

Gene Cronin
Terence P. Sherlock



Digital
Press

COM **Beyond** **Microsoft**

Designing and
Implementing COM Servers
on Compaq Platforms

O
P
E
R
A
T
I
N
G

S
Y
S
T
E
M
S

COM Beyond Microsoft

Designing and implementing COM servers on Compaq
OpenVMS and Tru64 UNIX platforms

COM Beyond Microsoft

Designing and implementing COM servers on Compaq
OpenVMS and Tru64 UNIX platforms


Terence Sherlock, OpenVMS
Gene Cronin, Tru64 UNIX



**Digital
Press**


Boston Oxford Auckland Johannesburg Melbourne New Delhi

Copyright © 2000 Compaq Computer Corporation
Digital Press is an imprint of Butterworth-Heinemann.

 A member of the Reed Elsevier group

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

 Recognizing the importance of preserving what has been written, Butterworth-Heinemann prints its books on acid-free paper whenever possible.



Butterworth-Heinemann supports the efforts of American Forests and the Global ReLeaf program in its campaign for the betterment of trees, forests, and our environment.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is subject to change without notice.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Compaq or an authorized sublicensor.

COMPAQ, VAX, VMS, the Compaq logo, and the DIGITAL logo registered in U.S. Patent and Trademark Office. ACMS, ALL-IN-1, Alpha, AlphaServer, DECforms, PATHWORKS, and Tru64, are trademarks of Compaq Computer Corporation.

The following are third-party trademarks:

ActiveX, Microsoft, MS, MS-DOS, Visual C++, Win32, Windows, and Windows NT are registered trademarks, and NT, Windows 95, and Windows 98 are trademarks of Microsoft Corporation.
Adobe, Display PostScript, and PostScript are registered trademarks of Adobe Systems Incorporated.
Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc.
Motif, OSF/1, UNIX and the "X" device are registered trademarks, and IT DialTone, X/Open, and The Open Group are trademarks of The Open Group in the United States and other countries.
Unicode is a registered trademark of Unicode, Inc.
Wind/U is a registered trademark of Bristol Technology, Inc.

All other trademarks and registered trademarks are the property of their respective holders.

Library of Congress Cataloging-in-Publication Data

Cronin, Gene, 1948–

COM beyond Microsoft : designing and implementing COM servers on Compaq
OpenVMS and Tru64 UNIX platforms / Gene Cronin, Terence Sherlock.
p. cm.

ISBN 1-55558-226-5 (pbk. : alk. paper)

1. COM (Computer architecture) 2. OpenVMS. 3. UNIX (Computer file) 4.

Client/server computing. I. Sherlock, Terence, 1954– II. Title.

QA76.9.A73 C72 2000

005.7'13769--dc21

00-031434

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

The publisher offers special discounts on bulk orders of this book.

For information, please contact:

Manager of Special Sales

Butterworth-Heinemann

225 Wildwood Avenue

Woburn, MA 01801-2041

Tel: 781-904-2500

Fax: 781-904-2620

For information on all Digital Press publications available, contact our World Wide Web home page at:

<http://www.bh.com/digitalpress>

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Contents

List of figures	x
List of tables	x
Preface	xiii
Part I: COM on Compaq Platforms	1
Chapter 1. Introduction	3
What is COM?	3
The evolution of application programming.....	3
Microsoft's object-oriented component implementation: COM	5
Basic COM terms and concepts.....	6
A component: methods and parameters.....	6
Connecting to other components: interfaces	7
Making components real: creating classes.....	8
Keeping track: globally unique identifiers (GUIDs) and class IDs (CLSIDs)	9
Registering components: DLL or EXE	9
Distributing components: putting the "D" in DCOM	10
Why distribute components?.....	11
Putting it all together.....	13
Example: Flight reservations (IReservation).....	13
Example: Ticket auction (IAuction)	14
Why implement COM on Compaq servers?	16
Benefits of COM	16
Summary	18
Chapter 2. COM as middleware	19
What is middleware?	19
Remote procedure calls (RPCs)	19
Message-oriented middleware (MOM)	20
Transaction processing (TP).....	20
Object request broker middleware (ORB).....	20
Component middleware	20
What does all this mean?	22
Guidelines for choosing a component technology	23
Summary	24
Chapter 3. Comparing COM across platforms	25
COM APIs	25

Microsoft RPC APIs	26
Service control manager (SCM)	26
Registry support.....	27
MIDL compiler	28
ActiveX Template Library (ATL) support.....	28
SSPI with NTLM support.....	28
Threading (MTA and STA).....	29
COM and COM+	29
Summary	29
Chapter 4. Implementing COM.....	31
Designing a new COM application	31
ActiveX Template Library (ATL).....	31
Development tools: Compaq Enterprise Toolkit	31
Making COM objects from an existing application	32
Encapsulating existing applications	32
What is encapsulation?	33
Encapsulation techniques	34
Real-world examples	37
Wrapping vs reengineering	42
Wrapping: the good, the bad, the ugly	42
Encapsulation tools.....	44
Summary	45
Part II: COM for OpenVMS.....	47
Chapter 5. OpenVMS infrastructure	49
A short history of COM on OpenVMS	49
The market and customer context	49
Affinity for OpenVMS Program	50
OpenVMS infrastructure projects.....	51
Implementation challenges	53
Creating a Windows NT environment on OpenVMS	53
Win32 APIs and COM APIs	53
MIDL compiler	53
The rpcss process.....	54
Registry	54
Events	55
Threading model	56

Security	56
Graphical vs character-cell interface	56
Service Control Manager (SCM).....	56
OpenVMS infrastructure architecture.....	57
COM for OpenVMS releases	59
Delivery of COM for OpenVMS	60
Chapter 6. COM for OpenVMS security	61
Starting a COM server securely on OpenVMS	61
Authentication.....	63
Authorization	63
Impersonation.....	64
Authentication, impersonation, and authorization on OpenVMS	64
Authentication, impersonation, and authorization in action: an example	65
Authentication and Credential Management (ACM) authority details	67
Rules for Windows NT authentication on OpenVMS.....	67
Using Windows NT and OpenVMS security together	68
Chapter 7. Getting COM for OpenVMS installed and running	71
Installing COM for OpenVMS.....	71
Configuring and running COM for OpenVMS	71
DCOM\$SETUP	72
DCOM\$CNFG	74
DCOM\$REGSVR32	75
Developing a COM application for OpenVMS.....	76
Step 1: Generate unique identifiers	76
Step 2: Build the app using the MIDL compiler	77
Step 3: Compile the COM app.....	77
Step 4: Link the COM app.....	79
Part III: COM for Tru64 UNIX	81
Chapter 8. Overview of COM on Tru64 UNIX	83
The historical context	84
The strategic context.....	85
The reality.....	86
Implementation details	87
Chapter 9. COM on the Compaq Tru64 UNIX platform	91
The implementation of COM on Tru64 UNIX	91
Implementation challenges	93
Creating and running a server application on Tru64 UNIX	94

Running COM Applications on Tru64 UNIX.....	95
Chapter 10. COM run-time environment on Tru64 UNIX	97
An inventory of the COM run-time environment.....	98
An inventory of the COM programming environment.....	100
The Win32 services on Tru64 UNIX.....	100
Starting and stopping Win32 services from the command line.....	102
Starting and stopping Win32 services automatically	102
The NT daemon (ntd).....	102
The Remote Procedure Call system services (rpcss)	103
Endpoint mapping (coolprip)	103
Network protocols, Windows NT, and Tru64 UNIX	104
Chapter 11. COM for Tru64 UNIX APIs	105
API support with COM for Tru64 UNIX	105
International support.....	109
Chapter 12. The COM registry on Tru64 UNIX	111
Registering COM applications on Tru64 UNIX	112
Creating an application registration file.....	112
Using sermon to add a registration file to the registry	114
Using regsvr to register a shared library in the registry	115
Chapter 13. Security considerations (authentication).....	117
Security.....	117
The security subsystem (paulad and paulas)	118
Password protection between paulad and paulas	119
Remote object activation	119
NTLM pass-through	122
Setting authentication levels with olecnfg	123
olecnfg system-wide registry settings and authentication levels	124
Olecnfg application-specific registry settings.....	126
Chapter 14. Utilities	129
Structured storage functions, stgview and df2t	129
Displaying version numbers with the dcomver utility	130
Chapter 15. Configuring Tru64 UNIX for COM.....	131
Chapter 16. Developing COM applications on Tru64 UNIX	135
Using MIDL to compile an IDL file.....	136
Type libraries	137
Compiling a module definition file with makedef.....	137

Why use C++?	138
Compiling COM executables and shared libraries	139
Required and optional shared libraries	141
Compiler and linker switches and flags	142
Part IV: Appendices	145
Appendix A. The COM demo CD-ROM	147
What does the COM demo do?	148
Installation prerequisites	148
COM for OpenVMS prerequisites	149
COM for Tru64 UNIX prerequisites	153
Windows NT client prerequisites	154
Installing the COM demo software	154
Windows NT installation	154
OpenVMS installation	155
Tru64 UNIX installation	156
Configuring the COM demo software	157
Running the COM demo software	157
Starting the demo	157
Running the demo	160
Building the COM demo software sources	162
Building sources on Windows NT	162
Building sources on Tru64 UNIX	163
Building sources on OpenVMS	163
Possible error messages	164
COM demo installed files	165
Appendix B. Porting 32-bit Windows applications to a 64-bit Tru64 UNIX platform	169
Data type macros	169
Marshalling pointers in embedded structures	170
OXID key size	170
Disabling UNIX permissions mask	171
Wide character encoding and Unicode	171
Marshalling wchar_t data	171
Building a UUID (GUID)	171
inet_addr return value	172
Appendix C. Acronyms	173
Glossary	175
Index	179

List of figures

Figure 1. Monolithic application architecture	4
Figure 2. Client/server application architecture	4
Figure 3. Object-based application architecture.....	5
Figure 4. Component: schematic view	6
Figure 5. Component, method, parameter	7
Figure 6. Interfaces.....	8
Figure 7. Components on the same system.....	10
Figure 8. Components on different systems.....	11
Figure 9. Client/server (two-tier) component deployment	12
Figure 10. Three-tier component deployment	13
Figure 11. Example: making a reservation	14
Figure 12. Example: auctioning an airline seat	15
Figure 13. Encapsulation using a COM server.....	34
Figure 14. "Monolithic" or executable encapsulation.....	35
Figure 15. Functional encapsulation.....	36
Figure 16. Drew Corporation: monolithic application.....	37
Figure 17. Drew Corporation: VP's access.....	38
Figure 18. Connor Collectibles: monolithic order entry	39
Figure 19. Connor Collectables: component order entry.....	41
Figure 20. OpenVMS infrastructure that supports COM for OpenVMS	57
Figure 21. How a COM client starts a COM server on OpenVMS	62
Figure 22. Authentication: the shared user account database.....	66
Figure 23. Persona extensions: Windows NT credentials on an OpenVMS system ...	67
Figure 24. DCOM\$SETUP OpenVMS COM Tools Menu	72
Figure 25. DCOM\$CNFG Main Menu	75
Figure 26. MIDL compiler input and output	77
Figure 27. Inventory of COM for Tru64 UNIX.....	98
Figure 28. Remote activation on Tru64 UNIX	120
Figure 29. COM Demo - GUI client menu	158
Figure 30. DOS windows showing server interaction.....	159

List of tables

Table 1. Comparing COM features on Windows and Compaq platforms	25
Table 2. Delivering OpenVMS and Windows NT integration in waves	51
Table 3. OpenVMS infrastructure projects and goals.....	52

Table 4. Summary of differences	59
Table 5. Files used to generate clients and servers.....	78
Table 6. Summary of COM for Tru64 UNIX releases.....	87
Table 7. The Implementation of COM features on the Tru64 UNIX platform.....	92
Table 8. COM application authentication levels on Tru64 UNIX.....	125
Table 9. Impersonation levels.....	126
Table 10. COM-related environment variables on Tru64 UNIX	132
Table 11. C++ switches for compiling COM applications on Tru64 UNIX.....	140
Table 12. Libraries for COM on Tru64 UNIX.....	142
Table 13. Required compiler flags for COM on Tru64 UNIX.....	143
Table 14. Possible COM demo errors	164
Table 15. Files installed on Windows NT system in C:\ProgramFiles\Compaq DCOM Demo.....	165
Table 16. Files installed on the OpenVMS system in the [.demo] directory.....	166
Table 17. Files installed on the Tru64 UNIX system in the /CpqDcomDemo/src/ directory.....	167
Table 18. Data type macro differences between Windows NT and Tru64 UNIX	169

Preface

Assumptions

We designed this book primarily for people who want to learn about Compaq's implementation of COM (Component Object Model) on OpenVMS and Tru64 UNIX platforms. We've assumed that most of our readers are at least somewhat familiar with COM—probably on Microsoft platforms like Windows NT—so we haven't spent too much time on how COM works. (We did, however, explain the technology of COM as it relates to Compaq servers.) Other books and training courses have covered basic COM material better than we could. Instead, we've tried to present an insider's view of how COM works on OpenVMS and Tru64 UNIX, and explain what's different between the two platforms and why.

How this book is arranged

This book has four parts:

Part I: COM on Compaq Platforms (topics common to OpenVMS and Tru64 UNIX)

Introduction to COM, basic terms and concepts, why implement COM on Compaq platforms; COM as middleware, positioning COM with other types of middleware, comparing COM, CORBA, and Java/EJB; comparing COM across platforms; designing a new COM application, making COM objects from existing applications, using COM to encapsulate existing applications.

Part II: COM for OpenVMS (OpenVMS-specific information)

A short history of COM on OpenVMS, OpenVMS infrastructure, how Compaq implemented COM on OpenVMS; COM for OpenVMS security, authentication, authorization, impersonation; implementation differences; getting COM installed and running on your OpenVMS system.

Part III: COM for Tru64 UNIX (Tru64 UNIX-specific information)

How Windows NT COM was ported to the Tru64 UNIX platform, the implementation of the Windows-based COM services on Tru64 UNIX, the COM run-time daemons and services, COM security on Tru64 UNIX, and COM application development on Tru64 UNIX.

