugging across multiple platforms

See the `\Platforms\Platforms.txt` file on Visual Cafe Enterprise Suite installation Disk One CD for information on the operating systems Visual Cafe supports.

Before debugging, you need to choose a way to start your debugging session and a way to start a debug VM for a process.

Starting a distributed debugging session

Visual Cafe provides three techniques for starting a distributed debugging session: Run in Debugger, Attach, and Debug in Waiting VM. Choose the technique that best suits the Java program you are developing.

Run in Debugger

You use just one Visual Cafe command to start both a debug VM and a debug session with the Run in Debugger technique. You can use this technique to test a Java program on a platform other than Win32.

To use the Run in Debugger technique:

- 1 Place your source files in a Visual Cafe project.
- 2 Set Project Options as needed.

Set Release Type to Debug. Set Project Type to the type of project you want to run, and if Execute (R) is set, be sure to establish the Remote Settings values.

- 3 Choose Project ► Run in Debugger or Project ► Step Into.

If you chose Run in Debugger, depending on the program type, the Java program runs until a breakpoint is hit, or to program completion. If you chose Step Into, execution is paused at the program entry point.

Visual Cafe launches your Java program in a debug VM and starts a debug session with this single command. The process can run locally or remotely.

If you are executing on a remote computer, Distributed Debugging Services on the remote computer starts a debug VM, and the Class Loader in the remote VM pulls the needed classes and resources to the remote computer. Classes are not copied onto the remote computer, but loaded into the remote VM. The output of the `System.out` and `System.err` classes are directed to the Visual Cafe Messages window.

For remote execution, you should use the `Application` project type only.

Attach

When you use this technique, Visual Cafe attaches to a process—a debug VM that launched a Java program—then lets you debug it, including stepping into the source code and setting breakpoints. For example, you

could debug an embedded VM (a Java application driven from an executable).

To use the Attach technique:

- 1 Start an application in a debug VM.
- 2 Choose File ► Attach to Process to attach to the VM.
- 3 Debug your application.

If the source files are in the active project, Visual Cafe automatically uses these source files. Otherwise, it prompts you for source files, so your source files do not have to be in a project for you to be able to debug. You can attach to VMs running locally or remotely.

Debug in Waiting VM

When you use this technique, Visual Cafe attaches to a debug VM that has not launched a Java program (a waiting VM). Then Visual Cafe supplies the main class name so you can debug your program. The waiting VM can be that on the local computer or a remote computer; however, if it is on the local computer, it is easier to use the Run in Debugger technique instead (there is no advantage to using the Debug in Waiting VM technique).

To use the Debug in Waiting VM technique:

- 1 Place your Java application or applet source files in a Visual Cafe project.
- 2 Start a debug VM so it waits for a debugger to attach to it and tell it what class to load and run.
- 3 Choose Project ► Debug in Waiting VM.

Comparison of Distributed Debugging techniques

Here is a comparison of some features of these techniques:

Feature	Run in Debugger	Attach	Debug in Waiting VM
Starts VM	Yes	No	No
Requires project with main class specified	Yes	No	Yes
Class Loader used	Only if executing remotely	Only if Execute technique is used to execute remotely	No
Distributed Debugging Services needed to debug with remote computer	Yes	No	No
Debug starting at beginning of Java program	Yes	No, you launch the application then attach	Yes
Can remotely debug applications only	Yes	Yes	Can remotely debug applets, too

You might want to run your Java program on another computer besides the development computer to check for cross-platform compatibility, to make sure it runs with other JDKs, or to test in different environments, such as checking resource availability issues. For all techniques, a process running on a remote computer uses the JDK present on the remote computer, not the JDK on the development computer.

Sometimes you might want to use a technique that does not involve the Visual Cafe Class Loader. For example, maybe you think class transfer is slowing application execution too much, or there are side effects you want to avoid, or you do not want to use classes built by Visual Cafe.

Visual Cafe uses Distributed Debugging Services with the Run in Debugger technique to start a VM and transfer class files when you execute remotely. You might want to use the Debug in Waiting VM technique if Distributed Debugging Services is unable to start VMs in the correct context, for example, with the correct user ID on Unix computers.

Starting a debuggable VM

Visual Cafe provides two techniques for starting a debuggable VM:

- ◆ **Execute technique** — With this technique, you place your source files in a Visual Cafe project, set Project Options, then choose Project ► Execute. Visual Cafe launches your Java program in a VM. If you execute remotely, Visual Cafe transfers the needed classes and resources through the Class Loader and Distributed Debugging Services.
- ◆ **Command Line technique** — Use this technique to start a debug VM and optionally a Java program from the command line.

You can use either technique to start a debug VM so you can attach to it. You use the Command Line technique to start a waiting VM.

Remember that when you use the Run in Debugger technique, Visual Cafe starts both a debug VM and a debug session.

Attaching to multiple VMs

Using the Attach technique, you can attach to multiple VMs during one debug session. This technique is ideal for debugging multiple-tier applications. If a debug session is in progress when you choose File ► Attach to Process, Visual Cafe attaches to the VM so you can debug the process in the current debug session. In addition, you can set Project Options so Visual Cafe automatically attaches to processes when you use the Run in Debugger or Debug in Waiting VM techniques.

Debugging and restarting

Once a VM is attached, Distributed Debugging Services will attempt to locate the source files for this process on the development machine. It uses the Source Search Path for the project to locate these files.

When you modify these files, you are modifying them on the development machine. Before these changes can take effect, the classes on the remote host machine must be updated. Distributed incremental debugging is not available. To effect the changes on remote host machines, you must stop

execution, modify the source, recompile, and update the classes on the remote host machines.

If a process is started using the Project technique, these updates are handled automatically by the Class Loader.

If a process is started using the Attach technique, then you must move them to the remote host machine independently of Visual Cafe.

When debugging is resumed, the list of attached processes is confirmed. If a previously-attached VM cannot be re-attached, and the Confirm Attachments parameter is set to Present Warning Dialog Box, and a dialog box appears. It tells you which VMs could not be re-attached, and offering you a chance to either ignore the matter or go to the Attach VMs dialog box and re-attach them.

Stopping processes

Whether or not the remote process gets stopped in various circumstances depends on a variety of things, including:

- ◆ Project technique
When debugging is stopped, processes are stopped according to the Stop Execution option in the Project Options dialog box.
- ◆ When Debugging Stops
The process and its VM is stopped when debugging is stopped. When debugging is resumed, a new VM is launched, the process is started in it, and it is automatically attached.
- ◆ When Visual Cafe Exits
The process is left running and attached until Visual Cafe is exited, at which time the VM is stopped. This is the default.
- ◆ Attach technique
When debugging is stopped, attached processes are left running and attached.
- ◆ Stopping VMs
You can also kill VMs that you started from the Available VMs dialog box. Choose View ► Available VMs to display it.

Adding Platform support for unlisted JDK versions

While the “generic” wrappers work on Linux and FreeBSD platforms, some editing must be done to a file called `Platform.properties` in order for a specific Linux or FreeBSD JDK version to be recognized. This file must first be unzipped from the wrapper, edited, and zipped back into the wrapper with 0% compression and the same directory names.

Use the following command line to unzip the file:

```
unzip com\symantec\itools\lang\Platform.properties
```

Edit according to conventions used in the file, which is commented to be user-friendly.

```
zip -r -0 generic-jdk-1.1.7
com\symantec\itools\lang\Platform.properties
```

Process factories

When Distributed Debugging Services spawns a `debugvm`, it uses a

process factory class to do so. The name of this class is specified in the file:

```
JAVA_HOME\com\symantec\itools\distdbg\agents\comm\
comm.properties.
```

The default value is:

```
com.symantec.itools.distdbg.ProcessFactoryImpl
```

This implementation uses the file `com\symantec\itools\distdbg\agents\defaultprocessfactory.properties` to determine the appropriate command line to use, for the current platform, to start a `debugvm`.

Here is a pair of example entries:

```
...
SolarisX =
(
"xterm",
"-e",
"\THECOMMAND",
);

Solaris =
(
"sh",
"-c",
"\THECOMMAND 2>/dev/tty >/dev/tty </dev/tty &",
```



```
);  
...
```

Distributed Debugging Services chooses which entry in the properties file will be used, based on information that is passed on the command line when `ddservices` is started, and on the value of the system property `os.name`.

For example, if you run `ddservices` with no command-line arguments on Solaris, the entry `Solaris` will be used.

However if you pass `-osuffix X` on the command line, the entry `SolarisX` will be used.

For more information, see the `com\symantec\itools\distdbg\agents\defaultprocessfactory.properties` file.

This file is located in `symclass.zip` in your Visual Cafe installation folder, as well as in the redistributable ZIP files for the various platforms.

Redirecting display

When you are executing remotely on a UNIX platform that offers support for X-Windows, you can redirect the display to another X-Server by setting the environment variable `DISPLAY` to the X Server of your choice, and by passing:

```
-osuffix X
```

to `ddservices` when you start it.

Handling socket factory classes

The sockets that are created for use by the Distributed Debugging Services are created with a socket factory.

The factory class is specified in:

```
JAVA_HOME\com\symantec\itools\  
distdbg\agents\comm\comm.properties.
```

The default factory class is:

```
com.symantec.itools.distdbg.agents.comm.SocketFactoryImpl
```

This implementation simply creates instances of `java.lang.Socket`. If you wish to use a different socket implementation, you can do so by

implementing your own socket factory and setting the `SocketFactory` property in `comm.properties` to the name of your implementation.

Note: You will have to make this change to `comm.properties` on all systems that are to use your implementation, and you will have to distribute your implementation to all the same systems.

Your implementation should extend `com.symantec.itools.distdbg.agents.comm.SocketFactory`.

The source for `com.symantec.itools.distdbg.agents.comm.SocketFactoryImpl` and that for `com.symantec.itools.distdbg.agents.comm.SocketFactory` are in the file `java/lib/symclass.zip`, in your installation folder.

Process factory classes

The processes that are created for use by Distributed Debugging Services are created with a process factory.

The factory class is specified in the following file:

```
JAVA_HOME\com\symantec\itools\distdbg\agents\comm\  
comm.properties
```

The default factory class is `com.symantec.itools.distdbg.ProcessFactoryImpl`.

This implementation just uses `java.lang.Runtime.exec` to create instances of `java.lang.Process`. It doesn't have the ability to set the user ID of the process it creates, nor to the ability to set the working directory of the process it creates.

You can implement your own process factory that does these things, and then you can set the `ProcessFactory` property in `comm.properties` to the name of your implementation, causing Distributed Debugging Services to use your process factory.

You can also create a process factory to provide for security other than what might be offered by your operating system.

Note: You will have to make this change to `comm.properties` on all systems that are to use your implementation, and you will have to distribute your implementation to all the same systems.

Your implementation should extend `com.symantec.itools.distdbg.ProcessFactory`.