Part IV: Appendices

Contents of the CD-ROM; porting issues; acronyms; glossary; index.

Those interested in The Big Picture might enjoy Part I, which covers issues like encapsulation and COM vs Java. Those interested in implementation conundrums should curl up with Part II or Part III. Those wishing to get grubby with code should go immediately to the CD-ROM.

Getting more information

You can find many books at your local computer bookstore and through the web that describe all aspects of COM and related technologies. We've referenced Rosemary Rock-Evans' *DCOM Explained* (Butterworth-Heinemann, 1998. ISBN 1-55558-216-8) several times in this book; we found it a good starting point for COM beginners.

If you want more detail about "how to" on COM for OpenVMS or COM on Tru64 UNIX, we can wholeheartedly recommend the documentation that accompanies those products (after all, we wrote those books, too). This documentation is also available from the OpenVMS and Tru64 UNIX websites.

Websites

For more information about the OpenVMS or Tru64 UNIX operating systems, see the following websites:

www.compaq.com/openvms

www.compaq.com/unix

For more information about the Compaq COM products for OpenVMS and Tru64 UNIX, see the following websites:

www.openvms.compaq.com/openvms/products/dcom/

www.tru64unix.compaq.com/com/

For information about Microsoft's COM, see the following website:

www.microsoft.com/com

You can find the *Component Object Model Specification* (which documents COM and the COM APIs) at the following location:

www.microsoft.com/com/comdocs.asp

Sending comments

We have an e-mail address to which you can send your carefully crafted comments or occasional rants about the content of this book:

`customerservice@bhusa.com`

Acknowledgments

This book, years in the making, represents the contribution of many talented people around the world, from both the OpenVMS and Tru64 UNIX communities.

Gene Cronin, Tru64 UNIX author

I would like to thank the COM for Tru64 UNIX Engineering group who gave me so much of their time and attention: Frank Hayes, Jim Wilkey, Geeta Aggarwal, Tom Frederick, Wei Li, James Lu, Tina Raspuzzi, Peter Shajenko, Fred Tibbitts, Jim Ferguson, Larry Dobkin, Francine LaValley, Keith Hillyard, Shirley Schneider, Joanna Siegel, and Rosemary Gaskell. I would also like to acknowledge the willingness of my managers to give me the time to write this book: MaryJane Grinham and Charlie Costigan.

Terry Sherlock, OpenVMS author

Thanks to everyone in OpenVMS who contributed to the content of this book. I am especially grateful to the COM for OpenVMS development team: Gaitan D'Antoni, Yehia Beyh, Gaurav Bahadur, and Kevin O'Kelley.

I must acknowledge the long-suffering work of the OpenVMS Infrastructure team: Andy Adams, Brian Allison, Rick Barry, Tim Beaudin, Virginia Boucher, Kassy Boulay, Mark Buda, Chris Chan, Steve Conway, Paul DeStefano, Matt Doremus, Gene Freyberger, Mike Furnanz, Dean Gagnon, Kevin Greaney, Andy Goldstein, Boris Gubenko, John Harney, Kim Hemphill, Doug Hoeger, Teresa Honkala, Steve Kennedy, Larry Kilgallen, Meagan Koes, Jim Lanciani, Scott LePage, L. Mark Pilant, Jim Potter, Jim McAndrew, Don Macintyre, Wayne Morrison, Andy Moskal, Sandy Motyl, Brian Ota, Merle Roesler, Gale Taylor, Barbara Thomson, and Eileen Tucker; and the Infrastructure alumni: Susan Azibert, Colin Blake, Elie Cohen, CJ Coppersmith, Bill Davenport, Denise Dumas, Cathy Foley, Gaurav Gupta, Ric Haskins, Paula Hargreaves, Mike Harvey, Jeremy Hitt, Nitin Karkhanis, Mike Kimmell, Shaun Lin, Ken Matsuda, and Angel Rosario.

I also owe a debt of gratitude to Rich Marcello, Vice President of OpenVMS, who sponsored this book; to CJ Coppersmith, my boss, who supported me in writing this book; to Merle Roesler, my ever-patient editor; and especially to Gaitan and Kevin, the real wizards behind all this technology, who reviewed draft after draft after draft. (I believe I now owe them a few *real* draughts.)

And finally . . .

We both want to thank our friends at Digital Press who made this book real: Phil Sutherland, our most patient publisher; Maura Kelly, our production editor; Jacqueline Brownstein, our proofreader; and Sherri Dietrich, our indexer.

And special thanks...

To Donna, my wife, for her continued support.

—Gene

To Margie: we are all still well at big heart.

—Terry

Part I: COM on Compaq Platforms

This part includes the following chapters:

Chapter 1. Introduction

Chapter 2. COM as middleware

Chapter 3. Comparing COM across platforms

Chapter 4. Implementing COM

Chapter 1. Introduction

What is COM?

COM, short for **Component Object Model**, is an object-oriented (o-o) technology jointly developed by Microsoft and Digital Equipment Corporation (now part of Compaq Computer Corporation) that lets developers create distributed objects. This is a great definition if you know about components and object models; if you don't, it's not very useful. So let me take a step back and provide a little more context.

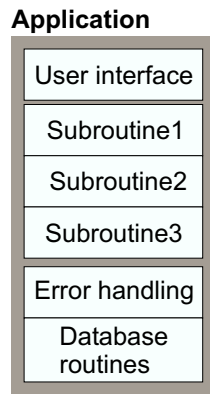
COM is an answer to a question that you might (or might not yet) have asked: *why doesn't somebody come up with an easier way to develop and write {client/server | distributed | multiplatform | heterogeneous environment | object-oriented} applications?* The problems that COM solves—allowing developers to create small, reusable building blocks of high-level language, platform-independent code that hides the complexity of communicating across a network—are the result of gradual changes in computing styles and languages: from one-size-fits-all mainframes to distributed, heterogeneous, *n*-tiered systems supporting all kinds of thin clients; and from assembly language to graphical, object-oriented development environments for reusable components.

This chapter gives a quick history of how we got here, where COM came from, and why Compaq chose to implement COM on its servers.

The evolution of application programming

In the early, time-share computing days, a programmer created an application in one piece—that is, the user interface, subroutines that actually did the application work, database access, and error handling, were all a continuous chunk of code. Because all the resources—application, database, and so on—were on the same system, the application architecture was pretty straightforward, as shown in Figure 1.

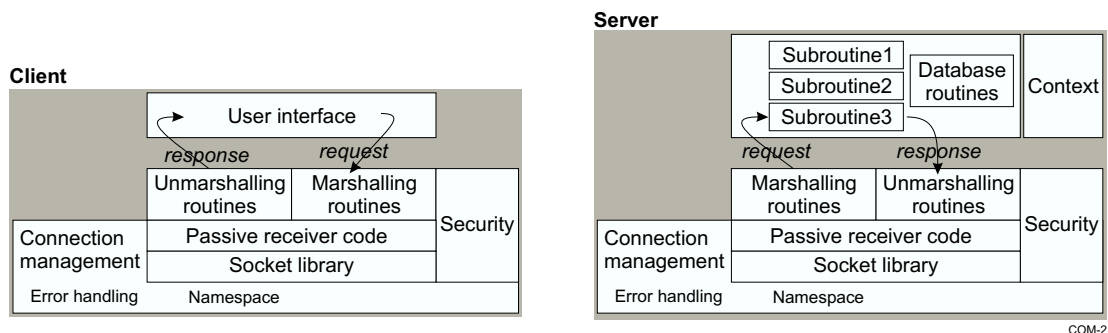
Figure 1. Monolithic application architecture



COM-1

With the introduction of distributed, client/server computing, things got more complicated. Now in addition to coding the user interface and the application's subroutines, the developer needed to think about marshalling and unmarshalling routines, passive receiver routines, a datastore to maintain context, security routines, socket libraries, connection management code, namespace management, and error handling code. Because the application's user interface might reside on a different system from the application and its resources, the developer needed to write (or borrow) a lot of complex code to make sure the plumbing between the client and server pieces worked (bonus points for those who got it to work *correctly*, and double bonus points to those who were able to optimize the connection code). Figure 2 shows a block architecture diagram of a typical client/server application.

Figure 2. Client/server application architecture



COM-2

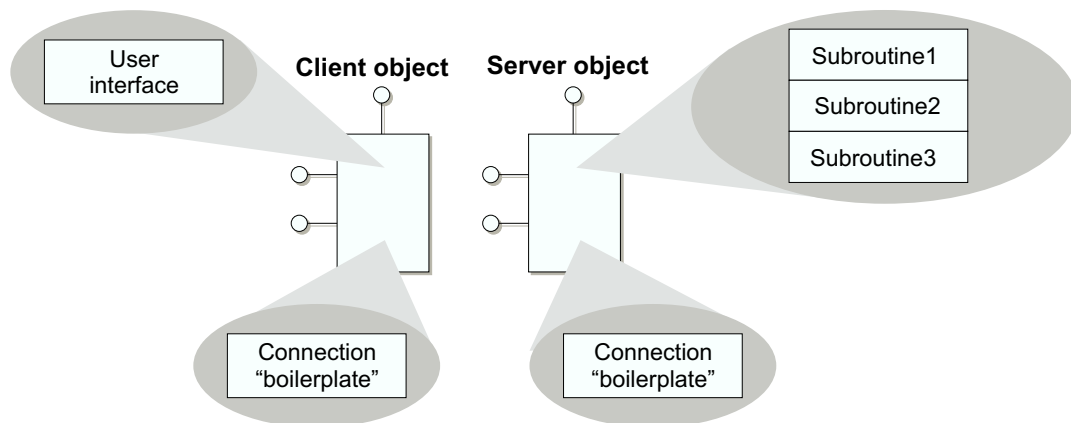
At some point, the industry realized programming at this level was error-prone, not productive, and cost too much. To simplify client/server computing, clever coders

invented RPCs (remote procedure calls), which simplified the process of creating and maintaining connections, and TP (transaction processing) systems, which introduced the idea of transactions and state to client/server systems. To further support client/server computing on distributed (and sometimes disparate) systems, developers began to use an object model architecture.

An *object model* is a programming architecture that allows a developer to define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure. Using this process, the data structure becomes an *object* that includes both data and functions. By constructing programs in this style, a developer could create reusable code objects (or components) that he or she could easily combine with other objects to create large applications. The object model hid the communications complexity in the application's packaging—freeing the developer to concentrate (once again) on the real work of the application.

COM is an implementation of this object model architecture. Figure 3 shows how COM components map to the client/server functions shown in Figure 2.

Figure 3. Object-based application architecture



COM-3

Microsoft's object-oriented component implementation: COM

Technically speaking, COM is an object-based programming model designed to promote software interoperability. That means COM allows two or more components to cooperate (pass requests and data) with one another easily, even if the components are written by different vendors at different times and in different programming languages, or if they are running on different machines with different operating systems. To support its interoperability features, COM defines and implements mechanisms that allow applications to communicate with each other as software objects.

Microsoft first dabbled with component technology in its Windows 3.x product under the name Object Linking and Embedding (OLE). OLE allowed objects or components within the same system to communicate or share data. For example, using Windows 3.1 users could embed an Excel chart in a Word document.

Digital Equipment Corporation (which was acquired by Compaq Computer Corporation in June 1998), a leader in networked computing, recognized the importance of this technology in a distributed environment. In 1994 Digital began working with Microsoft to develop a distributed OLE specification. This specification went through a series of names: NetOLE (Network OLE), DCOM (Distributed COM), and, finally, just plain COM.

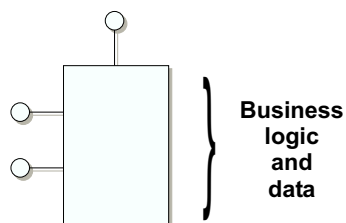
COM implementations are available on Windows NT, Windows 95™, Windows 98, Windows 2000, OpenVMS, and Compaq Tru64™ UNIX®, as well as other UNIX platforms. COM for OpenVMS and COM for Tru64 UNIX implement the Microsoft code that supports the COM draft standards.

Basic COM terms and concepts

Although this book is not an intro to COM, the following sections contain a crash course in some basic terms and concepts you'll need. You can skip these sections if you've worked with COM before.

Designers represent a COM component by a rectangle with some lollipops sticking out, such as you see in Figure 4. The rectangle represents the container (or package) that holds the business logic—the functions or code and data. The lollipops represent the component's interfaces.

Figure 4. Component: schematic view



COM-4

A component: methods and parameters

In object oriented (o-o) terms, an object (or component) is defined by what it does: its functions or methods. For example, you might have an object that has to do with selling

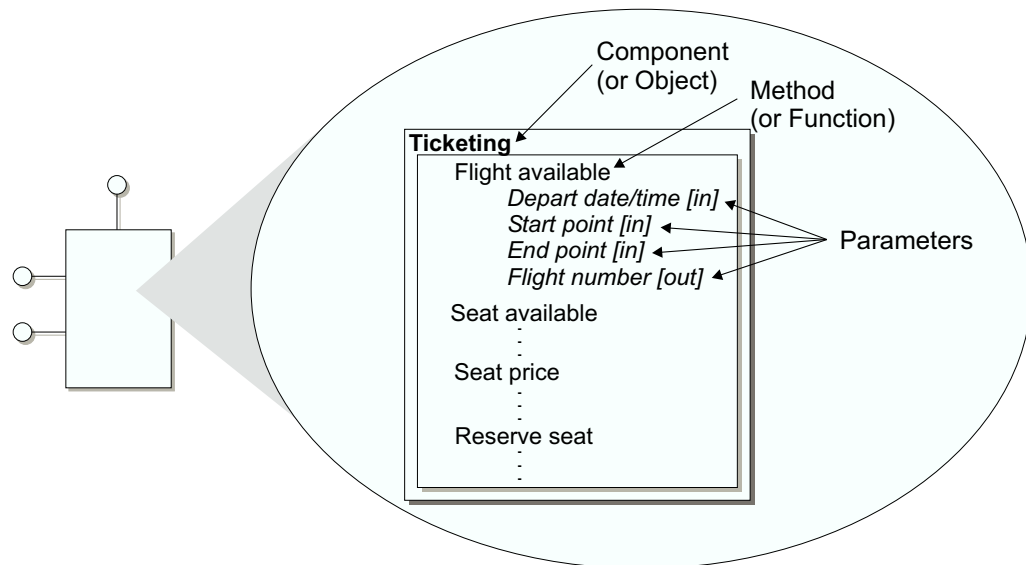
seats on an airplane. That object would have to do some of the following functions (or, in o-o speak, would include the following methods):

- Check that a flight is available.
- Check that a seat is available on a specific flight.
- Check the price of the seat on that flight.
- Reserve a specific seat on a specific flight at a specific price.