The source to `com.symantec.itools.distdbg.ProcessFactoryImpl` and to `com.symantec.itools.distdbg.ProcessFactory` is in the file `symclass.zip` in the `java\lib` directory of your Visual Cafe install directory.

Adding Visual Cafe machine's IP address to `comm.properties`

Some UNIX installations and Java installations do not support network broadcasts. This prevents Distributed Debugging Services on certain remote machines from communicating with other Distributed Debugging Services on the network—in particular, the one on the development machine (the machine running Visual Cafe). In these instances, you can add the IP address of any machines which will be running Visual Cafe to the `comm.properties` files on the remote machine.

To add an IP address to the remote `comm.properties` file:

- 1 Unzip the file `com\symantec\itools\distdbg\agents\comm\comm.properties` from the `symantec.zip` file on the remote computer.

Note: If `ddservices` is installed by a user who has write access to the `JAVA_HOME` directory, this file will be created there in the above mentioned directory. In this case, simply edit this file. No extracting and reziping is necessary.

- 2 Add the IP address to the `BroadcastTargetInfo` section of the file. For example:

```
BroadcastTargetInfo =
{
155.64.196.13 = <= address of remote machine
(
"255.255.255.255",
"155.64.197.228", <= addresses of VCafe machines
"155.64.196.129" <=
);
127.0.0.1 =
(
"127.255.255.255"
);
};
```

- 3 Zip the edited `comm.properties` file back using 0% compression and preserving the directory names.

Using the Seamless Stepping Option (RMI only)

You can enable seamless stepping in distributed applications that communicate through RMI by clicking the checkbox in:

Tools ► Environment Options ► Debugger ► Use Seamless Stepping

This setting controls stepping behavior and the display of the call stack.

Stepping behavior

When seamless stepping is on, the debugger skips over the RMI stubs and marshalling code, and steps directly into the code running on the remote VM. However, when seamless stepping is on, the debugger stepping process runs more slowly.

To stop in a distributed application when Seamless Stepping is off, you must first set and trigger a breakpoint in the code of the running remote process.

When Seamless Stepping is on, you can step through and directly into code on the remotely running VM.

The option is located in the Debugging tab in the Environment Options dialog box (see “Debugging tab” on page A-10), and is on by default.

Displaying the call stack

Besides controlling the stepping behavior, the seamless stepping option also controls the call stack decode across VMs. When Seamless Stepping is off, the call stack will display only the current VM’s call stack. When it is on, it omits the display of Stub entries and other details of inter-VM communication.

The seamless stepping option takes effect at the beginning of the debug session, so if you change it while you are debugging, you must restart debugging for the change to take effect.

This option applies to debugging RMI applications and does not apply to debugging CORBA applications.

The comm.properties file

The network settings for distributed debugging are held in the `comm.properties` file. Here is an example of the `comm.properties` file.

```
{
RetryCount = "4";
MulticastSupported="true"
BroadcastPort = "8894";
GetLocalHostReliable = "true";
BroadcastTargetInfo =
{
    155.64.196.74 =
    (
        "255.255.255.255",
        "155.64.197.255",
        "155.64.196.129"
    );
    127.0.0.1 =
    (
        "127.255.255.255",
    );
};
ProcTimeOut = "45000";
ServicesHasConsole="false"
Address = "155.64.196.74";
NetTimeOut = "10000";
VMDateFormat = "HH:mm:ss:SSSS dd/MMM/y";
ResourceCacheDir = "/opt2/home/debug/cache";
ServicesExecutable = "ddservicesw";
ProcessFactory =
"com.symantec.itools.distdbg.ProcessFactoryImpl";
SocketFactory =
"com.symantec.itools.distdbg.agents.comm.SocketFactoryImpl";
CommPort = "8894";
}
```

Each of the settings in the `comm.properties` file is discussed below. All settings are accessible from the Environment Options dialog box unless stated otherwise.

RetryCount

The number of times to retry operations such as broadcasting or checking for the successful creation of a spawned process.

Accessibility: This property is accessible from the Visual Cafe Environment Options dialog box.

Default: `RetryCount = "4";`

MulticastSupported

Indicates whether multicast is supported on this platform.

If multicast is not supported on a system, the Distributed Debugging Services may report a `java.io.IOException` with the message "Permission Denied". This exception would occur on start up of the Distributed Debugging Services.

Accessibility: This property is not accessible from the Visual Cafe Environment Options dialog box. (It is not relevant either, because Visual Cafe itself only runs on platforms that support multicast.)

Default: `MulticastSupported = "true";`

BroadcastPort

The port the Distributed Debugging Services uses for broadcasting the initial "hello I am here" message.

Accessibility: This property is accessible from the Visual Cafe Environment Options dialog box.

Default: `BroadcastPort = "8895";`

GetLocalHostReliable

Indicates whether this JDK implementation suffers from the bug that breaks `InetAddress.getLocalHost`.

The symptom of this bug is that any attempt to identify the local IP address results in 127.0.0.1 (localhost).

This bug causes the Distributed Debugging Services to be isolated from other machines, because it cannot broadcast its IP address.

If your JDK implementation suffers from this problem, you will see this warning when you start the Distributed Debugging Services:

```
"Warning: Initializing address to localhost. Remote
processes will not be able to connect to this system."
```

Note that seeing this message does not automatically mean that your JDK has this bug. It may be that your OS network configuration requires you to be dialed in to an ISP for IP address assignment, but you are not currently

dialled in. (This is frequently the case with Windows 95 systems that rely on dial-up connections to access the Internet.)

Accessibility: This property is not accessible from the Visual Cafe Environment Options dialog box, because the JDK it uses does not suffer from this problem.

Default: `GetLocalHostReliable = "true";`

BroadcastTargetInfo

Specifies the network interface cards to broadcast from, and the target IPs that each one sends to.

The form of this entry is:

```
NIC1 =
{
    "Target1",
    "Target2",
    ...
};
NIC2 =
{
    "Target1",
    "Target2",
    ...
};
...
```

For example

```
BroadcastTargetInfo =
{
    155.64.196.107 =
    (
        "255.255.255.255",
    );
    155.64.200.108 =
    (
        "155.64.201.255",
    );
    155.64.196.109 =
    (
        "200.108.100.4",
    );
};
```

```
127.0.0.1 =  
(  
    "127.255.255.255",  
);  
};
```

The above tells the Distributed Debugging Service to:

- ◆ Broadcast from 155.64.196.107 to the all systems on the local network. (255.255.255.255 is interpreted as “all machines on the local network”).
- ◆ Broadcast from 155.64.200.108 to all systems on the network 155.64.200.0. (Assumes a subnet mask of 255.255.254.0.)
- ◆ Send from 155.64.196.109 to the machine with the IP 200.108.100.4.
- ◆ Send to anything that happens to be listening on the local host pseudo NIC.

Accessibility: This property is accessible from the Visual Cafe Environment Options dialog box.

Default:

```
BroadcastTargetInfo =  
{  
    <local IP> =  
    (  
        "255.255.255.255"  
    );  
    127.0.0.1 =  
    (  
        "127.255.255.255"  
    );  
};
```

ProcTimeOut

The time in milliseconds the Distributed Debugging Services will wait for a spawned `debugvm` process to register before giving up.

On slower systems, or on systems that have heavily loaded CPUs, you may need to increase this value.

If a timeout has occurred during remote execution, you will see one of the following messages in the Visual Cafe messages window:

- ◆ Timeout while waiting for process to register. The process is still alive. You may be able to attach to it. If so, consider increasing the process time out value on the remote system. If not, please verify the installation of the Distributed Debugging Services on this system.
- ◆ Timeout while waiting for process to register. The Distributed Debugging Services cannot determine the process status because the process factory returned null. The current process factory is: <factory name>

Accessibility: This property is accessible from the Visual Cafe Environment Options dialog box.

Default: `ProcTimeOut = "45000";`

ServiceHasConsole

Indicates whether the spawned `ddservices` has a console for messages. This property is intended for internal use only.

Accessibility: This property is not accessible from the Visual Cafe Environment Options dialog box.

Default: `ServiceHasConsole = "false";`

Address

Indicates which network interface to listen on. If your system has only one interface, it's not very important.

However, if you have a multi-homed system (that is, one that has more than one network interface) then you can indicate a particular network interface by placing its IP address here.

Accessibility: This property is accessible from the Visual Cafe Environment Options dialog box.

Default: `Address = "155.64.196.75";`

NetTimeout

Specifies the time in milliseconds a component will wait on an attempted network communication before giving up. The affected components are `debugvm`, Visual Cafe, and the Distributed Debugging Services.

This property has no bearing on RMI, CORBA, general TCP/IP, or general UDP communication in user's projects.

Accessibility: This property is accessible from the Visual Cafe Environment Options dialog box.

Default: `NetTimeout = "10000";`

VMDateFormat

Specifies the format of the representation of the current system time.

See `Java.src.java.text.SimpleDateFormat` in the Visual Cafe installation folder for information on the format syntax.

If you run `debugvm` without specifying a VM name with the `-name` option and without specifying a main class, then a `String` representation of the current system time is used as the VM name.

Accessibility: This property is not accessible from the Visual Cafe Environment Options dialog box.

Default: `VMDateFormat = "HH:mm:ss:SSSS dd/MMM/y";`

ResourceCacheDir

Specifies the location to cache resources in.

The default value is the user's home directory.

The resources cached are those that your application loads by means of the methods `getResource` and `getResourceAsStream`.

For more information on resources in Java, refer to JavaSoft document *Accessing Resources in a Location-Independent Manner* at:

<http://www.javasoft.com/products/jdk/1.1/docs/guide/misc/resources.html>

Accessibility: This property is not accessible from the Visual Cafe Environment Options dialog box.

Default: `ResourceCacheDir = "c:\\users\\default";`

ServiceExecutable

Specifies the name of the executable that spawns `ddservices`.

This is used in the case that you start a debuggable VM on Windows without first starting the Distributed Debugging Services.

Accessibility: This property is not accessible from the Visual Cafe Environment Options dialog box.

Default: `ServiceExecutable = "ddservicesw";`

ProcessFactory

Specifies the name of the class to use for process creation. This lets you provide support for creating a process that has a particular user ID and start directory, and restrict access to machines that are running the Distributed Debugging Services.

The default Java process creation mechanism doesn't allow for this, and that always is platform dependent.

The default implementation just uses `Runtime.exec` to create processes, and ignores any values of user ID or start directory that it receives.

For more information, refer to the following two files:

```
com/symantec/itools/distdbg/ProcessFactory.java
com/symantec/itools/distdbg/ProcessFactoryImpl.java
```

These files are located in `symclass.zip` in the `java\lib` directory of your Visual Cafe installation folder.

Accessibility: This property is accessible from the Visual Cafe Environment Options dialog box.

Default: `ProcessFactory = "com.symantec.itools.distdbg.ProcessFactoryImpl";`

SocketFactory

Specifies the name of the class to use for socket creation. This lets you use mechanisms such as SSL.