Each of these methods requires a few pieces of information, or parameters. For example, to find out whether a flight is available, the method needs to know the planned date and time of departure and the starting and ending points of the flight. When the method has these three pieces of information (input), it can then return another parameter (output): the flight number.

As shown in Figure 5, we now have the beginnings of a ticketing component: we've defined a few methods, and we've defined the parameters for one of the methods. These functions form the core business logic of the component.

Figure 5. Component, method, parameter



COM-41

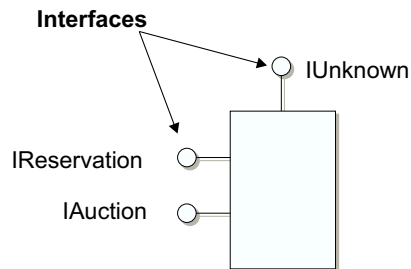
Connecting to other components: interfaces

A single component by itself is not very useful (after all, one *is* the loneliest number). So, to let objects talk to each other, components include interfaces. An interface is the class

plus one or more methods (and the method's associated parameters). (I'll cover classes in more detail on page 8.)

As a naming convention, every interface is designated I“Something”—in Figure 6 the interfaces are IUnknown, IReservation, and IAuction.

Figure 6. Interfaces



COM-42

IUnknown

Every COM component has an IUnknown interface. In schematic representations of components, like Figure 6, the IUnknown interface is always the lollipop that sticks straight out of the top of the component. The IUnknown interface makes the component self-describing. For example, a COM client might need a COM server with the IReservation interface. The client can make a system- or network-wide request to all components through their IUnknown interfaces, asking “Do you have an IReservation interface?” The first component responding to the query handles the IReservation client request.

Other interfaces

In addition to IUnknown, a COM component also has at least one other interface. In schematic representations of components, like Figure 6, these interfaces are the lollipops that stick out of the left side of the component.

If you've used Visual Basic™ (VB), you might also be familiar with the IDispatch interface. IDispatch, used exclusively by VB clients, is similar to (but not as efficient as) IUnknown.

Making components real: creating classes

When you create a component—that is, compile it and turn it into machine code—you have created a class. A class is an o-o term to describe a collection of methods and attributes. In COM, you define the class using MIDL (Microsoft Interface Definition

Language) and tag it with a GUID (described in the next section) to identify the class uniquely.

Keeping track: globally unique identifiers (GUIDs) and class IDs (CLSIDs)

To make sure that a component always connects to the correct corresponding interface and component, you must have some way to identify each interface and component uniquely to everyone in the network. This unique identification is especially important in a distributed environment because many different systems could be generating components with the same names.

To guarantee that each interface is unique, COM includes a facility that generates a globally unique identifier (GUID). A GUID is a 16-byte alphanumeric value that is unique in time and space. It is composed of a time stamp, a network interface card address, and a sequence number. COM software generates GUIDs automatically.

To guarantee that each component is unique, COM generates a class ID (CLSID)—a GUID assigned to a class. You register a class by storing the class CLSID in the system registry. When a client requests that the system create an object, the client uses the CLSID as a way to identify the class.

Registering components: DLL or EXE

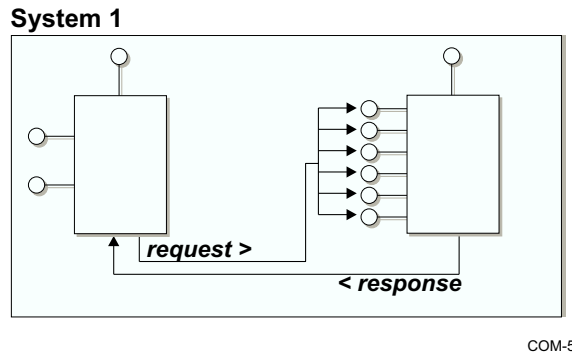
When you create a component, you can specify that component as either an in-process (in-proc) component or an out-of-process (out-of-proc or out-proc) component.

An in-process component, implemented as a DLL (dynamic link library file) on Windows systems, is a separately-linked image with a well-defined set of subroutines that are exposed to the outside world. Any executable image can call a DLL function; when called, the DLL is added to (and eventually deleted from) the user's address space.

An out-of-process component, implemented as an EXE (executable file), is like any other C++ executable program. It contains a main section and calls for the DCOM run-time so it can initialize itself as a component. After it initializes itself, it goes into a wait state until it gets a communication request from a client. You can design the out-proc component in different ways. For example, you can have the server component return to the command line when it satisfies the client request, or you can design the server component to be persistent—that is, when the client request completes, the server returns to a wait state until it receives another client request. In most cases, the rpcss process—not a user—starts an out-of-process server.

Figure 7 shows an example of in-proc components. Both components reside on the same system.

Figure 7. Components on the same system



For example, a word processing application might consist of an editor component (EDITOR.DLL), a spell-checker component (SPELL.DLL), and a thesaurus component (THESAURUS.DLL). A mail application could share the word processing application's editor component and spell-checker component.

When you register a class, the system adds information to two registry keys:

- InProcServer32: records the in-proc DLL name and location
- LocalServer32: records the out-proc EXE image name and location

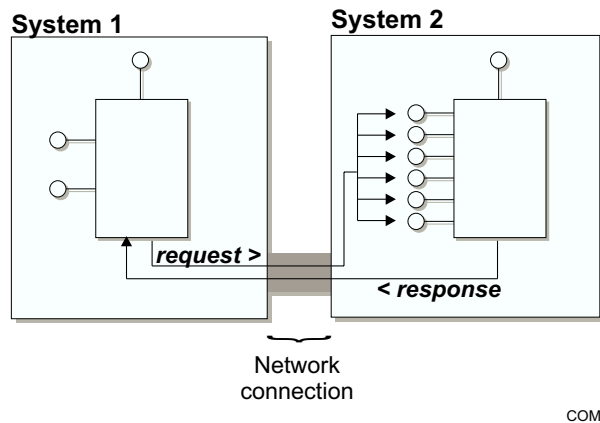
The registry contains a list of all classes available on the system, organized by GUID and by application name. This registry information maps the class to the image (the DLL or the EXE) that the system must activate. The registry also contains the security attribute of the application.

Distributing components: putting the “D” in DCOM

Object models that feature components grew out of structured programming languages and object-oriented languages. Distributed computing, which allows different components and objects that make up an application to be located on different computers connected to a network, is a natural outgrowth of object-oriented programming. Once software developers began creating objects that could be combined to form applications, it was a natural extension to develop systems that allowed these objects to be located physically on different computers.

Figure 8 shows an example of out-of-proc (or out-proc) components. The components reside on different systems. Out-of-proc components could also reside on the same system, but within different processes or address spaces.

Figure 8. Components on different systems



COM-6

For example, a word processing application might consist of an editor component on one computer (EDITOR.EXE), spell-checker component on a second computer (SPELL.EXE), and a thesaurus component on a third computer (THESAURUS.EXE). In some distributed computing systems, each of the three computers could even be running different operating systems.

Why distribute components?

One of the great things about COM is how easy COM makes it to create objects and move them around the network to other platforms. “But,” you might be saying, “why would I want to do that?”

Have you ever created an application on a particular system and then had your boss say, “This app is great, but we really need it on that other system out in Utah.” Or maybe you’ve created a great application that became so popular that everyone wanted to run it at the same time—slowing the system to a crawl.

If you can’t peer into the future and see all the places and people who will want to use your application, you should probably be creating your applications as a collection of components. Why? Because you can move the components around pretty easily and put the parts of the application on the platforms best suited to user or processing needs. Distributing components across one or more systems buys you a good measure of flexibility—an escape hatch, so to speak.

For example, in the case of your app being needed in Utah, maybe your boss doesn’t need the whole app in Utah. Maybe he needs only the user interface logic or the business logic in Utah—the database part can stay on your main system. Or the case where everyone wants to run the application at the same time—maybe they only need access to one or

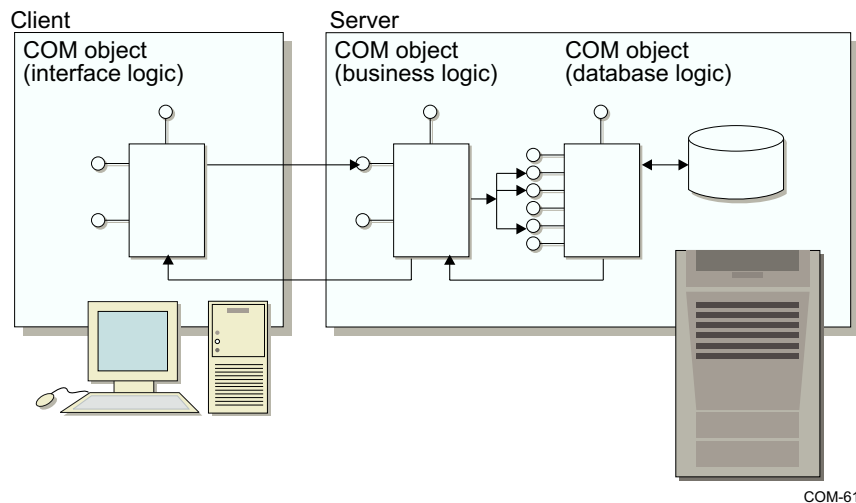
two pieces of the business logic on local servers; the rest of the components can stay on the Big Machine in the back room.

As you create components, you should also be thinking about how you might deploy them. For example, do you have specific database components that you want to keep close to the data? Do you have business logic that you can deploy on departmental servers near your users to improve response time for your users? Do you have mission-critical business logic that you want to keep on your highly available, secure servers? Do you have high-traffic business logic that needs to reside on big iron?

Components in two tiers

In the traditional client/server computing model, the client object (usually the user interface) sits on the client system, and all the business and database logic sits on a server system. Figure 9 shows three COM objects implemented in a client/server model.

Figure 9. Client/server (two-tier) component deployment



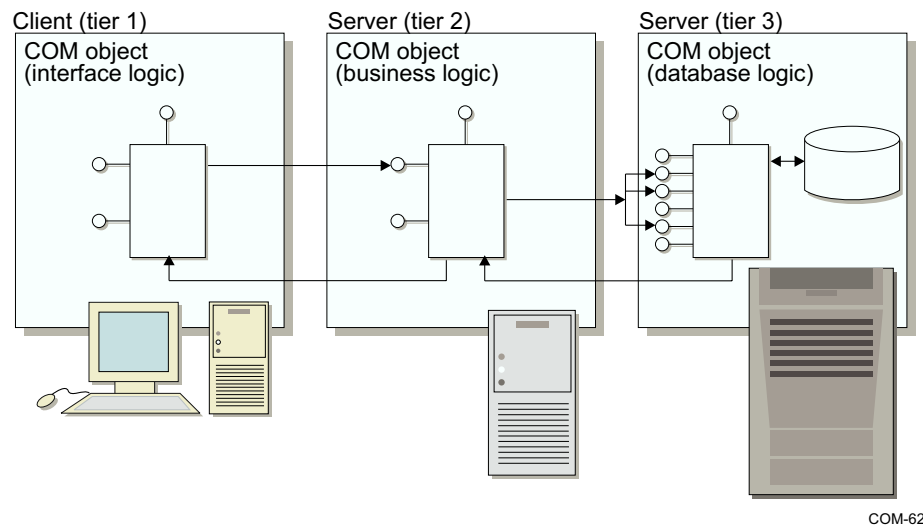
In this example, the client system hosts the interface, and the server system hosts the business logic and database logic. The designers chose to deploy the objects this way to gain the efficiency of keeping the business logic near the database logic, and to take advantage of the server platform's high throughput.

Components in three or more tiers

One of the great things about components is that they're pretty easy to reconfigure and redeploy. If you implement components in two tiers (as shown in Figure 9) and realize later you need three (or more) tiers to get the job done, that's OK. For example, you can

move the business logic components to one or more departmental servers and leave the number-crunching component on some minicomputer in the air-conditioned glass room. Figure 10 shows the same three COM objects from Figure 9 implemented in a three-tier model.

Figure 10. Three-tier component deployment



COM-62

In this example, the client system hosts the interface, the middle-tier server system hosts the business logic, and the third-tier system hosts database logic. The designers chose to deploy the objects this way to allow them to scale the middle tier by adding as many systems as necessary and to keep the database logic and database on a highly available, clustered, scalable server platform.

Putting it all together

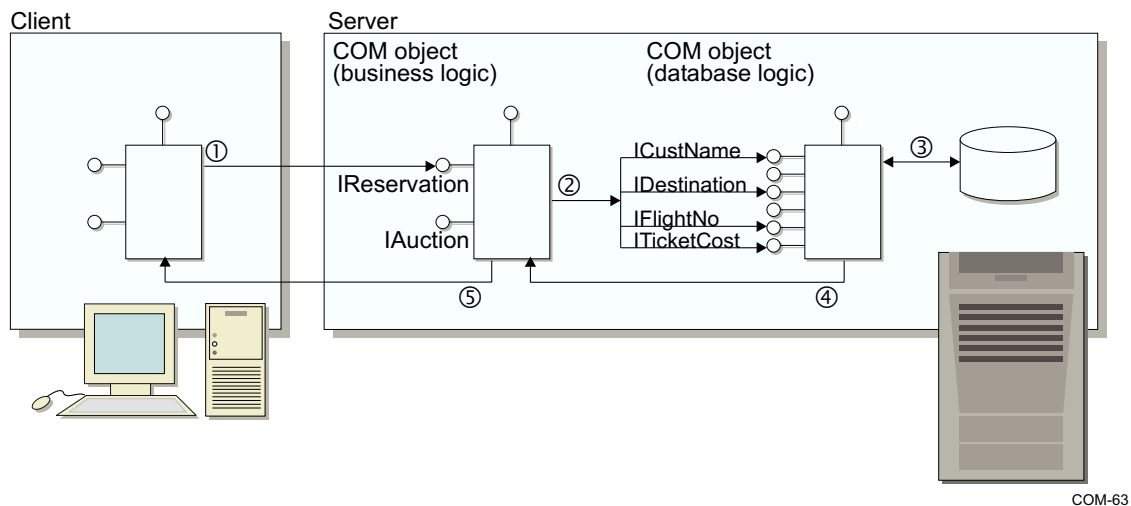
Up to this point, I've talked about COM at the atomic level and given you some general guidelines about designing components. The next two sections provide an example of how you might use the same components to do different things. The first example shows a flight reservation system; the second example shows an online ticket auction.

Example: Flight reservations (IReservation)

Trevor Travel is a charter travel agency that buys blocks of seats on flights for groups. In this reservation example, a member of a charter group reserves one of the seats in a block on a specific flight. Figure 11 shows a few pieces of this flight reservation system. It includes two components: one component with business logic that collects information

from a client through two interfaces, and a second component that contains database logic and controls a database.

Figure 11. Example: making a reservation



COM-63

This example uses the IReservation interface on the business logic component and uses a few of the interfaces on the database logic component. The numbers in Figure 11 correspond to the steps in the reservation process:

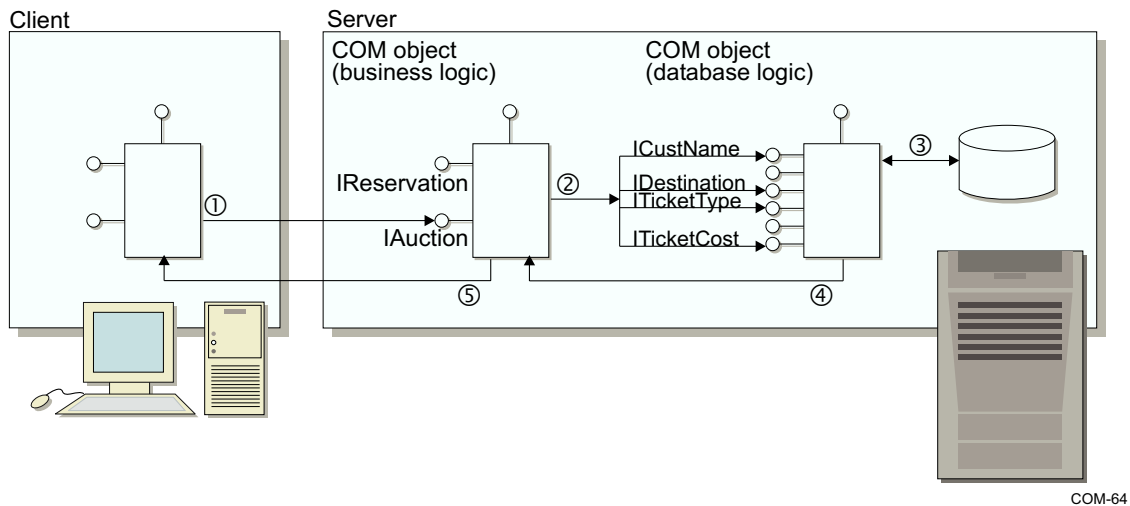
1. The client system—in this case, a terminal in the Trevor Travel office—collects the user's reservation data and makes a request through the IReservation interface. The business logic assembles the data to determine whether the requested seat is available. If the seat is available, the system reserves it.
2. The reservation component passes specific information—customer name, destination, flight number, and ticket cost to the database component through the specific interfaces.
3. The database component checks the specified fields in the database.
4. The database component returns a confirmation that the seat is available.
5. The business logic component passes the confirmed reservation back to the client.

Example: Ticket auction (IAuction)

As I mentioned in the last section, Trevor Travel is a charter travel agency that buys blocks of seats. Sometimes it has leftover seats and wants to dispose of them profitably. In this auction example, Trevor Travel makes these seats available to the general public

through its web-based online auction facility. Figure 12 shows a few pieces of this online auction application. It includes the same two components listed in Figure 11: one component with business logic that collects information from a client through two interfaces, and a second component that contains database logic and controls a database.

Figure 12. Example: auctioning an airline seat



This example uses the **IAuction** interface on the business logic component, and uses a few of the interfaces on the database logic component—similar but not the same as the **IReservation** example. The numbers in Figure 12 correspond to the following steps in the auction process:

1. The client system—in this case, a web client from the Internet—collects the user data and makes a bid on an available seat through the **IAuction** interface. The business logic assembles the data to determine whether or not to accept the bid. If the business logic determines the bid is within an acceptable range, it accepts the bid and reserves the seat.
2. The business logic component checks with the database component: It passes name, destination, ticket type, and cost information to the database component through the specific interfaces.
3. The database component checks the database; the database shows that that seat is available.
4. The database component returns the information to the business logic component.
5. Based on the date, bid price, and number of seats still available, the business logic component accepts the bid and passes the confirmed reservation back to the client.

Why implement COM on Compaq servers?

The short answer is: *to bring the reliability, availability, and scalability of OpenVMS and Tru64 UNIX to your enterprise COM applications.*

If you have Windows NT in your enterprise computing environment, you probably have a love/hate relationship with it. Your developers love the graphical interface and visual tools. You hate the integration (or lack of integration) with other operating systems. Your corporate users love the integrated office suite and e-mail. You hate the word “reboot.” Your accounting department loves the low cost of adding seats and hardware. You hate the additional overhead of managing so many systems. (I’m sure your list goes on for a few more pages.)

The idea behind implementing COM on Compaq server platforms was to combine the best of COM—the visual environment, ease of development, testing, and deployment—with the best of high-end enterprise servers—clustering, 64-bit architecture, consolidation, and connectivity with other systems.

Consider the problem of scaling applications that I outlined briefly in *Why distribute components?* on page 11. Imagine that the database logic and database I described in the third tier was for a world-wide car rental business, or for a stock market, or for a list of organ donors. If you have those kinds of bet-your-business/bet-your-life databases running in the third tier, you want the database to be available all the time, 24/7, 365 days a year, no holidays, no downtime. Or suppose the database is supporting a website that has to respond to tens of thousands or millions of hits per second. You want the database logic and the database running on hardware and software designed to scale to those extreme demands. OpenVMS and Tru64 running on AlphaServers are designed for these demanding, you-don’t-get-a-second-chance applications.

Because COM on OpenVMS and COM on Tru64 UNIX are a direct port of Microsoft’s COM (including the MIDL compiler and Win32 APIs), your developers can keep their familiar Windows development environment. Implementing a COM server on OpenVMS and Tru64 UNIX is as simple as moving the Visual C++ code to the Compaq server platform and running it through the MIDL compiler on OpenVMS or Tru64 UNIX. Your users can keep the comfortable Windows GUI while accessing COM server components transparently on OpenVMS and Tru64 UNIX systems. You and your accounting department can consolidate Windows NT servers while expanding computing power, availability, and reliability with AlphaServers.

Benefits of COM

Throughout this chapter I’ve hinted at some of the good things about COM. The following paragraphs summarize the benefits of the COM model.

Easy to write

Most programmers find COM pretty easy to work with. After you get through the knothole of object-oriented terminology and ideas like methods and interfaces, you'll find coding is easy. You can create components in a variety of languages, but the most common are C++ and Visual Basic.

COM doesn't prescribe the application's structure or content. Instead, it provides a standard way in which component objects communicate (through interfaces).

Takes care of all communication between components

As shown in Figure 2 on page 4, creating and maintaining communication and context between parts of client/server applications in traditional programming languages can be brain-bending stuff. COM does all this work for you, and hides the complexity.

De facto standard

Because Microsoft has defined the COM APIs and created a set of tools and utilities that support those APIs, you can consider COM a *de facto* standard—that is, it's unlikely that COM APIs will change very much because of the investment that Microsoft has already made in tools and supporting infrastructure.

Benefits of component technology

When you choose COM, you get all the benefits of building applications using components including the following:

- **Rapid prototyping**

With COM, you can put the skeleton of an application together quickly: define the methods and interfaces, identify components, and so on. After you have a few components made, you can use these as the basis for creating other components. You can take an existing component and add one or more new interfaces and quickly deploy the new component into your test environment.

- **Simplified testing**

When you make a change to a monolithic application, you have to recompile, relink, and retest the entire application. With components, however, you need to test only the new or changed components themselves. You can do this by building, registering, and immediately deploying the new or changed components in your existing test environment. The new or changed components interact with the existing components. This means at any time you are changing and testing only a small subset of an application's code.

- Modular deployment

When you create an application from component building-blocks, you can deploy the application all at once or in pieces. For example, you might choose to deploy a large application in pieces—with only the application's basic functions available in the first version. You can then expand the application's functions by deploying new or enhanced components as you have time.

With COM you can create a component with ten interfaces. When you deploy the component, you create a client that can access only five of the interfaces. Through time you can expand the client to address more and more of the component's other interfaces.

- Reuse

After you've created a few component applications, you'll find that you have components that you can use for a variety of applications—for example, database routines, user interfaces, and so on. Because these application bits are already self-contained components, you'll find it easy to share them with other applications. With a little planning and an accessible library, you'll find yourself with the beginnings of real code reuse.

Summary

At this point you should have a general understanding of what COM is, why it's important, how it works, and the benefits of implementing COM servers on Compaq platforms. The rest of this book covers *how* to implement COM.

Chapter 2. COM as middleware

What is middleware?

COM, like other component technologies, is technically middleware. Middleware is software “glue” that allows applications on different platforms to communicate with each other and actually do something. The good thing about middleware is that it lets you connect different applications to different operating systems; the bad thing about middleware is that it tends to slow things down. For example, if an application sits right on top of the operating system and uses the operating system’s APIs directly, that application should have good performance. If you add a layer of software that provides a translation function in between that same operating system and that same application, you can bet the application’s performance will suffer. So middleware represents a trade-off: if you want the ability to connect your applications to more operating systems without porting and to create your applications faster and easier, it’s going to cost you efficiency.

There are several kinds of middleware, ranging from RPCs to components. The following sections explain types of middleware and describe how COM fits.

Remote procedure calls (RPCs)

A remote procedure call (RPC) allows an application on one system to call a subroutine on another system. For example, a client application invokes a subroutine on a server.

When you don’t use RPCs (or some other middleware), you have to do most of the work of marshalling the data—that is, ordering commands and information for the remote system and packing everything into optimally sized chunks to send across the wire. After the request gets to the remote system, you have to write code to unpack and order the commands and data for the receiving application.

RPCs are synchronous—that is, the called process has to complete before the calling process can continue.

Message-oriented middleware (MOM)

Message-oriented middleware (MOM) allows you to send information from one operating system or application to another. An example of message passing is a sensor that reports temperature or readings at specified time intervals; a piece of middleware might wrap the sensor unit's message so that it can be understood by a system that drives multiple control units. Examples of message-oriented middleware include Microsoft MSMQ and the IBM MessageQ Series.

Transaction processing (TP)

Transaction processing (TP) middleware allows you to group several tasks that must happen together—such as a money or stock transfer—and guarantees that all the tasks have occurred. For example, suppose you are paying your electric bill automatically using an electronic fund transfer (EFT). In this case, when you authorize the payment, the bank deducts the specific amount from your checking account and credits that amount to the electric company. If the deduction fails for any reason (network failure, power failure, insufficient funds, and so on), you don't want the system to apply the credit to the electric company's account. To be sure this won't happen, the bank uses transaction processing software. In this example the deduction command and the credit command are wrapped in a single transaction; the transaction doesn't complete unless both commands complete successfully. Examples of TP middleware include MTS, BEA Tuxedo, and Compaq RTR.

Object request broker middleware (ORB)

Like an RPC, an object request broker (ORB) allows an application on one system to invoke an application on a different system. The difference between an RPC and an ORB is that an ORB supports an object-oriented programming model, allowing communication between objects, rather than subroutines. When you use an ORB, you have to do less marshalling and unmarshalling because the ORB provides infrastructure that simplifies the communication between well-defined objects. Examples of object request brokers include any CORBA implementation. CORBA is discussed in more detail on page 21.

Component middleware

Component middleware allows you to create distributed objects. You can deploy these objects either on the same system or across several systems and the objects can locate each other and communicate with each other. That is, an object contains enough information to know how to find a required object, to connect to the object, and to pass data between itself and the other object. The objects operate within a common environment or system infrastructure (sometimes called a container) that facilitates object operations.

Examples of component middleware include COM, CORBA, and Enterprise JavaBeans (EJB). The following sections briefly summarize CORBA and Java/EJB (the rest of this book is about COM, so it needs no introduction).

CORBA (Common Object Request Broker Architecture)

CORBA is a set of middleware technology standards from the Object Management Group (OMG), a consortium of eight companies (3Com Corporation, American Airlines, Canon, Inc., Data General, Hewlett-Packard, Philips Telecommunications N.V., Sun Microsystems, and Unisys Corporation) started in 1988. The consortium now includes over 800 members.

OMG was formed to create a component-based software marketplace by hastening the introduction of standardized object software. The organization's charter includes the establishment of industry guidelines and detailed object management specifications to provide a common framework for application development. Conformance to these specifications will make it possible to develop a heterogeneous computing environment across all major hardware platforms and operating systems.

OMG is working to establish CORBA as “middleware that's everywhere” through its worldwide standard specifications: CORBA/IIOP (Internet Inter-ORB Protocol), Object Services, Internet Facilities, and Domain Interface specifications.

For more information about CORBA and the OMG, see the OMG website at:

www.corba.org

Java and Enterprise JavaBeans (EJB)

According to Sun, Java is a “simple, object-oriented, distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded and dynamic” programming language.

You can implement a Java application as a client-side app or server-side app. For comparisons with COM on Compaq platforms, I'll focus on server-side Java and EJB.

The object-oriented facilities of Java are essentially those of C++, with extensions from Objective C for more dynamic method resolution.

The Java language and Java virtual machine (JVM) provide the local execution environment, but Enterprise Java is the platform that extends Java applications across machines and across the Internet.