The default implementation simply creates instances of `java.lang.Socket`.

For more information, refer to the following two files:

```
com/symantec/itools/distdbg/agents/comm/  
SocketFactory.java  
  
com/symantec/itools/distdbg/agents/comm/  
SocketFactoryImpl.java
```

These files are located in `symclass.zip` in the `java\lib` directory of your Visual Cafe installation folder.

Accessibility: This property is accessible from the Visual Cafe Environment Options dialog box.

Default: `SocketFactory = "com.symantec.itools.distdbg.agents.comm.SocketFactoryImpl";`

CommPort

Specifies the port that TCP communication is to take place on. Except for the initial "Hello I am here" message, all other communications between the instances the Distributed Debugging Services, debuggable VMs, and Visual Cafe occur on this port.

Accessibility: This property is accessible from the Visual Cafe Environment Options dialog box.

Default: `CommPort = "8894";`

Troubleshooting

Here is a short list of things to do when remote execution of a project fails. A more detailed treatment is available in the *Visual Cafe Enterprise Suite Getting Started* manual.

Things to try when remote execution fails:

- 1 Try running (and debugging) the program locally.
This will help you to determine whether the program is executable at all.
- 2 Try running the program “locally remote”.
Select your own IP address as the target system and then run in the debugger. This will help you to determine whether the Visual Cafe class transfer feature is the problem.
- 3 Try running in the debugger on your target system.
This will help you to determine whether the application is exiting due to some uncaught exception.
- 4 Try copying the files manually to the target system and running.
This will help you to determine whether the Visual Cafe class transfer feature is the problem.
- 5 Also, before running in the debugger, consider wrapping the contents of your main method’s block with `try/catch/throwable` first.

To do this, change:

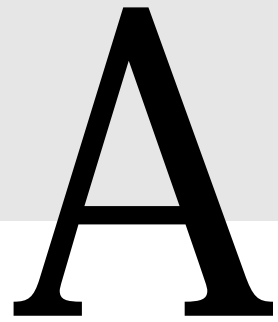
```
public static void main( String[] args ) {
    ...
}
```

To:

```
public static void main( String[] args ) {
    try
    {
        <original contents of main>
    }
    catch( Throwable t )
    {
        t.printStackTrace();
    }
    System.out.println( "done" );
}
```

And then set a breakpoint either on the `println` or on the `printStackTrace` call.

Now that you have some idea of the facilities offered for debugging a distributed application, you might just try a bit of distributed debugging by continuing the walkthrough. See “Exploring distributed debugging” on page B-9. No screen shots are provided—just a quick list of steps to follow.



Environment Options for Distributed Development

There are two tabs in the Environment Options dialog box where you can set default values to be used by the distributed development features.

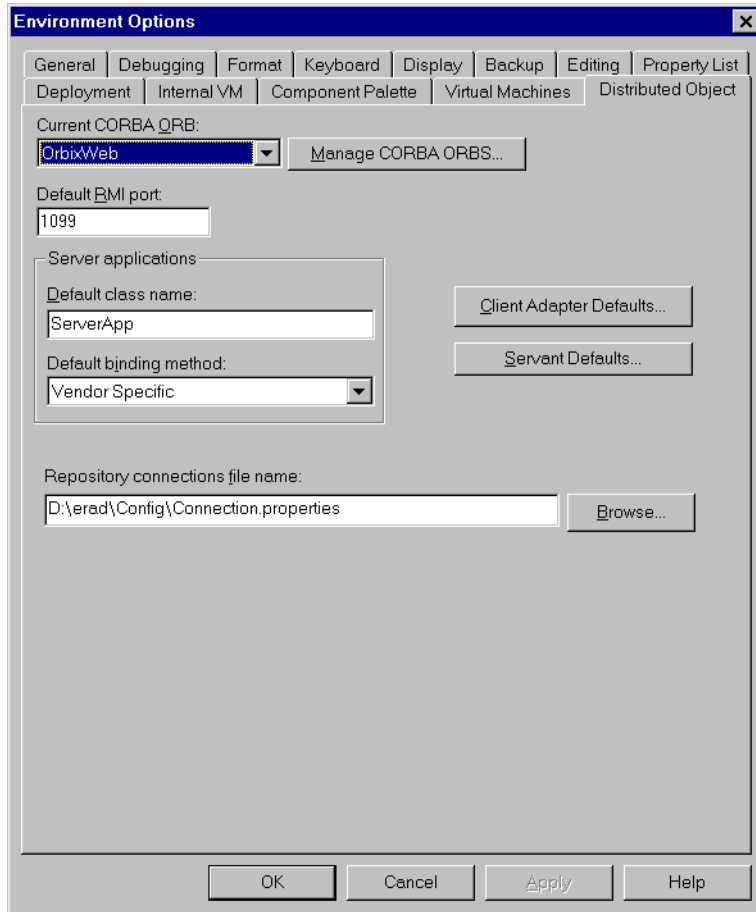
The Distributed Object tab discussion begins on page A-2. Related to the Distributed Object tab are:

- ◆ “Manage CORBA ORBs dialog box” on page A-4
- ◆ “Client Adapter Wizard Defaults dialog box” on page A-6
- ◆ “Servant Class Wizard Defaults dialog box” on page A-7

The Debugging Tab discussion begins on page A-10.

Distributed Object tab

The Distributed Object tab in Visual Cafe Enterprise Suite looks like this:



The elements of the tab are discussed below.

Current CORBA ORB

This defines the set of classes and ORB vendor tools that will be used for functions such as code generation and IDL compilation. The options in this drop-down list box are set in the Manage CORBA

ORBs dialog box (see page A-4). If no ORBs have been defined the combo box displays “(none)”.

Note: When you switch from one ORB to another, you must to restart Visual Cafe to make the change take effect.

Default RMI port

This defines the default port to be used when connecting to an RMI Registry.

Default class name

This defines the default name for the server application class. The wizard also uses this to name the `.properties` file for the server application. The default value is `ServerApp`.

Default binding method

This defines which binding method will be used when servants are instantiated in the server application.

The options are:

- ❖ `Inter-operable ORB Reference`
- ❖ `Naming Service`
- ❖ `Vendor Specific (is always done)`

Repository connections file name

This defines the location and name of the `connection.properties` file which contains the set of repository connections displayed in the Manage Repository Connections dialog box. This dialog box is used in both the Servant Class Wizard and the Client Adapter Wizard. The default location is:

installation_directory \Config\connection.properties

Alternatively, an administrator could set up these connections and place the file on a central server to save other users the effort of defining the settings.

Client Adapter Defaults

This button brings up the Client Adapter Wizard Defaults dialog box (see page A-6).

Servant Defaults

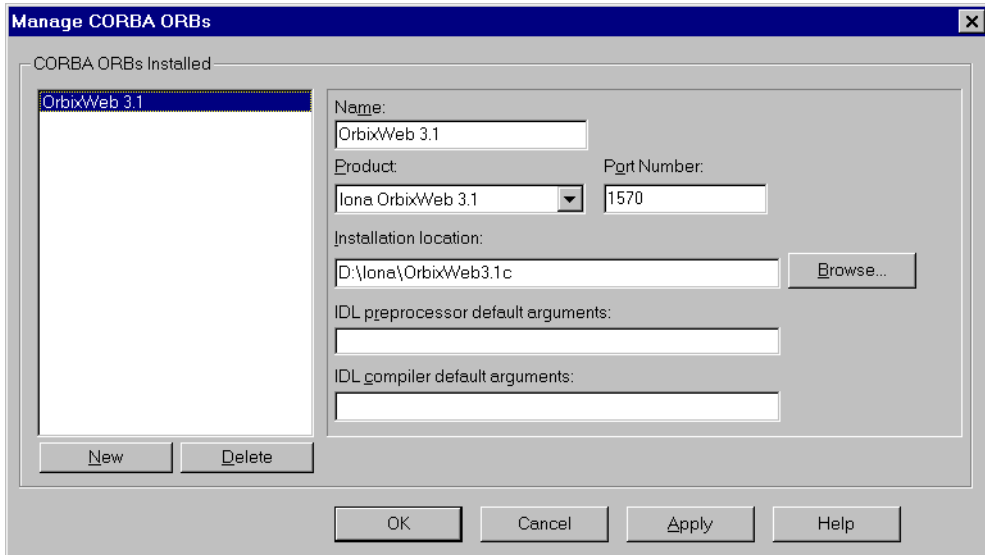
This button brings up the Servant Class Wizard Defaults dialog box (see page A-7).

Manage CORBA ORBs dialog box

The set of available ORBs is defined in the Manage CORBA ORBs dialog box. When you install Visual Cafe, it searches for installed ORBs and adds an entry listing all of the parameters for each ORB.

If there is no name associated with an installed ORB, the display shows its product name and port number. The Name field defaults to *product@port number*.

When you select an ORB in the list on the left, the settings for that ORB are shown in the panel on the right.



You can create any number of ORB definitions by selecting different ORB products or by selecting multiple versions of the same product on different ports. The options available include:

Name

The user-defined name for an instance of an ORB. These names appear in the Current CORBA ORB drop-down list box.

Product

The product name and version number of the ORB. This drop-down list shows only the set of ORBs that are supported. This string also

appears in the Client Adapter and Servant Class Wizards where <current CORBA ORB> is referenced.

Currently this list includes:

IONA OrbixWeb 3.1
Inprise VisiBroker for Java 3.x

Port Number

The port on which the ORB is running.

Installation location

The path to the initial folder where the ORB files were installed.

IDL preprocessor default arguments

This allows you to specify preprocessor commands for the IDL compiler.

IDL compiler default arguments

This allows you to specify the set of compiler arguments for the IDL compiler.

New

Adds an “Untitled” item in the list of ORBs, selects it, and clears the fields. You can then type in a name and type or select field values of your choosing.

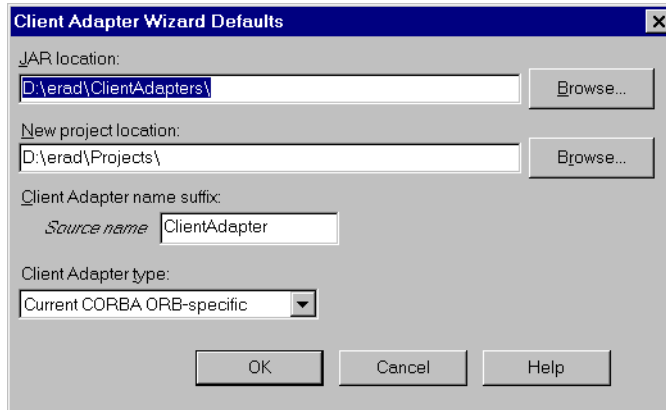
Delete

Removes the selected ORB from the list of defined ORBs.

When you have entered your values, click Apply to create the new ORB definition in the list.

Client Adapter Wizard Defaults dialog box

When you click Client Adapter Defaults on the Distributed Object tab of the Environment Options dialog box, the following dialog box appears:



The default values established here will be used by the Client Adapter Wizard when it generates the code for a new client adapter.

The options are:

JAR location

This defines where the JAR is to be created relative to the installation folder. If this path doesn't exist, it will be created when you click Finish in the wizard. The default path is *installation folder*\ClientAdapters\.

New project location

This defines where new project directories are to be created. If this path doesn't exist, it will be created when you click Finish in the wizard. The default location for this is *installation folder*\projects\.

Client Adapter name suffix

You can define the text that you would like included in the client adapter's name, appended to the class name (either an RMI servant or a CORBA servant's interface you selected in the Wizard). So, if the servant class name is `audit`, and you had "CA" in this text field, the Bean is named `auditCA`. The default suffix value is `ClientAdapter`.

Client Adapter type:

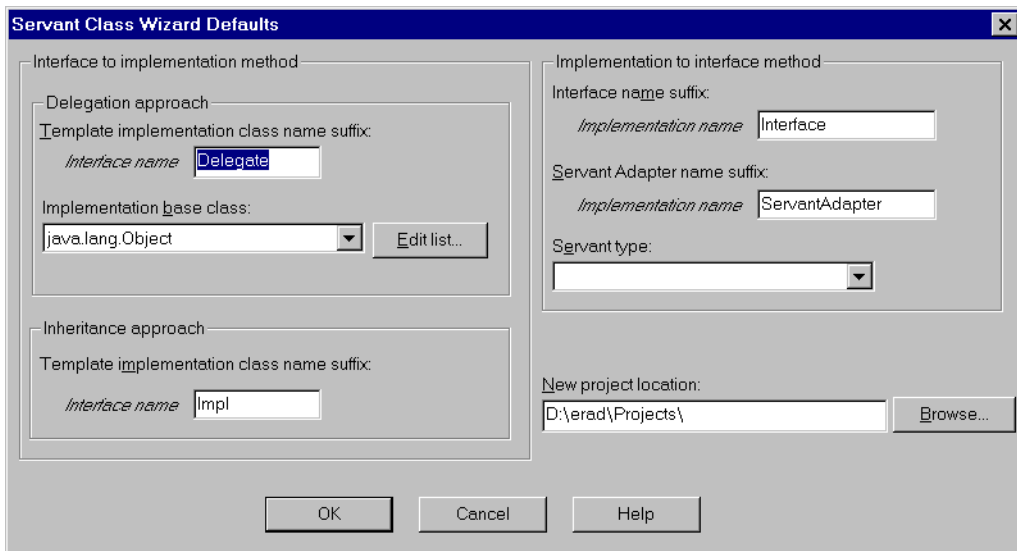
This defines the default client adapter type that will appear in the wizard drop-down list.

The options are:

Current CORBA ORB-specific

Servant Class Wizard Defaults dialog box

You can accept the default name suffixes that Visual Cafe provides for the components Java generates for the servant, or you can tailor them to your needs by using this dialog. When you click Servant Defaults, the following dialog box appears:



The default values established here are used by the Servant Class Wizard when it generates the code for a new servant adapter. The initial default values are shown in the image. There are three sections to the dialog box:

- ◆ Interface to implementation method group
- ◆ Implementation to interface method group
- ◆ New project location

Interface to implementation method group

The Interface to implementation group establishes the class name suffixes and the base class name for servants created from interfaces. It contains two subgroups: Delegation approach and Inheritance approach.

The Delegation approach group's elements are:

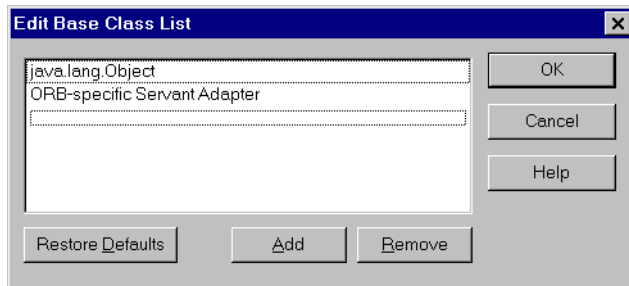
Template implementation class name suffix

This defines the default name suffix to be used for the template implementation file generated when the delegation method is used. The string entered will be appended to the name of the Interface you select.

Implementation base class

This defines the default base class that the template implementation should extend. The default is `java.lang.Object`. This item appears in the Extends combo box of the Customize Implementation class page of the Servant Class Wizard.

If you click Edit list, the following dialog box appears:



By default the list contains:

- ❖ `java.lang.Object`
- ❖ `ORB-specific Servant Adapter`

This dialog box allows you to add classes to appear in the Extends combo box.

Add

Creates an “untitled” entry in the list.

Remove

Deletes the currently selected class.

Restore Defaults

Removes any classes in the list and replaces the default list above.

The Inheritance approach group's only element is:

Template implementation class name suffix

This defines the default name suffix to be used for the template implementation file generated when the inheritance approach is used. The string entered will be appended to the name of the Interface you select.

Implementation to interface method group

The Implementation to interface method group establishes the suffixes for the interface name and servant adapter class name, and the servant type for servants created from Java implementation classes.

Servant Adapter name suffix

This defines the default name suffix to be given to the servant adapter created. The string entered will be appended to the name of the Implementation file you select.

Servant type

This defines the option that will appear in the Servant type drop-down list. The options are:

- ❖ Current CORBA ORB-specific
- ❖ RMI compliant (default)

If the servant is being added to a servant project, the default type is defined by the type of the server project.

New project location

This section consists of the New project location field and a Browse button.

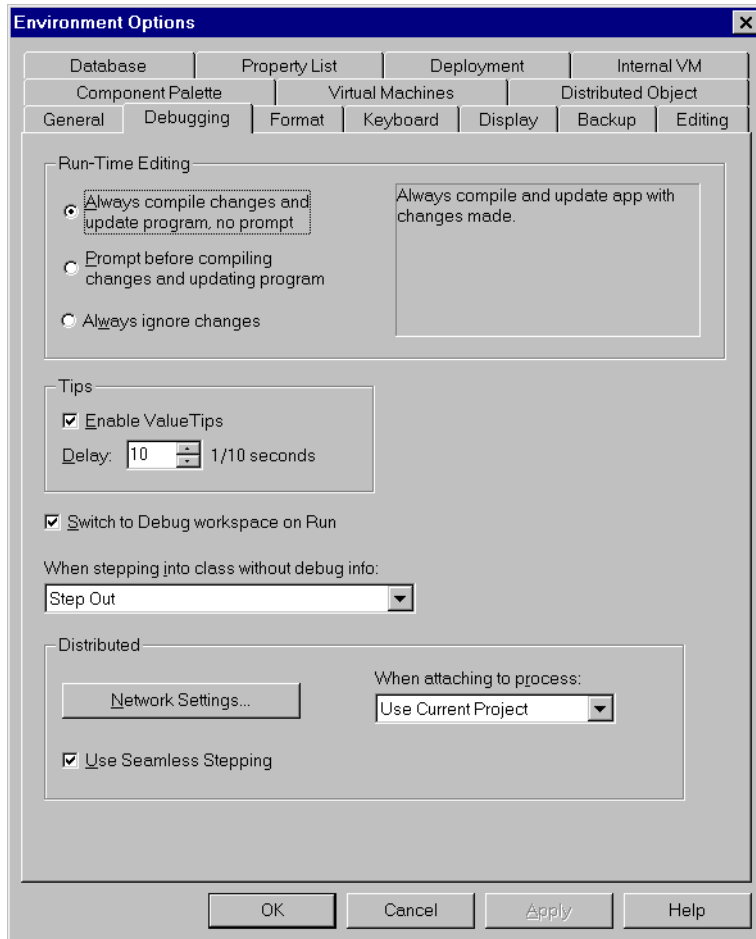
New project location

This defines the path in which the servant project is to be created. You may type a value in the field or use the Browse button to find the appropriate existing value.

If the value you type represents a path that doesn't exist, the path is created when you select Finish in the wizard.

Debugging tab

The Debugging tab specifies a number of configuration settings for debugging a Java application. The Distributed group contains a Network Settings button, a When attaching to process drop-down list, and a Use Seamless Stepping checkbox.



Network Settings:

This button brings up the Network Settings dialog box (see “Network Settings dialog box” on page A-11).

When attaching to process

This defines the behavior of Visual Cafe when the File ► Attach to Process option is executed. The options are:

- ❖ Create New Project
- ❖ Use Current Project

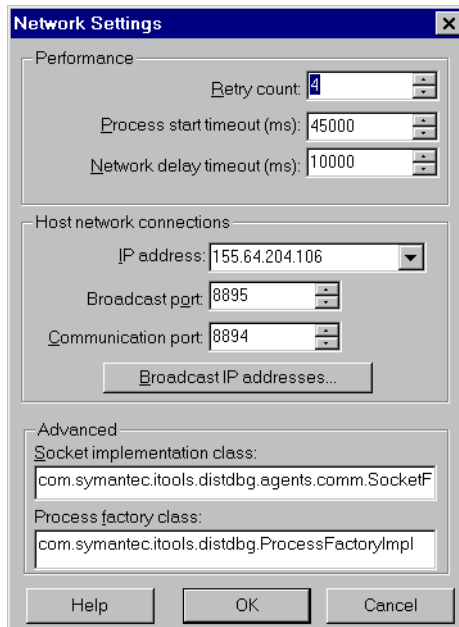
Use Seamless Stepping

When this is checked the debugger:

- ❖ Steps past all of the stub marshalling code (on the local VM)
- ❖ Stops at the first line of user code (on the VM where the remote call is running)

Network Settings dialog box

The default settings should allow the software to operate without user configuration. If necessary, you can modify the default settings through the Network Settings dialog box. On the development machine, this dialog box can be accessed from the Debugging tab of the Environment Options dialog box.



Use this dialog box to set the following parameters:

Retry count

This is the number of times the process will attempt to connect to the Distributed Debugging Services. The default value is 14.

Process start timeout (ms)

This is the amount of time, in milliseconds, that the Distributed Debugging Services program will wait for the `debugvm` process that it is trying to start. It is related to the speed and availability of the CPU on which the process is running. The default value is 1,000 ms.

The total time before it gives up is this setting multiplied by the retry count.

Network delay timeout (ms)

This allows you to adjust for the delay that occurs in socket communication among Visual Cafe, the Distributed Debugging Services, and the VMs loaded by `debugvm`. It is related to the speed of the network on which the application is running. The default value is 10,000 ms.

IP address

This is the address on which a socket connection will be made to this machine to be used for debugging communication. The drop-down list will contain all of the network cards on the development machine. The default value for this will be the IP address that Java provides.

Broadcast port

This is the port on which a debug VM will send a broadcast message advertising its availability to the Distributed Debugging Services program. The default port is 8895.

Communication port

This is the port on which communication with this VM will occur after it has registered with the Distributed Debugging Services program (for example, Start/Stop messages). The default port is 8894.