Users also look to the Enterprise Java services for middleware independence. With the Enterprise Java services, Java applications can be developed independently of the underlying middleware platform. An application can be built for vanilla TCP protocols,

deployed on the web using HTTP, and then reused in conjunction with CORBA or with Microsoft COM, all without application changes.

Enterprise JavaBeans™ (EJB) is the component model of the Enterprise Java APIs. Enterprise JavaBeans allow business applications to be assembled out of reusable, network-ready components (much like GUI components revolutionized client programming). Beans are easily deployed anywhere on the network, easily reused within other business applications running on disparate platforms, and easily managed from a remote console. EJB hides the complexities of making server-side business objects persistent, finding them over the network, securing them, isolating them from sharing conflicts, protecting them from failures, managing their lifecycle, and ensuring their scalability and availability—letting you focus on the business logic and not the infrastructure.

For more information about Java and EJB, see the Sun website at:

`java.sun.com`

What does all this mean?

If you've read this far in this chapter, you're probably pretty serious about component technology. Now you're trying to figure out which technology is right for your company: COM, CORBA, or Java. One of your concerns about these technologies is probably their long-term viability.

When I'm not thinking about, reading about, or playing with technology, I'm reading about what industry analysts think about technology. While I don't always agree with the pundits, I do pay attention when all the big-name prognosticators generally concur on where things are going. In the component area, everybody seems to agree that the market will eventually standardize on two component architectures: COM and Java. The following sections provide a synopsis of current thinking on component technologies.

COM

Developers who are familiar with COM from previous development projects on the Wintel platform will probably choose COM again. Corporations that have identified Wintel as their desktop and departmental computing platform will develop COM expertise and buy COM solutions. Given the number of Windows (and that means COM-enabled) installations throughout the world, this represents an immense installed base.

Java/EJB

Developers who are familiar with o-o applications on larger systems and who are not comfortable with Wintel dominance will probably choose Java. Corporations that are doing web development have been using Java for several years; as Java matures as a

technology, those corporations will be more inclined to buy enterprise Java solutions (JavaBeans) from component suppliers.

And CORBA?

Developers who have big-system o-o development expertise or who are not comfortable with Wintel dominance will probably favor CORBA. Corporations that have CORBA as their object development model will probably adopt Java as well to get a simplified (GUI-based) development environment for their developers.

CORBA faces two significant obstacles:

- CORBA is a *standard*, not an *implementation*. As a result, each CORBA vendor is free to implement CORBA as that vendor sees fit—as long as the vendor adheres to the standard's definitions. This approach has resulted in many CORBA implementations, not all of which interoperate. As a result, you have to choose your CORBA vendor carefully, based on the platforms you need to cover. It also means that adding additional operating system platforms to your CORBA environment later may be difficult.
- CORBA has always been the ORB of choice for big iron because it's fairly complex to implement—that is, it requires very experienced programmers and designers who understand all the pieces (transport layers, sockets, security, and so on) and who can implement them. To a generation of programmers raised on visual development environments and o-o languages that hide connection management, security, and socket library coding, CORBA might appear too difficult to use. (Remember the client/server architecture picture [Figure 2 in Chapter 1]?)

Unless CORBA can overcome these obstacles, it will probably lose market share until it becomes a legacy implementation.

Summary

Software houses (that is, ISVs and VARs) who move into the component business will have to support both COM and Java object models, either with complementary coding or with internal cross-development tools. COM and Java will coexist in the marketplace by necessity.

Guidelines for choosing a component technology

Every business has different computing needs and different computing goals. While it's impossible to list every option in choosing the correct component technology for your company, I offer the following guidelines for your consideration:

- If your environment includes a few business-critical applications that must run across large systems using several operating systems, you should consider a CORBA

solution—as long as you have a stable set of operating systems and a deep bench of CORBA programmers.

- If your environment includes many applications that must run across many operating systems, you should consider a Java/EJB solution.
- If you are a current Windows user who is concerned about outgrowing the Intel platform, you should consider creating applications in COM. If you need to move to a more robust, available, reliable, and scalable platform, you can choose OpenVMS or Tru64 UNIX and bring your COM server applications with you.
- If the main focus of your environment is Windows and OpenVMS only, Windows and Tru64 UNIX only, or Windows, OpenVMS, and Tru64 UNIX only, you should consider using COM for new application development or for encapsulating existing applications.
- If you are an OpenVMS customer who is using Affinity for OpenVMS products to interoperate with Windows, you should consider using COM for new application development or for encapsulating existing applications.

I admit that these guidelines are pretty general and your mileage may vary, but I think they represent a pretty good summary of the strengths and weaknesses of the technologies.

Summary

Component technologies will become increasingly important as the Internet permeates every aspect of computing and life. Components are the building blocks of large, cooperative, distributed computing environments—such as e-business platforms.

Chapter 3. Comparing COM across platforms

To a COM application, the Windows NT COM environment, the OpenVMS COM environment, and the Tru64 UNIX COM environment are all the same. A COM applications programmer will find the same programming interfaces and the same application support for COM servers on all the three platforms. Table 1 lists and compares COM features on Windows NT, OpenVMS, and Tru64 UNIX; the sections that follow Table 1 explain the OpenVMS and Tru64 UNIX implementations in more detail.

Table 1. Comparing COM features on Windows and Compaq platforms

COM features	Windows NT	OpenVMS	Tru64 UNIX
COM APIs	Yes	Yes (except for GUI functions)	Yes (except for GUI functions)
Microsoft RPC APIs	Yes	Yes	Yes
Service Control Manager (SCM)	Yes	Yes	Yes
Registry support	Yes	Yes	Yes
MIDL compiler	Yes	Yes	Yes
ActiveX Template Library support	Yes	Planned	Yes
SSPI with NTLM support	Yes	Yes	Yes
Threading (MTA and STA)	MTA and STA	MTA only	MTA and STA

COM APIs

Both OpenVMS and Tru64 UNIX support all the non-GUI COM APIs. That is, if you can call the COM function on a Windows platform, you can invoke the same function on OpenVMS and Tru64 UNIX.

Compaq chose not to implement any COM APIs that used the Windows GDI because we expect developers to implement only server functions on OpenVMS and Tru64 UNIX—that is, we expect you'll use an OpenVMS or Tru64 UNIX system to do the computational heavy lifting, then use a Windows system to present the results to the user.

COM for OpenVMS uses the Win32 APIs provided by Bristol Technology Inc. These APIs are part of the COM for OpenVMS run-time kit. The COM for OpenVMS developer kit includes all the header files that you need to code to the APIs. For more information about the Win32 APIs supported by COM for OpenVMS, see Chapter 5. For the current list of the supported COM APIs on OpenVMS, see the *OpenVMS Connectivity Developer Guide*, available on the OpenVMS website.

The COM for Tru64 UNIX engineering team worked closely with Microsoft to port COM API support to the UNIX platform. As with COM for OpenVMS, the ported APIs are part of the run-time kit and the header files needed to code Tru64 UNIX server applications are included with the COM developer kit. The COM for Tru64 UNIX, Building and Running Server Applications (found on the COM for Tru64 UNIX website) lists the APIs and their respective header files. See Chapter 11 for a summary of the supported APIs.

Microsoft RPC APIs

On Windows NT, COM translates a method call into a remote procedure call (RPC). This RPC layer is invisible to the COM programmer—COM handles all the RPC interactions. RPC packs and unpacks message data, converts data formats, packs and unpacks buffers for transmission, establishes sessions, and handles network calls and transmissions.

The underlying transport mechanism for COM is RPC (remote procedure call). Microsoft uses its own version of RPC (MS RPC) that is based on The Open Group's DCE RPC specification, but is not built using the DCE RPC code. COM for Tru64 UNIX uses MS RPC; COM for OpenVMS uses DCE RPC. Although the wire protocol is the same for DCE RPC and MS RPC, Microsoft did make some implementation changes. As part of the COM port, Compaq resolved the differences so that COM on Windows NT interoperates with COM on OpenVMS and Tru64 UNIX.

Service control manager (SCM)

The DCOM service control manager (SCM) is a process that maintains a table of running objects (ROT) and launches processes to activate objects when needed. On Windows NT this process is called `rpcss`. The `rpcss` process also provides RPC endpoint mapper functions. This SCM interaction is invisible to the COM programmer.

On OpenVMS, the `DCOM$RPCSS` process provides the ROT and activation services. The DCE RPC daemon (`DCE$RPCD`) provides the endpoint mapper functions. An

endpoint map is a list of network protocols, network addresses, and end points (for example, socket numbers) that each server listens to. For more information about these functions of OpenVMS, see Chapter 5.

COM for Tru64 UNIX implements the Service Control Manager as a daemon called `rpcss` (Remote Procedure Call System Services). In addition to the SCM, `rpcss` also provides the functions of Object Exporter and Running Object Table. The `rpcss` daemon also invokes a privileged process on Tru64 UNIX to provide the functions of an endpoint mapper establishing a communications link between server and client.

Registry support

On Windows NT, the Windows registry is a hierarchical database that stores information about COM applications and general configuration switches that describe how COM runs on a specific system.

OpenVMS includes a Windows-like registry. The OpenVMS Registry has two parts: the registry database and the registry server. The OpenVMS Registry database is a systemwide or clusterwide hierarchical database that stores information about COM applications; the OpenVMS Registry server is a multithreaded, cluster-aware process that controls all registry operations. COM applications can read or write to the OpenVMS Registry using COM APIs or system services. A server management utility allows you to work with the OpenVMS Registry database from the command line.

The OpenVMS Registry is compatible with the Windows registry—that is, you can import and export keys between the databases, and you can use Windows client applications like `Regedt32` to connect to and to edit the OpenVMS Registry. For more information about the OpenVMS Registry, see Chapter 5.

The COM Registry implemented on Tru64 UNIX is a multi-user database containing information about COM applications, including GUIDs, paths, and security specifications. Unlike the Registry on Windows NT, the COM for Tru64 UNIX Registry contains only application information, as a result you can initialize the COM Registry without impacting system operations. You can modify the COM Registry in a number of ways. You can add application information with the `sermon` utility, you can register a DLL with the `regsvr` utility, you can use the `olecnfg` utility to modify COM Registry security, or, if you understand the inner workings of the COM Registry, you can edit it by hand. See Chapter 12 for a full description of the COM for Tru64 UNIX Registry and the tools you use to modify it.

MIDL compiler

On Windows systems, the COM development environment includes an interface definition language compiler to create language-dependent files that can be added into or read by the COM client or server.

Both OpenVMS and Tru64 UNIX include a port of the Microsoft Interface Definition Language (MIDL) compiler. This MIDL compiler is identical to the Microsoft MIDL compiler V3.00.44. For more information about the OpenVMS MIDL compiler, see Chapter 5; for more information about the Tru64 UNIX MIDL compiler, see Chapter 16.

ActiveX Template Library (ATL) support

On Windows systems, COM includes a set of ActiveX templates that integrate with the development environment to help COM developers create COM applications easily.

OpenVMS will support ATL 3.0. The COM for OpenVMS ATL is based on Microsoft's ATL V3.0 implementation. Compaq COM for OpenVMS provides ATL as source code that you include in your application. ATL on OpenVMS requires the Compaq C++ V6.2-016 for OpenVMS Alpha V7.2-1 compiler. For availability information about ATL for COM for OpenVMS, see the OpenVMS website.

COM for Tru64 UNIX supports Version 2.1 of the ActiveX Template Library. As with OpenVMS, ATL on Tru64 UNIX requires the use of the Version 6.2 C++ compiler. For more information about ATL, see the COM for Tru64 UNIX website.

SSPI with NTLM support

On Windows NT 4.0 systems, COM uses the NTLM challenge/response security model to authenticate a COM client that requests a service on a remote system. This SSPI layer is invisible to the COM programmer—COM handles all the SSPI interactions.

OpenVMS implements the full NTLM security model. For more information, see Chapter 6.

COM for Tru64 UNIX implements an NTLM pass-through security model in which user authentication is provided by a paulad daemon on Tru64 UNIX and a paulas service on a Windows NT Primary Domain Controller. The authentication link between Tru64 UNIX and the NT machine is password protected. See Chapter 13 for a full description of user authentication and the use of NTLM pass-through.

Threading (MTA and STA)

On Windows, COM supports two threading models: STA (single-threaded apartment) model and MTA (multithreaded apartment) model. The STA model allows Windows to run legacy applications or new applications that are not thread-safe.

COM for OpenVMS supports only the multithreaded apartment (MTA, also known as free threads) model for application servers. The multithreaded apartment model allows a component to have more than one thread. OpenVMS implemented only the MTA model because server applications need to be thread-safe for efficiency.

COM for Tru64 UNIX supports both the multithreaded apartment model and the single-threaded apartment model.

COM and COM+

With the release of Windows 2000, Microsoft has introduced COM+. Some writers are now calling COM “COM Classic” to distinguish it from COM+. What’s the difference between COM (COM Classic and COM+)?

The big difference is that COM+ supports MTS (Microsoft Transaction Server), a transaction processing engine that enables you to do “two-phase commit” transactions with COM objects. This means you can now use COM objects to handle transactions and you can talk directly to MTS through new COM APIs.

Does COM+ make COM Classic obsolete? No—all the existing COM APIs continue to work. COM+ objects communicate with COM objects. COM+ security protocols work with COM clients and servers using NTLM (for backward compatibility).

Summary

COM is COM. While you will see operating system platform implementation differences, COM functionality and services are consistent across the platforms. Part II and Part III provide specific details about these platform differences.

Chapter 4. Implementing COM

Designing a new COM application

The following sections describe tools and techniques you can use to develop COM applications from scratch.

ActiveX Template Library (ATL)

The ActiveX template library is a set of standard component templates that integrate with the Microsoft Visual Studio development environment to help COM developers create COM applications easily.

Development tools: Compaq Enterprise Toolkit

The Compaq Enterprise Toolkit is a programming environment that extends the capabilities of Visual C++ and Visual Fortran to let you develop, debug, build, and tune applications for OpenVMS and Tru64 UNIX.