Socket implementation class

Defines the class used for implementation of the socket communication. The default value is `com.symantec.itools.distdbg.agents.comm.SocketFactoryImpl`.

Process factory class

This is an optional feature. The default value is `com.symantec.itools.distdbg.ProcessFactoryImpl`, which is what is used if the field is left empty.

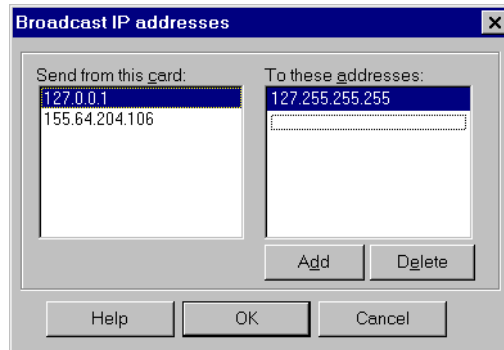
You can define a process factory that will be used to start processes within the debug VM. This will provide the ability to identify the user name and password to be used—a feature that is absent from the standard JDK implementation. A Process Factory is provided for NT by default.

Regardless of platform, the default process factory will receive and ignore any values for user name, password, and start folder.

The default process factory works on most platforms. It builds a command line for a given platform based on information found in the file `defaultprocessfactory.properties`, a copy of which can be found in `symclass.zip` in your installation folder.

Broadcast IP addresses

If you click Broadcast IP addresses, the following dialog box appears.



The IP addresses in the left listbox are the addresses of the cards from which broadcast messages are sent. Those in the right listbox are the addresses to which broadcast messages are sent.

To add a new address:

- 1 Position the cursor in the correct text area box.
- 2 Click Add.

An untitled address is added to the list box, which you can edit in place.

You can cut and paste addresses in the right list box.

To delete an IP address:

- 1 Select one or more addresses.
- 2 Click Delete.

The selected addresses are deleted.

The Delete button is disabled unless one or more addresses are selected.

To save the current settings:

- ◆ Click Apply.

The current settings are saved and the dialog box remains open.

Exploring Distributed Application Development

This appendix provides a brief exploration of the creation of a simple RMI-based servant, a client to interact with it, and some debugging of the distributed application.

A walk-through of the Servant Class Wizard

The Servant Class Wizard supports RMI, the Iona OrbixWeb ORB, and the Inprise Visibroker ORB. Because a distributed application would require two machines, and RMI is part of Java rather than a third-party product, this walk-through was designed to keep things simple by running on a single machine using RMI. The basic steps are outlined below. More detailed descriptions are provided in the next several pages.

To create an RMI-based servant class from existing Java source:

- 1 Create the `Hello` server application.
- 2 Create the `Hello` implementation file.
- 3 Use the Servant Class Wizard to create the `Hello` servant adapter and interface.
- 4 Run the server application.

The next few pages assume you have a basic working knowledge of the Visual Cafe product. Consult the *Visual Cafe User's Guide* for details of any operation that is not familiar to you.

Create the server application

Your first task is to create the server application—the mechanism to instantiate and publish your servant.

To create the Hello server application:

- 1 Start Visual Cafe Enterprise.
- 2 Create a new RMI Server application.
 - a Choose File ► New Project.
 - b Select RMI Server.
 - c Click OK.

The New Project dialog box disappears and a Project window with a name like `Untitled1` appears.

- 3 Save the project.
 - a Click the title bar of the Project window to select it.
 - b Choose File ► Save As.
 - c In the Save As dialog box, create a new folder.
 - d Open the new folder.
 - e Type a name for your project and click Save.

Create the implementation file

Your next step is to create a simple Java implementation that sends a reply to any client that invokes it.

To create the implementation file:

- 1 Create a new Java source file to be the implementation class for the `Hello` servant. It should contain the following:

```
public class Hello{
    public String sayHello()
    {
        return "Hello From Server";
    }
}
```

- 2 Save the file as `Hello.java` in the project folder you created above.

In the Save As dialog box, leave the checkboxes Add to project and Enable RAD selected.

- 3 Select `Hello.java` and click Add.

Note that `Hello.java` is now in the list under Files in Project.

- 4 Click OK.

The Project Files dialog box closes.

- 5 To build the project, choose Project ► Build Application.

Prepare the servant

Your next, and main, step is to use the Servant Class Wizard to produce:

- ◆ a servant adapter to handle the communication through RMI
- ◆ an interface file that the server application uses to bind the servant class to the RMI Registry.

Before a client can invoke a servant's methods, it must obtain a reference to the object. "Binding" the servant class to the RMI Registry means establishing a relationship in the Registry between the instance of the servant and a human-readable name.

To prepare the servant using the Servant Class Wizard:

- 1 Invoke the Servant Class Wizard by choosing File ► New Servant Class.

The Introduction page appears, giving a brief synopsis of what is going to happen, and offering you a checkbox to prevent this page from appearing in the future. Do not bother to select it now.

- 2 Click Next.

The Choose source type page appears, offering you two options for creating servants. You use the Java implementation option and the implementation file you created in step 2 of the preceding section.

- a Select Java implementation.
- b Click Next.

The Select implementation class page appears.

- 3 Select the class file to use as the implementation for the servant.
 - a Click Browse next to the Implementation class box.
 - b Browse to the `Hello.class` in your project folder and select it.

- c Return to the Select implementation class page.
- d Examine the default values, and note the following:
 - The default project into which the wizard generates files is your current open project.
 - The Servant type is `RMI compliant`. (If you were generating a CORBA servant, you could choose between ORBs in the Servant type drop-down list.)
 - When the wizard automatically inserts an instance of the servant into the `ServerApp.properties` file of the currently open project, it gives the instance the default name `Hello`. See “Server application properties” on page 2-39.
- e Click Next.
 - The Customize output files page appears.
- 4 Examine the default names that the wizard displays for the interface file and the servant adapter file it is to generate.
- 5 Click Next.
 - The Review options page appears.
- 6 Complete building your servant.
 - a Review the choices you made.
 - In a production environment, you might spot something that you want to change. To change a choice, click the Back button to go to the appropriate page of the Servant Class Wizard.
 - b Click Finish.

The Servant Class Wizard then generates your servant classes, adds the new `Hello` servant to your server application project, and terminates. The project is still active.

Run the application

Having completed the construction of the application, you can now run it to verify that it works. First, you can execute it locally, on the development machine, to make sure it executes at all.

To run the server application locally:

- 1 Choose Project ► Execute (L).

If the Project dialog box offers Execute (R) instead of Execute (L), select Project ► Options, and on the Project tab of the Project Options window, click the Execute drop-down list, select Locally, and click OK.

- 2 Your server application opens a new command window and tells you when it has launched your servant. Your servant class is now available and visible in an RMI registry which the server application has started.
- 3 The exploration of the Client Adapter Wizard, beginning below, needs to have the servant running locally.

If you are not continuing your exploration, halt your server application by typing CTRL-C in the command window.

If you look in the folder where you stored your project, you see:

- ◆ `HelloInterface.java` and `HelloInterface.class`: the java interface definition, derived from `Hello`.
- ◆ `HelloServantAdapter.java` and `HelloServantAdapter.class`: the servant adapter for `Hello`.
- ◆ `ServerApp.java` and `ServerApp.class`: the server application that manages the `Hello` servant.
- ◆ `ServerApp.Properties`: the property list for the server application.

A walk-through of the Client Adapter Wizard

In the first section of this appendix, you created and executed a servant. Here again, you are just exercising the wizard (in this case, the Client Adapter Wizard). Again, no screen shots are provided—just a quick list of steps to follow.

To create a client adapter, the server application must be running. This is because the Client Adapter Wizard uses the information the application published about the servant interface.

Your first task is to create the server application—the mechanism to instantiate and publish your servant.

To create the server application

- 1 Start the Client Adapter Wizard (select File ► New Client Adapter).
The Introduction page appears.

- 2 Click Next.

The Select servant page appears.

If you do not have any repositories defined at this point, the Select servant page is quickly overlaid by a Manage Repository Connections dialog box, which itself is then overlaid by a Select New Repository Type dialog box.

- a Select RMI Registry.
- b Click OK to close the Select New Repository Type dialog box. The Manage Repository Connections dialog box is now visible.

Note: Usually you enter the name or the IP address of the computer that the RMI Registry is running on in the Host text field, and the port number in the Port text field. But because you are connecting to a local RMI Registry, you can use the default Host and Port values of `localhost` and `1099`, respectively.

On your subsequent visits to the Select servant page, the Manage Repository Connections dialog box does not automatically appear, and your repository definition list persists. You can click the Manage Repositories button on the Select servant page to open the Manage Repository Connections dialog box to add and delete connections.

- 3 The Manage Repository Connections dialog box shows a connection to the remote RMI Registry that appears as a node in the Select Server Object page. The node should display the following text: `RMI Registry @ localhost; port 1099`. To troubleshoot your connection, click Help.
 - a Expand the node and select `HelloServantAdapter`.

Note: `HelloServantAdapter` represents the implementation class on the server. If you expand it by clicking the plus (+), you see the interface, `HelloInterface`. `HelloServantAdapter` contains the code that implements the methods defined in `Hello`. By expanding the `HelloInterface` node, you see the methods that the interface defines. To create the client adapter, you must select the `HelloServantAdapter` node. You can generate an RMI client adapter from either an implementation or stub class only.

b Click Next.

The Save Client Adapter page appears.

4 The Save Client Adapter page gives you options to save the client adapter you are generating. Make sure that:

a In JAR file is selected.

b Put bean in Visual Cafe Component Library is selected.

c In Visual Cafe Project is deselected.

d Include serialized instance is deselected.

e Click Next.

The Select Component Library folder page appears.

5 In the Select Component Library folder page, notice that the generated Bean is set to be placed in the Client Adapters folder, then click Next.

6 In the Review options page, click Finish.

The client adapter is now ready to use.

Using the client adapter

In order to use the client adapter you have created, you need to add it to a new or an existing project.

To add the client adapter to a new project:

1 Create a new AWT application project.

a Select File ► New Project.

b Select AWT Application and click OK.

c Select the project by clicking the title bar in the Project window.

d Save the project in a new directory as ClientApp.

2 Add the client adapter to the new project.

a Select View ► Component Library to open the Component Library.

b From the Client Adapters folder, drag HelloClientAdapter onto any open area in the client application Frame1.

If the Form Designer isn't open, double click the Frame1 object in the Project window's Objects tab to open it.