You can use Compaq Enterprise Toolkit to code applications on NT and deploy them on Compaq servers. With Microsoft Visual Studio C++ and the supporting functions of the Enterprise Toolkit, you develop, debug, build, and tune C and C++ server applications on NT for the Compaq platforms. The Toolkit's supporting functions include wizards for setting up remote projects (establishing remote and local file sets and access to a remote host), functions for remotely building applications on Tru64 UNIX or OpenVMS, Visual Studio search functions, the Ladebug Debugger with a graphical interface, and access to online documentation.

The Enterprise Toolkit also helps you build efficient applications that take maximum advantage of system resources. Tools such as a Heap Analyzer, Memory Usage Analyzer, Multi-Process Viewer, and Profiler let you monitor your application, identify problems before they become problems for your users, and tune your application's performance.

The Enterprise Toolkit is a set of extensions to Visual Studio that you can use to more easily develop COM code and to support Tru64 UNIX or OpenVMS COM servers. Using the Enterprise Toolkit extensions to Visual Studio, a developer can:

- Use a bundled wizard to define the project. The Wizard collects files that make up the project and the account information you need to access the server.
- Develop code using a combination of Visual Studio editors and code constructors.
- Build the application remotely, that is, transfer any modified files to the Compaq server where the build is performed.
- Use point and click to examine diagnostics and display problematic code in source files.
- Use an HTML Help viewer to access information from Visual Studio, the Enterprise Toolkit, the Tru64 UNIX or OpenVMS operating system, and Windows.

Making COM objects from an existing application

In some cases, you'll want to turn an existing application (or parts of the application) into COM objects. Generally speaking, you have two options: you can either rewrite the existing application code as COM objects, or encapsulate the existing application.

If you decide to rewrite the application, you must first separate the application's user interface from the business logic. You must also consider how you will handle threading in the application. For example, you might approach a rewrite as follows:

1. Review the user interface. Is it a command line or files based interface or a graphical interface? Does the interface call into routines or tools directly?
2. Get the list of subroutines. Does the interface call the subroutines directly? Does the interface collect all the parameters and pass all the parameters and arguments to the subroutine at once, or does the subroutine request additional parameters interactively?
3. Identify the functions within the application. Does each function map to one or more subroutines? Do functions share common subroutines?

If you decide to encapsulate the application, skip to the **Encapsulating existing applications** section.

Encapsulating existing applications

If you plan to write all new COM applications from scratch, consider yourself lucky—and you can skip this discussion. However, if you live in the real world of legacy data and existing applications but want to add jazzy graphical client interfaces to your big-iron

applications, or want to add new component-based functionality to existing monolithic applications, or want to replace those COBOL programs with components gradually and non-invasively, you should read (and re-read) this discussion very carefully.

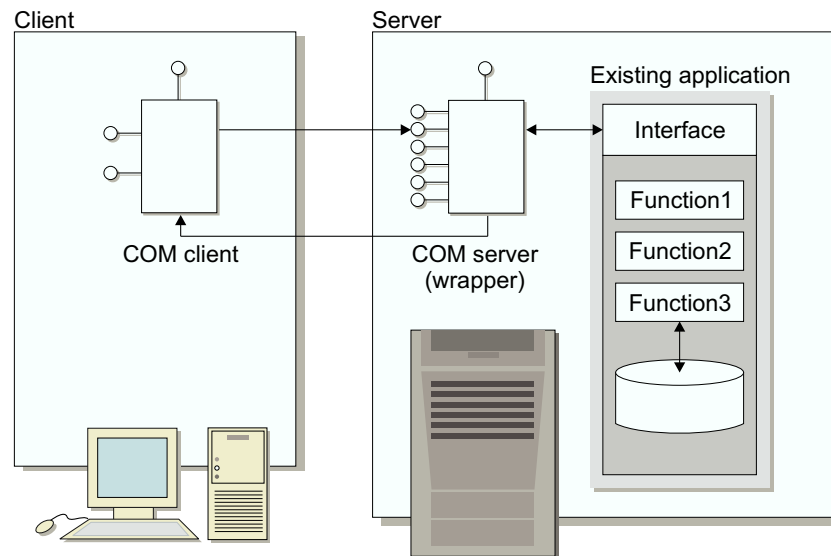
What is encapsulation?

Encapsulation (or *wrapping*) is creating software that replaces, expands, or hides another application's interface or functions. A wrapper or jacket logically surrounds existing code with another layer of software. This new software jacket could, for example, hide an application's existing command line interface and instead make a monolithic application look like it is one or more components.

If you have a monolithic application written in a procedural language (such as Fortran and COBOL) with a character-cell interface, you can put a COM wrapper or jacket around this applications to allow it either to run on new platforms or to remain on OpenVMS or Tru64 UNIX and run in a client/server environment.

On an OpenVMS or a Tru64 UNIX system, a COM wrapper is usually a COM server that acts as an intermediary between a requesting COM client and a monolithic application. The COM server does the work of converting the client request into a form that the application understands (maybe through the application's command line interface), and passing the application's results back to the requesting client in a form that the client understands (for example, as an Excel-readable, comma-delimited table). Figure 13 shows a COM server wrapping an existing application through the application's existing interface.

Figure 13. Encapsulation using a COM server



COM-7

Examples in this book show COM clients and servers in a two-tier (client/server) architecture. You can also implement COM clients and servers in multi-tiered environments—for example, a COM client in the first tier (desktop display), a COM server in the second tier (workgroup server or business logic), and another COM server in the third tier (enterprise server or database logic).

Encapsulation techniques

The next few sections discuss different encapsulation approaches: encapsulating the whole application or encapsulating parts (or functions) of the application.

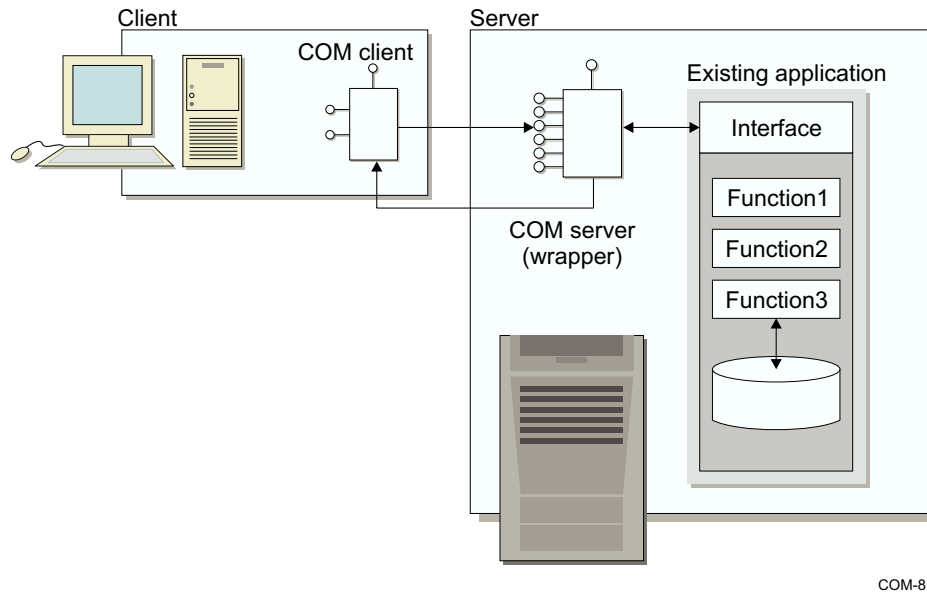
Encapsulating executable files

Every shop has a piece of code that nobody wants to touch because nobody understands how it works, there are no sources, it's fragile, it's not documented, or it's too critical to the business to risk breaking (or all of the above).

Figure 14 shows a monolithic application whose logical parts (Interface, application Function1 and Function2, and database Function3) have been encapsulated by a single COM server. In this example, the single COM server replaces the entire application's input and output. The server accepts requests from a COM client and converts the request data into an input stream that the existing application understands—that is, if the application expects keyboard input from a terminal, the COM server feeds input data to

the application as if the server were a very fast typist. In the same way, the COM server accepts output from the application, reformats it as necessary, and passes it back to the requesting COM client.

Figure 14. “Monolithic” or executable encapsulation



COM-8

You might consider encapsulating executables as a migration strategy: it lets you keep your existing application running, but allows you to move the application to a different platform or later operating system version.

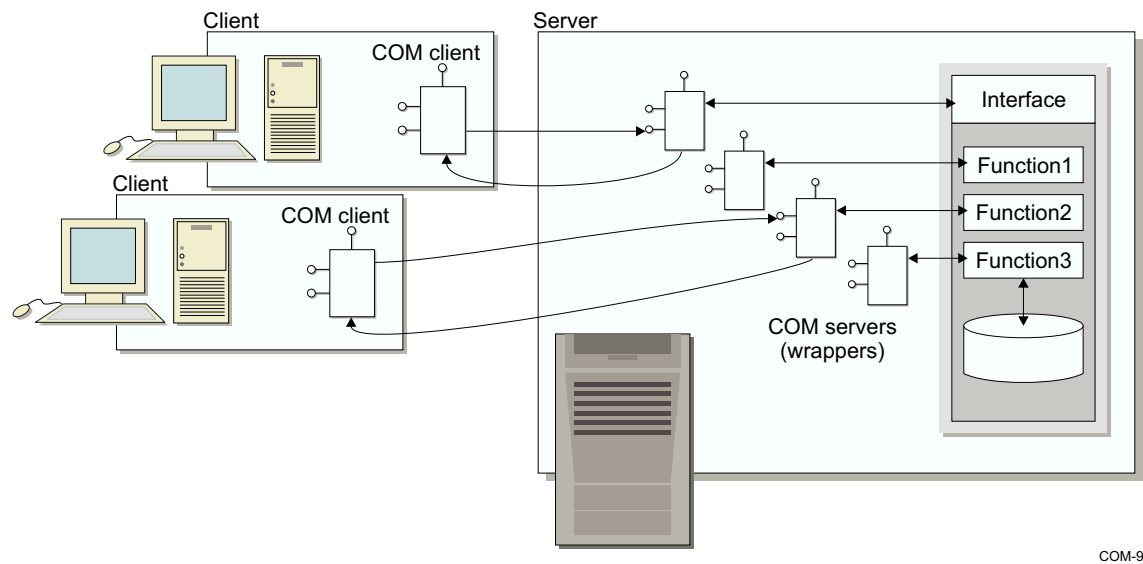
Encapsulating functionality

Sometimes you have an older application that you use only one or two functions of a few times a year. It's not worth rewriting and it works. But it sure would be nice to be able to call that function from a graphical interface or the web. In this case, you want to encapsulate a piece—a single function—of the existing application. This functional encapsulation is relatively easy if you have the application sources or if the application's subroutines are modular and well documented.

Figure 15 shows a monolithic application whose logical parts (Interface, application Function1 and Function2, and database Function3) have been encapsulated by four COM servers. In this example, multiple COM servers replace the application's entry points. Each server accepts requests from COM clients and communicates directly with a specific application function—that is, a client can use the application's computation

function, database functions, or projection function independently of the other application functions.

Figure 15. Functional encapsulation



COM-9

There are several layers of a traditional procedural application that you can encapsulate: the user interface (UI), the database, and the data manipulation routines.

User interface

If you choose to encapsulate the user interface, the UI could be supported on some other platform (for example, from a graphical user interface [GUI] on a Windows NT system).

Encapsulating and moving the UI to the user's desktop can mean that the rest of the application remains on the OpenVMS or Tru64 UNIX server system. Batch processing programs are well suited to user interface encapsulation. Applications that do screen management (for example, SMG or FMS) could have their older character-cell interface encapsulated using COM, providing users access through newer Windows NT style dialog boxes or through a web interface.

Database

If you choose to encapsulate a database using COM, the database could be accessed from parts of a distributed application running on other platforms. The advantage of this approach is that the programmer can keep the database on the server system,

while the user interface and data access routines are on remote (and perhaps less reliable) systems.

Database manipulation routines

If you choose to encapsulate the database manipulation routines, the routines could be accessed from any other COM component in a heterogeneous computing environment.

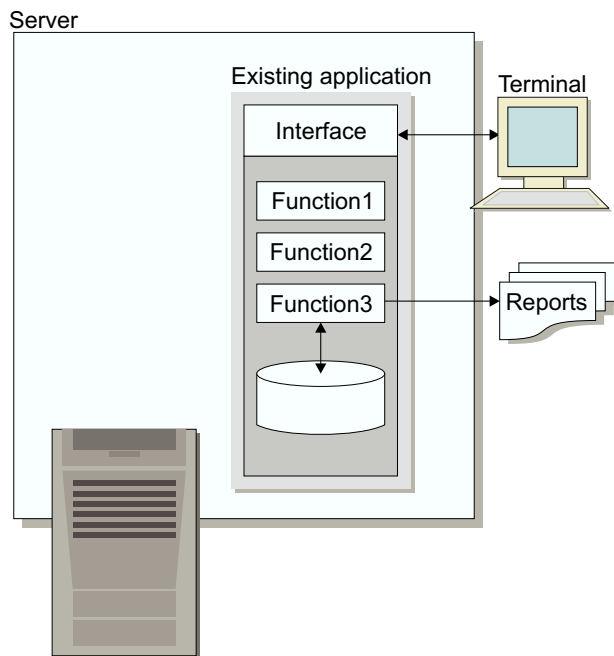
Real-world examples

The next few sections provide short case studies of encapsulation.

Encapsulating an interface: “the VP gets a PC”

The Drew Corporation’s accounting department uses a customer-written application running on an OpenVMS system to handle all the accounts payable and accounts receivable information. Every week, the accounting department generates many reports for the divisions and executives, including a report that lists past-due accounts for the vice president of sales. Figure 16 shows the current application architecture.

Figure 16. Drew Corporation: monolithic application

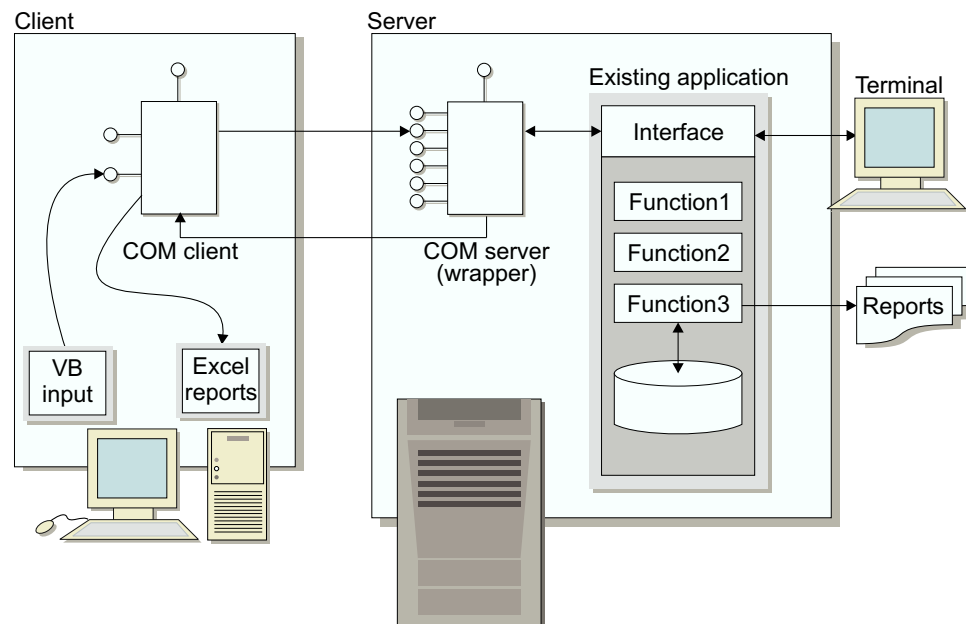


The Drew Corporation wants to move its business to the Internet. As a result, the VP of sales now needs to see the past-due account information daily—and he wants to see it on his PC in Excel spreadsheet form.

The VP's request poses a challenge for the IS department. The accounts payable and accounts receivable programs were written back in the late 1980s by a contracting company that is long gone—along with the source code.

The IS department chose a COM encapsulation solution. The developers first use Visual Basic to create a Windows COM client for the VP's PC. This client is a small program with a few buttons that allows the VP to specify the report start and end dates and provides a launch button. On the OpenVMS system the developers use C++ and COM for OpenVMS to create a COM server that talks to the existing application interface, converting information from the COM client into a format that the existing application's command line interface understands. Finally, the developers add code to the COM server that allows it to read in the ASCII report data and reformat it to a comma-delimited form suitable for reading into Excel. Figure 17 shows this new architecture.

Figure 17. Drew Corporation: VP's access



COM-11

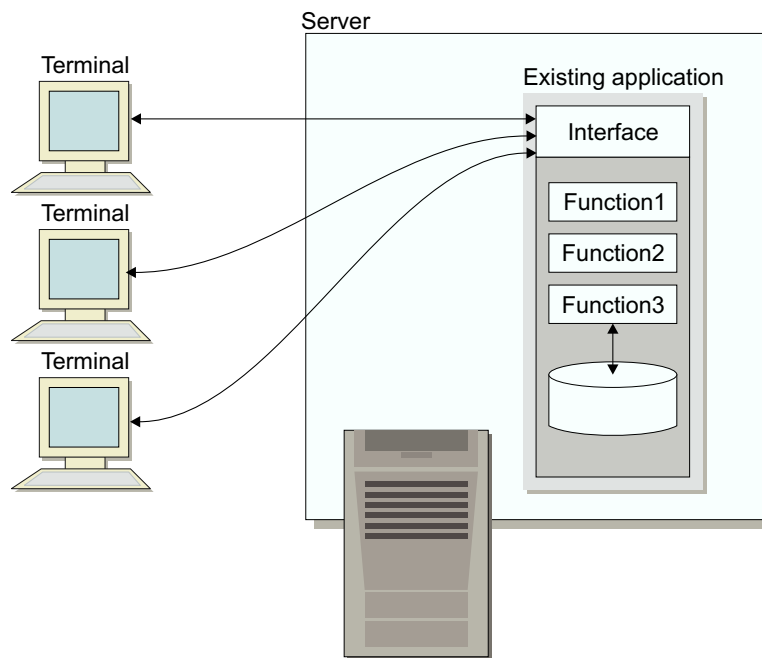
This solution allows the Drew Corporation to continue to use its existing application with no modifications, and it extends the application's accessibility by providing a Windows GUI for the VP to generate and review his report.

Encapsulating functionality: “Oops—we’re too successful”

Connor Collectibles is a mail-order company. Right now, its quarterly advertising catalog is Connor Collectibles’ only contact with its customers. A customer calls the toll-free number with one or more item numbers, and an operator enters the items into Connor Collectibles’ terminal-based order fulfillment system. The order fulfillment system shows customer information, item stats (available, back ordered, or out-of-stock), tracks inventory (removes items from inventory when ordered), tracks prices, creates an invoice, prints a pick/pack list for the warehouse, and generates a customer mailing label. This order fulfillment system and database are home-grown and are constantly updated to add new features.

Connor Collectibles has been growing at 25% per year for the last four years. Over the last year they’ve had trouble keeping up with the volume of telephone orders. At first they solved the problem by adding a few more operators and a few more terminals. Figure 18 shows the current architecture.

Figure 18. Connor Collectibles: monolithic order entry



COM-12

The additional terminals seemed to work well at first, but operators started to notice that more and more items that were supposed to be in stock were appearing on the invoices as back ordered. The inventory confusion was becoming a customer satisfaction issue:

customers didn't like hearing that something was in stock at the start of the order, but then was listed as out-of-stock when the shipment arrived. After some investigation, the IS department found that the item count wasn't updated until an operator completed the entire order. If the first operator took an order that included several items, all the items would show as "in stock" at the beginning of the order. Meanwhile, a second operator might take an order for a single in-stock item. The second order completed ahead of the first, removing the item from stock. If this item was the last one of its kind in the warehouse, the once-available item was now listed as back ordered on the first operator's order.

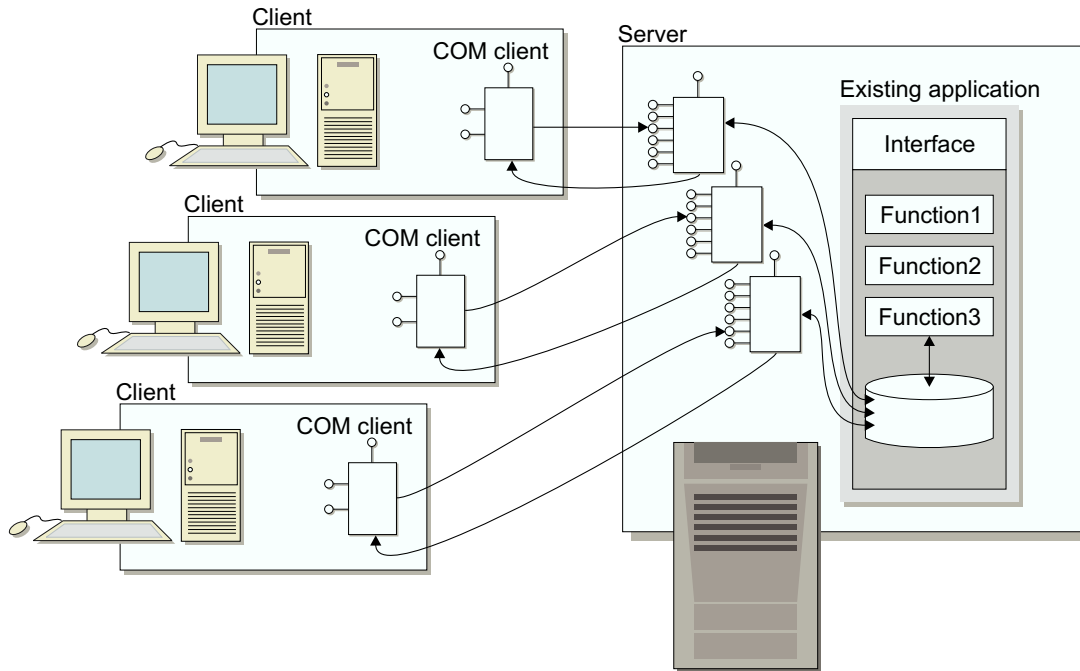
In addition to the inventory tracking problem, Connor Collectibles' operators noticed that all the terminals slowed down during heavy entry loads—another customer satisfaction issue. The IS department noted that they could improve the system performance by adding faster processing hardware, but the software bottlenecks would remain—an important consideration because Connor Collectibles was planning to open an electronic store on the Internet.

The IS department decided they needed to rewrite most of the order entry code, but they absolutely didn't want to touch the customer order database. The IS department redesigned the order entry system as follows:

1. Identified the data collection parts of the system and implemented these as Windows GUI COM clients.
2. Replaced the operator's terminals with PCs.
3. Identified the functional parts of the order entry system (enter customer information, display item availability, build order list, create invoice, print pick/pack list, generate mailing label) and rewrote these as components on the server.
4. Identified the database's API, and wrote a COM server to interface with the existing database.

Figure 19 shows the new architecture.

Figure 19. Connor Collectables: component order entry



COM-13

In this implementation, the COM server component talks directly to the order database. As a customer requests each item and the operator picks the item from a list box, the COM server component locks that item in the database. When the operator commits the entire order, all the ordered items are removed from stock and processed; if a customer changes his mind during the order, the item is released so someone else can order it. This design solves the inaccurate inventory problem. More important, however is the performance improvement: each operator gets his or her own instantiation of the COM server component, so each customer reserves each item in real time.

A big win for Connor Collectables is that they don't need to move the most critical part of their business: the database. Encapsulating the database routines—separating the database functions from the business logic—positions them for a gradual migration to a full component environment. That is, the IS department can start by encapsulating the database logic, then reface the user interface, then turn the business logic (Function1 and Function2 in Figure 19) into components at another time.

This solution addresses not only Connor Collectables current problems, but also positions them to implement the order entry system as part of their planned Internet store. For the Internet store, the user interface changes to a web page, possibly an active server page

(ASP) containing ActiveX objects. These ActiveX objects are simply a different COM client; the COM objects on the server interact with the ActiveX objects COM clients in the exact same way.

Wrapping vs reengineering

At this point you may be saying “Well, if I’m going to go through the trouble of writing all these COM servers to talk with my existing applications, why not just reengineer the whole codebase and do this the right way from the get-go?”

In some cases, reengineering is the correct choice, and the business can tolerate this level of redeployment. However, the risks associated with completely reengineering some older applications are high. Your existing business-critical applications are probably very large—hundreds of thousands to millions of lines of code. If the applications have been around longer than six or seven years, they are probably on their second (or third) generation of maintainers. And, unless you run a very tight development group, the applications might not be well documented. In this case, reengineering the existing application is like opening a war on two fronts: you need to keep the existing application running while you completely redesign the underlying business processes and then implement the new business processes in component code. Few businesses have the programming resources or expertise to maintain existing 2GL or 3GL code while developing and deploying the latest component technology simultaneously.

Encapsulating a legacy application can be less risky than reengineering; you can, in fact, consider wrapping the first step in a rewrite. That is, you can, over time, replace pieces of your existing application while the application itself remains stable and available. You can redeploy fewer developers, allowing you to transition your workforce gradually to component technology and tools. Encapsulation can also give your developers practice in creating reusable components, which can save you time and resources in new component development.

Wrapping: the good, the bad, the ugly

How do you decide if encapsulation is right for you? There are no absolute rules or easy answers. You should consider encapsulation within your overall IS strategy and architecture. For example, if you are considering moving your codebase to object or component technology and are already working in a heterogeneous environment with Windows NT systems, wrapping might be a cost-effective and limited-risk first step—it will allow your development team to experiment with object models without risking the time and cost of a large-scale development effort, and you will be able to provide new features quickly. On the other hand, if you are in a stable, homogeneous systems environment with no need to integrate with other systems, you might find reengineering applications as components a better choice.

The following sections offer some guidelines for choosing or not choosing encapsulation.

The good

You might consider encapsulation if:

- You no longer have the source code for the existing application.
In this case, you have only two choices: rewrite and reengineer the application from scratch (costly and time-consuming), or encapsulate the executable. For all the reasons outlined in the previous section, choose encapsulation.
- You want to migrate to component software slowly.
If your environment is sensitive to change, you can reduce your risk of moving to a component architecture by starting small and learning as you go. You can begin by identifying a well-understood application and encapsulating one or two of the application's functions. As your engineering team gains experience and confidence, they can tackle large applications and write component applications from scratch. This is a conservative and logical approach.

The bad

You should probably *not* encapsulate if:

- You can't afford an increase in code management complexity.
If your development or maintenance engineers are already having trouble keeping up with the existing code, you probably should not add another layer of code on top of what you already have. This new code will probably be in C or C++, and might be unfamiliar to 2GL or 3GL programmers. Don't ask for trouble.
- You might want to modify the underlying code.
If the existing application still has areas of active development (or undergoes frequent extensive maintenance), it's probably not a good candidate for encapsulation. Encapsulating applications that change means that you'll double your work: changing the application and updating the wrappers as well. Don't go there.
- You can't afford a loss of performance.
Encapsulation adds another layer of software on top of what's already there. The wrappers, in the form of servers, will be converting or transforming data. If your systems are already heavily loaded, or if your performance is only marginally acceptable now, encapsulation will probably worsen performance, not improve it. Skip it.

The ugly

Here are two final thoughts about interfaces:

- New interfaces don't improve bad code.

If your existing application is unstable or hard to maintain, you won't improve the application by wrapping it—in fact, the wrapper might make the application's stability or maintenance worse. Look for a solution elsewhere.

- New interfaces can improve old code.

Sometimes all you need (or all your customers want) is an interface-lift. Try it.

Encapsulation tools

An encapsulation tool can help you wrap a large application quickly. An encapsulation tool looks at either an executable file or at the application's source code, identifies entry points to subroutines, and creates additional code (in the case of COM, the tool would create COM objects) that provide another set of entry points (in the case of COM, a set of COM interfaces) to the application. As you might guess, your success with an encapsulation tool depends on how well the original application is structured.

Encapsulation tools can require a lot of handholding to use, and the resulting code sometimes needs human cleanup. For example, it's helpful when you can tell the tool which subroutines you want encapsulated and which ones you want it to skip. If you can't control how the encapsulation tool works, the tool ends up brute-force wrapping the entire application. But brute force is still faster and cheaper than hand coding, and in some cases, encapsulation tools do exactly what you need.