3 Add a button and a text field to the Form Designer - Frame1. Label the button Get Text.

- 4 Using the Interaction Wizard, create an interaction between the button and the text field to use the `sayHello` method of the `HelloClientAdapter` to set the text of the text field.
To create the interaction:
 - a Click the Interaction Wizard icon (on the left edge of the Visual Cafe menu bar, looking like two electrical plugs).
 - b Click the Get Text button and drag the thick line to the text field.
 - c In the Interaction Wizard, click Next.
 - d Make sure that Action performed in wizard is selected, then click Next.
 - e In the Action page, scroll down to select Set the text for Textfield, then click Next.
 - f In the Available objects list, select `HelloClientAdapter1`.
 - g Make sure the Methods checkbox is selected.
 - h Scroll down and select the `sayHello()` method.
 - i Click Next to review your choices, then click Finish (or just click Finish) to complete the interaction.
- 5 Open the Property List if it is not open yet (View ► Property List) and click the client adapter in the Form Designer. The Property List shows the client adapter's properties.
Observe two property fields: Hostname and Port Number. In these, you specify the name of the computer where the servant is running, and the port that it is listening on. This is the same information that you entered when you created the repository source in the Client Adapter Wizard. Because the server object is running on your local machine and listening on the default port, enter `localhost` for the hostname and `1099` for the Port Number.
- 6 In the `ServantName` field enter: `Hello`.
- 7 Save the project.

If you look in the `Client Adapters` folder in the Visual Cafe installation folder, you find the `HelloClientAdapter.jar` file that you created. It contains the files `HelloClientAdapter.class`, `HelloInterface.class`, `HelloClientAdapterBeanInfo.class`, as well as the JAR manifest.

At this point, you can run the client.

Exploring distributed debugging

In the first two sections of this appendix, you created a servant, a client adapter, and a client. Here, you are exercising the debugging facilities. Again, no screen shots are provided—just a quick list of steps to follow.

The RMI application must be running. Now we can explore debugging the servant from Visual Cafe.

To debug the servant:

- 1 Open the `Frame1.java` file in your `HelloClient` project, and place a breakpoint at the line of code that gets the message text in the `button1_ActionPerformed_Interaction1` method.
- 2 Start debugging by stepping into the client application (Project ► Step into).
- 3 When the application opens, click the Get Message button. Execution will stop at the breakpoint.
- 4 Open the Processes window (you can use the threadlike icon in the Visual Cafe toolbar).

In the Processes window (Threads window), note that both running VMs are displayed, but the server VM is disabled. In order to debug the server you need to attach to it.

- 5 To attach to a remote VM, select the File ► Attach.

Right-clicking a VM allows you to attach and detach it dynamically while debugging. Attached VMs have expandable plus (+) signs that reveal their threads. You can right-click to set the focus of any thread and see the call chain and source, and examine its context. Right-clicking allows you suspend, pause, and resume threads as well.

- 6 Step into the Server by pressing F11.

Note: You are stepping into code for the server. This code you are attempting to step into is not part of the client project, so Visual Cafe presents you with an Open File dialog box so that you can navigate to your server source code. In order to step through server source code, you must have a local copy of your Server source code accessible to Visual Cafe.

- 7 Once you have stepped into the server code, step through it using F10.

There are a number of interesting things to examine at this point.

- ❖ In the Calls window, note the call chain and how seamlessly the entry into the server VM is displayed. It is as if there were no inter-VM barrier to be concerned with.
 - ❖ A dashed line shows the separation between calls in different VMs.
 - ❖ Double-clicking a call chain entry displays the appropriate source and sets the context for examining variables.
 - ❖ The Messages window displays client output, the output of attached VMs, and environment messages.
- 8 Select Debug ► Stop to halt debugging.

G L O S S A R Y

A

adapter Bean A Bean that serves as a proxy for accessing another object. Drop an adapter Bean for a remote object into your project, and you can invoke methods of that Bean to access methods of the object.

attaching to a VM Establishing a connection between the Visual Cafe debugger and a debuggable VM. When a VM is attached, you can debug code running in that VM. The status of debuggable VMs is displayed in the Processes window. See also *VM*.

B

bind To *bind a name* is to establish an association of an object name with an object reference; one of the functions of the CORBA Naming Service and of the RMI registry.

See also *CORBA, Naming Service, and RMI Registry*.

bind method In CORBA, the method in the `NamingContext` interface that associates an object's name with its binding context.

binding A name-to-object relationship, recorded in a CORBA Naming Service database or in the RMI Registry.

C

client An object that invokes an operation on a distributed, or remote, object to obtain services. In many cases, a client features a graphical user interface. Note that the client is not responsible for launching the server object. See also *server object*.

client adapter A Bean that serves as a proxy for accessing a remote server object. Drop a client adapter Bean into your project, and you can invoke methods of that Bean to access methods of the server object.

Client Adapter Wizard A Visual Cafe wizard for building client adapters.

Client Adapters folder	The default destination for JARs created in the Client Adapter Wizard. Its name is <code>ClientAdapters</code> and it is found within the installation folder.
client computer	The computer system where you are running a client program (contrast with <i>development computer</i> , <i>servant computer</i> , <i>remote computer</i>).
codebase	<p>In HTML, there is a <code>CODEBASE</code> attribute on the <code>APPLET</code> tag that specifies the directory in which the <code>.class</code> file for the applet is to be found.</p> <p>In RMI, there is a <code>codebase</code> property that indicates the server's codebase URL where classes are available for clients to download. You can access it from <code>java.rmi.server.codebase</code>.</p>
comm.properties file	<p>A file of attributes that are used in distributed debugging. It is found in:</p> <pre>%JAVA_HOME%\com\symantec\itools\distdbg\agents\comm\comm.properties</pre>
connections file	A file that records the <i>repository connections</i> made by Visual Cafe. It is found in the <code>Config</code> folder of your <i>installation folder</i> .
CORBA	The Common Object Request Broker Architecture, developed by the Object Management Group (OMG), that defines the open-standard architecture enabling communication between objects in a network.
Customizer	A Java Bean customizer, provided to enable the developer to change properties (and, possibly, other attributes) associated with a Bean.

D

Distributed Debugging Services	Tracks debuggable VMs on the network, serves class files to instances of <code>debugvm</code> on other hosts, and lets you start processes on remote hosts from Visual Cafe.
ddservices	<p>An executable that starts the Distributed Debugging Services.</p> <p>The <code>ddservices</code> daemon handles the administration and registration of debuggable VMs. The</p>

	<p><code>ddservices</code> daemon must be started on each physical machine on which a VM will be running. It is an executable on Win32 systems, and a script under UNIX. If you are debugging a particular server machine, it might make sense to make <code>ddservices</code> a startup process on that machine. The <code>ddservices</code> daemon runs without a console window.</p>
debuggable VM	<p>A Symantec version of the Java Virtual Machine that is equipped for debugging.</p>
debugvm	<p>A bootstrap executable that starts a debuggable VM, registers it with the debug registry, and then executes a Java class. Classes run by <code>debugvm</code> are debuggable by Visual Cafe. The <code>debugvm</code> command automatically handles passing the debug password to Visual Cafe.</p> <p>If called with no arguments, it starts a VM to which a debugger can later attach.</p> <p>Usage: <code>debugvm [options] [class]</code></p> <p>where <i>options</i> include:</p> <ul style="list-style-type: none"> -name <i>xxx</i> to specify a name to identify this app -help prints the full set of java options (followed by any other <code>java.exe</code> options)
default ORB	<p>One of the set of ORBs that a Visual Cafe installation keeps record of. The developer can set this default value in the Environment Options dialog box. This determines which Interface Repositories will be visible, and which types of client/servant adapters can be created.</p>
delegation (tie delegation) model	<p>A CORBA implementation model in which two classes are generated from a skeleton file. If an IDL file, <code>soda.idl</code>, is compiled to generate a <code>_sodaImplBase</code> class, the Servant Wizard can generate two files under the delegation model:</p> <ul style="list-style-type: none"> ◆ a tie class (named <code>_tie_soda</code>) that inherits from <code>_sodaImplBase</code>, but delegates all calls to the delegate class

- ◆ the delegate class (named `sodaDelegate`) that implements the IDL-generated `sodaOperations` interface, which defines the IDL functionality

development computer

The computer system where you are running Visual Cafe (contrast with *client computer*, *remote computer*, *servant computer*).

F

factory

A factory (object factory) produces objects. It accepts some information about how to create an object, such as a reference, and then returns an instance of that object.

H

host

A computer that has send and receive access to other computers on a TCP/IP network. Each host has a unique IP address.

host name

The name of the host on which a process (ORB, Naming Service, the Distributed Debugging services, and so on) is running.

I

IDL

Interface Definition Language is a language for declaring the parameters of objects, enabling objects compiled from different languages to freely pass parameters back and forth. An IDL compiler translates from IDL to a target language such as Java or C++, producing skeletons and stubs.

IDL compiler

Reads an IDL file and writes a number of source files in a target language (such as Java or C++). These output files include skeletons, stubs, interface files, and so on.

.idl file

A file of IDL declarations. Defined as part of CORBA, it provides a language-independent description of the data elements manipulated by an interface.

IDL2java compiler	The Inprise VisiBroker IDL compiler is named <code>idl2java</code> and it generates Java code from a file containing IDL specifications. Sun has one called <code>IDLtoJava</code> .
IIOp	Internet Inter-ORB Protocol—a protocol built on top of TCP/IP and used by ORBs to communicate across the Internet.
implementation file	In terms of the Visual Cafe implementation approach, a <code>.class</code> file that performs the tasks of a server, but does not include CORBA or RMI communication facilities. You feed it into the <i>Servant Class Wizard</i> , which creates a <i>servant adapter</i> that provides the CORBA or RMI facilities. The servant adapter and the implementation class executing together constitute a servant.
Implementation repository	Provides a run-time repository of information about the classes a server supports, the objects that are instantiated in that server, and their IDs.
Inprise VisiBroker	One of the ORBs supported by Visual Cafe.
installation folder	The root folder of the Visual Cafe installation, typically named <code>vCafe</code> .
interface file (.idl file or .java file)	A source file containing a description of an interface, either in IDL or in Java.
interface repository	In CORBA, an online database of interface definitions. This repository contains the metadata that lets a component retrieve a list of interface definitions. The component can then ask the ORB or a <code>CORBAService</code> for reference to an object that implements a certain interface.
introspection	The process of discovering a Bean's design-time features by one of two methods: <ul style="list-style-type: none"> ◆ Low-level reflection (through the core reflection API) that uncovers a Bean's features through the <code>java.beans.Introspector</code> class, and storing information into a <code>BeanInfo</code> object. The information

	<p>data such as properties, events, and all the accessible methods.</p> <ul style="list-style-type: none">◆ By examining an associated Bean information (BeanInfo) class that explicitly describes the Bean's features.
invocation (object invocation)	<p>The process of performing a method call on an object. In a distributed application, a local object can invoke a remote object without knowing the object's location in the network.</p> <p>When the interface is known at compile time, static invocation can be used: that is, the caller invokes the client stub which invokes a server skeleton on the remote system. The skeleton then invokes the desired server-side object. When the interface is not known at compile time, dynamic invocation must be used.</p>
IONA OrbixWeb	<p>One of the Object Request Brokers supported by Visual Cafe.</p>
IOR	<p>An interoperable <i>object reference</i>. An object reference that is valid across ORB implementation boundaries.</p>
IOR (Interoperable Object Reference) string	<p>An IOR string allows an application to make remote method calls on a CORBA object. Once an application obtains an IOR string, it can access the remote CORBA object through IIOP. The CORBA object can be implemented with any CORBA 2.0 compliant product that supports IIOP. The application that obtains the IOR string can be developed with a different CORBA 2.0 product. The application constructs a GIOP message and sends it. The IOR string contains all the information needed to route the message directly to the appropriate server.</p>
IP address	<p>An Internet Protocol address, composed of four bytes. Usually seen in the format <i>nnn.nnn.nnn.nnn</i> where each <i>nnn</i> is between 0 and 255.</p>

O

object reference The information needed to specify an object within an ORB. Two ORB implementations may differ in their choice of object reference representations.

ORB Object Request Broker. In the Common Object Request Architecture (CORBA) an ORB enables objects to transparently make and receive requests in a distributed environment. It is the foundation for building applications from distributed objects and for interoperability between applications in hetero- and homogeneous environments.

Sometimes the term “ORB” is used to refer to a specific piece of software executing on a single computer. At other times, it designates a sort of an object bus for CORBA, the transparent pathway mediated across a network by an ORB at the client end and an ORB at the server end.

OrbixWeb See *IONA OrbixWeb*.

P

port An abstraction: a two-byte number that supplements an IP address to specify the particular “mailbox” of a given protocol (such as FTP or HTTP).

properties file A file containing a persistent table of properties. A program can read a set of properties to support user preferences and user customization.

R

reflection A process by which the methods and events of a Bean can be inspected to establish the properties of the Bean, for use by a visual design tool. In order for reflection to work, the Bean must conform to certain design patterns.

Registry Host Name The host name of machine on which RMI Registry is running.

remote computer	A computer system other than the one on which you are running Visual Cafe (contrast with <i>client computer</i> , <i>development computer</i> , <i>servant computer</i>). Visual Cafe communicates with a remote computer by means of its IP address.
remote object	An object that executes on a remote computer.
remote process	A process running on a remote computer.
remotable	Capable of being adapted to run on a remote host as a servant.
repository connection	A definition of a connection to sources to be used by the client and servant wizards. This can be a CORBA interface repository, an RMI registry, or a file folder.
RMI (Remote Method Invocation)	A standard Java mechanism that enables one Java program to invoke another on a different VM, possibly on a different host. A Java program can make a call on a remote object once it obtains a reference to the remote object. It does this either by looking up the remote object in the bootstrap-naming service provided by RMI, or by receiving the reference as an argument or a return value.
RMI interface files	Either a <code>.java</code> file or the <code>.class</code> file compiled from that <code>.java</code> file, defining an interface to a servant.
RMI registry	In RMI, a non-persistent naming service that enables the developer to register and retrieve servants using simple names. The server application, coupled with servants generated by the Visual Cafe Servant Class Wizard, creates a registry if none is running when it executes. A single, stand-alone registry can support all the Virtual Machines on a server node.
rmic	The RMI compiler—a tool that reads a class file representing a server and generates stub and skeleton classes that facilitate the communication between a client and the server through the RMI protocol.

S

.ser file	A file that contains a serialized version of a Bean.
serialization	The mechanism whereby Java provides a framework for propagating persistent objects by inserting object state information into a stream, and for creating new objects of the same type from that information. Java provides the mechanism whereby primitive data types are stored and retrieved, but objects must write their own contents and must contain enough information to reconstruct the object.
servant	See <i>servant instance</i> and <i>servant class</i> .
servant adapter	A Symantec class created by the Servant Wizard that enables a servant to participate in a distributed application.
servant class	The class definition of a servant, which defines exactly how an RMI or CORBA interface is implemented.
Servant Class Wizard	A Symantec tool that helps you create Servant classes that can be used in distributed server applications.
servant computer	The computer system where you are running a Servant (contrast with <i>client computer</i> , <i>development computer</i> , <i>remote computer</i>).
servant instance	The Visual Cafe 3.0 term for a server object—an object that is live and serving requests from clients.
server	A program that responds to requests and provides service to the requesting program.
server application	A Java application that initializes an ORB or an RMI registry, then creates and binds servants (server instances), providing service to the invoking client.
server object	An object that acts as a server, responding to requests. Also termed a “servant instance”.

skeleton	In the context of CORBA and RMI, one of the blocks of code generated by either an <i>IDL compiler</i> or the <i>rmic</i> compiler for the server side of a client-server interaction.
skeleton file	A file containing skeleton code generated from an interface.
socket	A software object that acts as one endpoint for sending and receiving data between computers. It connects an application to a network protocol, commonly TCP/IP.
source	In the context of the Client and Server wizards, an item (such as interface or an implementation) that can be used to create a client adapter or a servant.
stub	In the context of CORBA and RMI, one of the blocks of code generated by either an <i>IDL compiler</i> or the <i>rmic</i> compiler for the server side of a client-server interaction. The remote object reference held by the client points to the client stub, which is specific to the interface from which it was generated and contains the information the client needs when it invokes a method on the server object that was defined in the interface.
stub file	A file containing stub code generated from an interface.
T	
TCP/IP	Transmission Control Protocol/Internet Protocol—the basic communication protocol of the Internet.
template	In the context of Visual Cafe, one of the files generated by the Servant Wizard when it is given an interface. The file is a Java application containing empty methods. You can then complete the code in the methods and compile it to produce an executable servant. In the inheritance approach, the default name of the template file is the interface name followed by “ <code>Impl</code> ”. In the delegation approach, it is the interface

name followed by “Delegate”. You may change these values in the Servant Class Wizard Defaults dialog box, accessed from the Distributed Object tab of the Environment Options dialog box.

tie object

A .java file that implements an interface, but contains no private data and no logic—has just minimal method bodies). It implements a *delegation* model rather than an *inheritance* model. A delegation implementation does not provide any semantics of its own—it merely delegates every call to the real implementation class, which is implemented separately and can inherit (through an interface) from whichever class it wants.

V

VisiBroker

See *Inprise VisiBroker*.

VM
(Java VM)
(Java Virtual Machine)

Virtual machine provided with the JDK (Java Development Kit). The machine contains:

- ◆ Bytecode translator that converts a downloaded binary Java file into instructions that the client machine can execute.
- ◆ Library routines that a Java applet calls.

In debugging, VMs are specified by a sequence of:

- ❖ an IP-address
- ❖ a slant (“/”)
- ❖ the debug password
- ❖ an optional name

You can specify a name for the VM when you start it using the `debugvm` command. If the name is not specified, then the time and date when the VM was started is used as the name.

Examples of VM names:

```
155.64.123.234/ijk 12:03:36:1236 05/Jun/98  
155.64.123.234/ijk FRED
```

For more information, see the Java Web page (<http://java.sun.com>).

W

Win32 (computer) systems	Those computers using the Windows 95, Windows 98, or Windows NT operating systems.
wizard	A series of dialog boxes that guide you through the process of completing a complex task, automating at least some of the steps. You can step forward or backward through the dialog boxes.

I N D E X

A

- Add VM button, 4-22
- Address property, 4-33, 4-37
- Advanced OrbixWeb Settings dialog box, 3-17
- Advanced RMI Settings dialog box, 3-14
- Advanced Visibroker Settings dialog box, 3-23
- Applet parameter
 - OrbixWeb IOR binding, 3-17
 - VisiBroker IOR binding, 3-23
- ApplicationCreatesRMIRegistry property, 2-41
- ApplicationServants property, 2-41
- ApplicationUsesORBOnHost property, 2-42
- ApplicationUsesORBOnPort property, 2-42
- ApplicationUsesRMIRegistryOnPort property, 2-41
- ApplicationUsesVendor property, 2-40
- arrow, meaning in this manual, xi
- Attached VMs list, 4-21
- AutoGenerateCodebaseURL property, 2-42
- Available VMs dialog box, 4-10

B

- binding
 - of servant name, 2-37
 - OrbixWeb, 3-15
 - VisiBroker, 3-20
- bold, meaning in this manual, xi
- Broadcast IP Addresses, A-13
- Broadcast port, A-12
- BroadcastPort property, 4-33, 4-34
- BroadcastTargetInfo property, 4-33, 4-35

C

- call stack, displaying, 4-32
- Choose implementation approach page
 - creating CORBA servant, 2-7
 - Servant Class Wizard, 2-21
- Choose source type page
 - creating CORBA servant, 2-7
 - creating RMI servant, 2-5
 - for Java implementation, 2-26
 - Servant Class Wizard, 2-13

class

- adding to server application, 2-36
- created by Servant Class Wizard, 2-9
- Class Source URL (in Remote Execution Settings dialog box), 4-16
- Classname property, 2-44
- client, defined, 1-7
- client adapter
 - changing the name suffix of, A-6
 - creating, 1-4, 3-3
 - customizing, 3-13 to 3-26
 - default naming convention, 3-7
 - defined, 1-4
 - saving, 2-31, 3-8
 - saving in a JAR file, 2-32, 3-9
 - saving in a project, 2-31, 3-8
 - saving in Component Library, 2-33, 3-9, 3-11
 - saving source file in JAR file, 2-32, 3-9
 - selecting the interface, 3-4
 - serialized instance of, 2-32, 3-10
 - types, 3-7
- Client Adapter Defaults button, A-3
- Client Adapter name suffix field, A-6
- Client Adapter type drop-down list, A-7
- Client Adapter Wizard
 - defaults, A-6
 - introduced, 1-4
 - using, 3-1 to 3-13
- Client Adapters folder, 3-11
- Client/server development
 - background, 1-7
- CodebaseURL property, 2-41
- comm.properties
 - adding VCafe machines to, 4-31
 - Address, 4-37
 - BroadcastPort, 4-34
 - BroadcastTargetInfo, 4-35
 - CommPort, 4-40
 - GetLocalHostReliable, 4-34
 - MulticastSupported, 4-34
 - NetTimeOut, 4-38
 - ProcTimeOut, 4-36
 - ResourceCacheDir, 4-38

- RetryCount, 4-33
- ServiceExecutable, 4-39
- ServiceHasConsole, 4-37
- SocketFactory, 4-40
- VMDateFormat, 4-38
- comm.properties file, 4-12, 4-33 to 4-40
- Common Object Request Broker Architecture
 - See CORBA
- CommPort property, 4-33, 4-40
- Communication port, A-12
- Component Library
 - saving client adapter in, 2-32, 2-33, 3-9, 3-11
- Confirm Attachments group, 4-22
- Connection failed dialog box, 2-19
- connection, adding/deleting/editing, 3-6
- conventions used in this manual, xi
- CORBA
 - defined (Common Object Request Broker Architecture), 1-9
 - servant created from interface, 2-6
 - server application using, 2-4
 - server project template, 2-39
 - sources of information, 1-11
 - typical development steps, 1-10
- CORBA Binding settings
 - OrbixWeb, 3-17
 - Visibroker, 3-22
- COSNamingServiceServer property, 2-42
- creating a client adapter, 1-4, 3-1
- creating a servant, 1-3
- Current CORBA ORB drop-down list, A-2
- Customize implementation class page
 - Servant Class Wizard, 2-21
- Customize output files page, 2-5, 2-30
- Customizer page
 - OrbixWeb client adapter, 3-15
 - VisiBroker client adapter, 3-21
- customizers, 3-13
- customizing output files, 2-30

D

- ddservices, the executable, 4-2, 4-3
- ddservicesw, the executable, 4-3
- debuggable VM
 - defined, 2-1
 - starting, 4-26

- debugger
 - attaching to a VM, 4-11
 - running distributed session, 4-23
 - stopping for exceptions, 4-19
- Debugging Tab, A-10
- debugging VM
 - attaching to, 4-5
- debugvm, the executable, 4-2, 4-3
- Default Binding Method drop-down list, A-3
- Default class name field, A-3
- Default RMI port field, A-3
- delegate class, 2-6, 2-9
- delegation approach, 2-6, 2-21
 - using, 2-23
- distributed application
 - background of, 1-6
 - illustration of complete, 1-5
 - running, 1-4
- distributed debugging
 - attach technique, 4-23
 - comparison of techniques, 4-25
 - configuring, 4-13
 - debug in Waiting VM technique, 4-24
 - executables, about, 4-2
 - introduced, 1-13
 - overview, 4-2
 - project technique, 4-13
 - run in debugger technique, 4-23
 - scenarios, 4-4
 - starting a session, 4-23
 - stopping processes, 4-27
- Distributed Debugging Services
 - installing on UNIX, 4-9
 - installing on Windows, 4-7
 - running on UNIX, 4-9
 - running on Windows, 4-8
 - stopping, 4-3
- distributed development
 - background review, 1-6
 - CORBA without Visual Cafe wizards, 1-10
 - RMI without Visual Cafe wizards, 1-9
 - with Visual Cafe wizards, 1-11
- Distributed Object tab, A-2
- documentation set for Visual Cafe Enterprise, xii

E

Edit Base Class List, A-8
Enterprise Rapid Application Development, steps of, 1-12
exceptions that stop the debugger, 4-19
Exclude other attachments (in Security Settings dialog box), 4-17
ExportIOR property, 2-45
ExportNamingService property, 2-44

F

File parameter
 OrbixWeb IOR binding, 3-17
 VisiBroker IOR binding, 3-23
Finish button
 Review Options page, 2-35
 Save Client Adapter page, 2-32
 Select interface page, 2-20
folder, used in place of “directory” in this manual, xii
fonts, their meanings in this manual, xi
Free BSD JDK version, adding platform support for, 4-28

G

GetLocalHostReliable property, 4-33, 4-34

H

hard disk space required for Visual Cafe
 Enterprise, xiv
Help menu, xiii

I

IDL compiler default arguments field, A-5
IDL preprocessor commands field, A-5
IIOPPort property, 2-43
Impl class, 2-9
ImplBase class, 2-6, 2-9
implementation approach to servant creation, 1-3
implementation class, 2-21, 2-22
 selecting, 2-27
Implementation file requires modification message, 2-29
implementation, creating servant from, 2-26

In JAR file checkbox
 on Save Client Adapter page, 2-32, 3-9
In Visual Cafe project checkbox
 on Save Client Adapter page, 2-31, 3-9
Include serialized instance checkbox
 on Save Client Adapter page, 2-32, 3-10
inheritance approach, 2-6, 2-21
Insert Servant dialog box, 2-36
Installation location field, A-5
interface
 approach to servant creation, 1-3
 creating servant from, 2-14
 updates, 2-39
Inter-operable ORB reference binding
 OrbixWeb, 3-17
 VisiBroker, 3-24
Introduction page
 Client Adapter Wizard, 3-3
 creating CORBA servant, 2-7
 creating RMI servant, 2-5
 restoring, 2-12, 3-3
 Servant Class Wizard, 2-12
IORFilePath property, 2-45
IP address, A-12
IP addresses, broadcasting, A-13

J

JAR
 saving client adapter in, 2-32, 3-9
 saving client adapter source files in, 2-32
JAR location field, A-6

L

Linux JDK version, adding platform support for, 4-28
LocalHostName property, 2-43
LogCalls property, 2-42

M

Manage CORBA ORBs, A-4
Manage Repository Connections dialog box
 in Servant Class Wizard, 2-17
 opened because connection failed, 2-19
middleware, defined, 1-7
MulticastSupported property, 4-33, 4-34

multiple registries, 4-12
multiple VMs, attaching to, 4-26

N

Name field, A-4
Naming Service binding
 OrbixWeb, 3-19
 VisiBroker, 3-25
NamingHost property, 2-43
NamingIP property, 2-43
NamingPath property, 2-45
NamingPort property, 2-43
NamingRepositoryPath property, 2-43
NamingRootName property, 2-43
NetTimeOut property, 4-33, 4-38
network
 configuring for debugging, 4-12
 connections, verifying, 4-10
Network delay timeout, A-12
Network Settings button, Debugging tab in
 Environment Options, A-10
Network Settings dialog box, A-11
Never (Confirm Attachments) setting, 4-22
New Client Adapter menu item, 3-3
New project location field, A-6, A-9
New Servant Class menu item, 2-5, 2-7
 starts Servant Class Wizard, 2-12
Next button
 Select interface page, 2-20
Next button, on Save Client Adapter page, 2-32

O

Object Management Group, agency that defines
 CORBA, 1-9
Online Help, xiii
ORBagentHost property, 2-42
ORBagentPort property, 2-43
OrbixdPort property, 2-43
OrbixWeb client adapter
 customizing, 3-15 to 3-20
output files, customizing, 2-30

P

parameter marshalling, defined, 1-8
Password (in Security Settings dialog box), 4-17
Port number field, A-5

port, configuring for debugging, 4-12
Present Warning Dialog (Confirm Attachments
 setting), 4-22
process factory classes
 about, 4-30
 defining, A-13
Process name (in Remote Execution Settings
 dialog box), 4-15
Process start timeout, A-12
ProcessFactory
 property comm.properties
 ProcessFactory property, 4-39
ProcessFactory property, 4-33
ProcessFactoryImpl class, 4-28
ProcTimeOut property, 4-33, 4-36
Product field, A-4
project
 saving client adapter in, 2-31
project location, setting, A-6, A-9
Project Options dialog box
 Debugger General, 4-18
 Debugger tab, Exceptions, 4-19
 Debugger, Distributed, 4-20
 Project tab, 4-13, 4-14
project technique, 4-13
properties file
 location set in Environment Options, 2-19
 ServerApp, 1-4

R

RAM required for Visual Cafe Enterprise, xiv
remotable object interface, defined, 2-3
Remote Execution Settings dialog, 4-15
Remote Method Invocation
 See RMI
Remote Procedure Calls, 1-7
Remote processes, running, 4-22
remotifying a class, 2-28
Repository connections file name field, A-3
repository types used by Visual Cafe, 2-17
repository, connecting to/refreshing, 3-6
ResourceCacheDir property, 4-33, 4-38
RetryCount property, 4-33, A-12
Review options page, 2-8
 Client Adapter Wizard, 3-12
 Servant Class Wizard, 2-33

RMI

- defined, 1-7
- development steps, 1-9
- registry, special requirements, 2-45
- servant created from Java application, 2-5
- server application using, 2-4
- server project template, 2-39
- sources of information, 1-9

RMI client adapter, customizing, 3-13 to 3-15

RPC, 1-7

Run on host (in Remote Execution Settings dialog box), 4-15

S

Save Client Adapter page

- in Client Adapter Wizard, 3-8
- in Servant Class Wizard, 2-31

seamless stepping, 4-32

Security Settings dialog box, 4-16

Select Component Library folder page, 2-33, 3-11

Select implementation class page

- creating RMI servant, 2-5
- using, 2-27

Select interface page

- creating CORBA servant, 2-7

Select interface page, in Servant Class Wizard, 2-14

Select New Repository Type dialog box, in Servant Class Wizard, 2-17

Select Servant page, 3-4

serialized instance, saving client adapter as, 2-32, 3-10

servant

- completing, 2-35
- creating, 1-3, 2-10
- creating from an implementation, 2-26
- creating from an interface, 2-14
- executing, 2-37
- selecting, 3-4

Servant Adapter name suffix field, A-9

servant adapter updates, 2-39

Servant Class Wizard

- chart, 2-10
- classes created by, 2-9
- defaults, A-7

starting, 2-12

using, 2-10

Servant Defaults button, A-3

servant record application properties, 2-44

Servant Type drop-down list, A-9

ServantName property, 2-44

server application

- adding a class to, 2-36

defined, 1-12

inserting a servant instance, 2-28

introduced, 1-3

project, about, 2-3

project, creating, 2-4

server development, about, 2-3

server project templates, 2-39

server, defined, 1-7

ServerApp.properties file, 1-4, 2-39

ServerAppPropertiesFormat property, 2-41

ServerName property, 2-44

ServerTimeout property, 2-43

ServiceExecutable property, 4-39

ServiceHasConsole property, 4-37

ServicesExecutable property, 4-33

ServicesHasConsole property, 4-33

Show Available on Debug (Confirm Attachments setting), 4-22

skeleton, defined, 1-8

Skip this page in the future checkbox

- Client Adapter Wizard, 3-3
- Servant Class Wizard, 2-12

socket factory classes, 4-29

Socket implementation, A-12

socket protocol, 4-11

SocketFactory property, 4-33, 4-40

sources

- icons representing, 2-14
- in Select Interface tree (Servant Class Wizard), 2-14
- in Select Servant tree (Client Adapter Wizard), 3-4

Stop execution (in Remote Execution Settings dialog box), 4-16

String parameter

- OrbixWeb IOR binding, 3-17
- VisiBroker IOR binding, 3-23

stub, defined, 1-8

T

TCP/IP, for client/server communication, 1-7

template

- server project, 2-39

Template implementation class name suffix field,

- A-9

tie class, 2-6, 2-9

TIENAME property, 2-44

U

updating

- interfaces, 2-39

- servant adapters, 2-39

Use Seamless Stepping checkbox, Environment

- Options setting, A-11

UseCodebaseOnly property, 2-42

UseIIOP property, 2-43

User name (in Security Settings dialog box), 4-17

V

vendor-specific binding

- OrbixWeb, 3-15

- VisiBroker, 3-21

VisiBroker client adapter, customizing, 3-20 to
3-26

Visual Cafe, documentation set, xi

VM

- adding, 4-22

- attaching to multiple, 4-26

- stopping, 4-27

VM arguments (in Remote Execution Settings
dialog box), 4-16

VMDateFormat property, 4-33, 4-38

W

When attaching to process drop-down list,

- Environment Options setting, A-11