Compaq BridgeWorks

Compaq BridgeWorks is an easy-to-use encapsulation tool that creates reusable, distributable components from an existing OpenVMS VAX or Alpha applications. You can then make these components available to a wide range of desktop and web-based applications. BridgeWorks works with any application called through the OpenVMS calling standard. The application must have callable routines that can be logically mapped to a new front end (Windows- or web-based). If you're not sure which integration method is right for you, BridgeWorks includes an online Knowledgebase to help you decide.

BridgeWorks assumes a three-tier model. BridgeWorks encapsulates applications on an OpenVMS VAX or Alpha system and creates COM business logic for the second tier. You can then create a COM client for the first tier that communicates with the COM business logic on the second tier. The COM objects in the second tier communicate with the encapsulated application in the third tier (the VAX or Alpha application) using RPC.

Compaq BridgeWorks can wrap your application, exposing the application's interfaces as COM components in the second tier. This allows you to keep your application on the same secure third-tier platform where it's always been and provide access to the application's functions through COM interfaces in the second tier.

Summary

You can use COM to create new applications, reface (or encapsulate) existing applications, or to help you move from a traditional programming codebase to a component model gradually. Whatever your requirements, several tools and development environments support COM and can help you speed and simplify your component creation work.

Part II: COM for OpenVMS

This part includes the following chapters:

Chapter 5. OpenVMS infrastructure

Chapter 6. COM for OpenVMS security

Chapter 7. Getting COM for OpenVMS installed and running

Chapter 5. OpenVMS infrastructure

“OpenVMS infrastructure” is the term that OpenVMS engineers use to describe the software plumbing above the kernel that supports COM for OpenVMS. The infrastructure supports more than just COM for OpenVMS—in fact, it touches just about every OpenVMS subsystem—but it’s a convenient collective term to describe all the supporting players in the COM story.

A short history of COM on OpenVMS

Before I explain the *how* of COM for OpenVMS—the technical details of the architecture and subsystems that support it—I’d like to cover the *why*.

At first, you might think that COM for OpenVMS is just another layered software product. It’s not. COM for OpenVMS is the result of years of planning, design, coding, and testing to make integration of Windows NT with OpenVMS transparent to users. COM for OpenVMS marks both the completion of the Affinity for OpenVMS Program and the start of making component technologies available on OpenVMS. The following sections explain the business needs, customer requirements, projects, and what we built along the way.

The market and customer context

In the early 1990s, the love affair between businesses and the low-cost, Windows/Intel computing platform blossomed. Corporate spending on mid-range and large systems dropped as businesses scrambled to put Windows on every worker’s desk.

As the 1990s progressed, Windows systems moved from user desktops into the corporate datacenter. Many companies began deploying low-cost Windows NT systems as departmental servers; smaller branch offices began using NT as their “enterprise” systems. Corporate IT centers suddenly had to integrate NT with their existing systems (which soon became “legacy systems” as NT servers pushed into computer rooms).

At the same time, new and established software developers, ISVs, and VARs shifted their development efforts to the largest-selling software platform: Windows—a “follow the

money” strategy. Updates and new releases of business software on minicomputers and mainframes stretched from months to years, and, in some cases, stopped altogether.

With business applications flooding the desktop and with NT becoming entrenched in the second tier, IT departments started to realize that although Windows was easy to use and had lots of application available, it wasn’t really reliable enough for enterprise use. In addition, users were now clamoring for access to the years of data stored on those legacy systems. The corporate computing centers began looking to minicomputer and mainframe suppliers to help them preserve their business-critical systems and data by integrating their old, reliable big iron with the new Windows systems.

Affinity for OpenVMS Program

Like other minicomputer operating systems, OpenVMS was at ground zero of these changes. Based on Digital’s own experiences and on customer requests, in early 1995 an engineering strategy team began looking at ways to integrate OpenVMS with Windows NT systems. The work of this core team eventually formed the basis of the Affinity for OpenVMS Program.

In May 1995, Digital Equipment Corporation formally announced its Windows NT and OpenVMS integration strategy. As part of its technical strategy, Digital built on the natural affinity between OpenVMS and Windows NT to allow customers to run Windows NT applications in combination with OpenVMS applications and databases, and to extend the functionality, availability, and scalability of OpenVMS to Windows NT while protecting the installed base’s investment in their current hardware and software.

The natural affinity between OpenVMS and Windows NT systems

From the very beginning the Affinity for OpenVMS Program had as its goal *integration* with Windows NT, not just *interoperation*. The engineering strategy team planned COM for OpenVMS as the fulfillment of that goal—that is, bringing the Windows NT applications needed to run an enterprise back to a reliable, available, and scalable computing platform while making critical business data available to Windows desktops.

The OpenVMS engineering team was confident that OpenVMS could achieve a tight integration with NT—after all, Windows NT is a direct descendent of VMS. Dave Cutler, the primary system architect of Microsoft Windows NT (new technology), worked on several operating systems while he was employed by Digital in the 1970s and 1980s. Dave served as the technical project leader for VMS V1.0. After VMS V1.0 shipped, Dave worked on several compilers and another operating system before joining Microsoft. So it’s not surprising that NT architecture and internals bear a strong resemblance to OpenVMS.

Alliance for enterprise computing

In August 1995, Digital and Microsoft announced an expanded agreement that formalized their joint activities to bring Windows NT to the enterprise. As part of the business strategy, Digital entered into a strategic alliance with Microsoft. This alliance facilitated exchange of technology and expertise between the two companies.

Waves of integration

To customers, Affinity for OpenVMS meant a collection of products, services, and operating system enhancements that enabled data and application interoperation. These products and services were delivered in waves; Table 2 lists and describes the major functions delivered in each wave.

Table 2. Delivering OpenVMS and Windows NT integration in waves

	Date	Important features
Wave 1	Fall DECUS 1995	OpenVMS 7.0: 64-bits, VLM, expanded system integration services and support.
Wave 2	Spring DECUS 1996	Application Developer Package, OpenVMS Management Tools, Netscape browser and server.
Wave 3	Fall DECUS 1996	ACMSxp, MAPI drivers, PATHWORKS 32 enhancements.
Wave 4	Spring DECUS 1997	Application Developer Environment enhancements, mail/messaging enhancements, middleware, RTR, ISG Navigator.
Wave 5	Fall DECUS 1997	Java, mail integration tools, system management tools, compilers, PPP.
Wave 6	Fall DECUS 1998	OpenVMS 7.2, COM for OpenVMS, Advanced Server, Solution sets.

As you can see from Table 2, the important features for each wave focus on 24x356 computing—extending the unlimited high end—and on seamless integration with the Windows environment. With each wave, the integration between OpenVMS and Windows became tighter: from connecting, to sharing data, to sharing applications.

OpenVMS infrastructure projects

To OpenVMS engineering, Affinity for OpenVMS meant a series of projects that focused on changes and enhancements to the OpenVMS infrastructure to enable COM for OpenVMS. Table 3 lists and describes the main infrastructure projects.

Table 3. OpenVMS infrastructure projects and goals

Project	Goal
Authenticated RPC	Extend DCE RPC on OpenVMS to interoperate with the Windows NT implementation of MS RPC.
SSPI (Security Support Provider Interface)	Implement MS SSPI on OpenVMS.
Single sign-on	Extend the OpenVMS LOGINOUT functions by adding ACMEs.
OpenVMS Registry	Create a Windows NT connectable registry on OpenVMS.
OpenVMS Events	Provide Windows NT events logging for COM events on OpenVMS.
Win32 APIs	Port specific COM-related APIs to OpenVMS.
Advanced Server for OpenVMS	Enhance Advanced Sserver to support NTLM security, OpenVMS Registry, OpenVMS events.
COM for OpenVMS	Implement COM on OpenVMS.

Table 3 catalogues only the major infrastructure projects—I've left many small projects and subprojects off the list. Of the major projects I've listed, most are invisible to OpenVMS customers; I've listed them to give you an idea of size and scope of the work necessary to bring COM to OpenVMS.

The vision of COM for OpenVMS was to allow OpenVMS to function as a full member of a Windows NT domain. This meant support for distributed objects and applications (through COM), common accounts (through single sign-on) and distributed file and print services (through Advanced Server for OpenVMS).

Aside from being a long and ambitious project, the infrastructure work was a rat's nest of interdependencies. For example, engineering had to have the RPC layer in place before they could start the SSPI work; single sign-on had dependencies on Advanced Server; engineering needed the COM APIs before they could implement COM; and engineering had to test and qualify each layer separately, then retest and requalify as they integrated each layer.

Throughout the project, the OpenVMS infrastructure engineers worked closely with their counterparts in Tru64 UNIX. This included discussing architectural and design approaches, sharing code, and participating in joint demonstrations. For example, the demo code on the CD-ROM included with this book was a joint effort among Microsoft, OpenVMS, and Tru64 UNIX.

Implementation challenges

When COM for OpenVMS shipped its Version 1.0 release in January 1999, it represented over four years of continuous development work by a collection of coordinated engineering teams that numbered about 65 people at the project's high-water mark. (In the middle of all this ongoing work, Compaq acquired Digital Equipment Corporation in June 1998.)

The following sections describe a few of the technical hurdles that OpenVMS engineering needed to address before bringing COM on OpenVMS to market.

Creating a Windows NT environment on OpenVMS

The more Windows NT environment we provided on OpenVMS, the less Windows code we needed to port. As a result, OpenVMS engineering implemented as much of the underlying Windows NT environment as needed on OpenVMS to support native Windows code.

Windows is optimized for a 32-bit Intel platform. To bring key pieces of Windows technology to OpenVMS, the code had to be ported and reimplemented for a 64-bit Alpha platform.

Win32 APIs and COM APIs

On Windows systems, Win32 APIs provide a set of public interfaces to Windows operating systems functions—for example, processes, threads, synchronization, registry, events, Windows NT services, performance monitor, and so on. COM APIs provide a set of public interfaces to the DCOM functions—for example, threads APIs, activation APIs, security interface APIs, and so on.

On OpenVMS, the Win32 APIs are provided by Bristol Technology. Bristol's implementation of the Win32 APIs on OpenVMS was in place before the COM on OpenVMS work started; OpenVMS engineering chose to buy, rather than build, the Win32 environment. OpenVMS instead decided to spend its engineering effort porting the COM APIs to OpenVMS.

MIDL compiler

On Windows systems, Microsoft RPC consists of run-time libraries, an interface definition language (IDL), and an IDL compiler. The Microsoft IDL (MIDL) compiler allows you to create language-dependent files that can be added into or read by the COM client or server.

The OpenVMS engineers ported the Microsoft Interface Definition Language compiler to OpenVMS. This MIDL compiler on OpenVMS is identical to the Microsoft MIDL compiler.

The rpcss process

On Windows NT the rpcss process does several things: builds endpoint maps of server locations, builds and maintains the ROT (running object table) for all COM components, starts servers as needed, and communicates with rpcss processes on other systems to locate and start COM components.

The OpenVMS engineers ported the Microsoft rpcss process to OpenVMS, where it appears as DCOM\$RPCSS. OpenVMS uses the DCE endpoint mapper to maintain the location of the DCOM\$RPCSS process and the RPC applications.

Registry

The Windows NT Registry is a single, systemwide hierarchical database of configuration information about hardware and software (both the operating system and applications). The Windows NT Registry replaced Windows 3.x .ini files, providing a single place for storing application and configuration information.

To allow OpenVMS and Windows NT to interoperate, Compaq created a registry on OpenVMS. Like the Windows NT Registry, the OpenVMS Registry is made up of two components: the OpenVMS Registry database and the OpenVMS Registry server. The OpenVMS Registry database is a systemwide or clusterwide hierarchical database of configuration information. This information is stored in a database structure of keys and associated values. The OpenVMS Registry server controls all OpenVMS Registry operations, such as creating and backing up the OpenVMS Registry database, and creating, displaying, modifying, or deleting keys and values.

The OpenVMS Registry includes interfaces (COM APIs and system services) that allow applications to control the OpenVMS Registry server and to read and write to the OpenVMS Registry database. The OpenVMS Registry also includes server management utilities that allow system managers to display and update OpenVMS Registry information from the OpenVMS DCL command line.

The OpenVMS Registry is compatible with the Windows NT Registry. Windows NT client applications such as RegEdit 3.2 can connect to and edit the OpenVMS Registry.

The OpenVMS Registry stores keys and values for both COM for OpenVMS and Advanced Server. Both COM for OpenVMS and Advanced Server include commands to initialize the OpenVMS Registry. Because the OpenVMS Registry serves both COM for

OpenVMS and the Advanced Server, you must be careful when you reinitialize the OpenVMS Registry—you can affect running components.

Reading and writing to the OpenVMS Registry

You can read and write to the OpenVMS Registry in the following ways:

- Using COM for OpenVMS, through the COM APIs available on OpenVMS. This allows application programmers to enter, modify, and delete OpenVMS Registry keys and values.
- Through the \$REGISTRY and \$REGISTRYW system services and the OpenVMS Registry server management utility commands. This allows application programmers to enter, modify, and delete OpenVMS Registry keys and values.
- From Windows NT, through the Windows NT Registry APIs, or using RegEdit 3.2 (the Windows NT Registry Editor). This allows Windows NT users to view and edit OpenVMS Registry keys and values.

OpenVMS Registry server management

The REG\$CP server management utility allows you to display and update OpenVMS Registry information from the OpenVMS DCL prompt. The utility also allows you to back up and restore the entire OpenVMS Registry database to or from a file, as long as you have the required system privileges.

Events

On a Windows NT system, an *event* is any significant occurrence in the system or an application—for example, a service starting or stopping, a user logging on or off, or accessing resources. When the system encounters an event, the Event Log service writes the event (or audit entry) in the form of a record that contains date and time, source, category, event number, user, and computer information to a system, security, or application log, creating an audit trail. On Windows NT systems, you display these logs and their recorded events using the Event Viewer.

COM for OpenVMS supports both Windows NT logging and Advanced Server for OpenVMS logging of COM for OpenVMS events. You can log COM for OpenVMS events (such as the starting of a COM server on OpenVMS), and review them from a Windows NT system or an OpenVMS system.

The system logs OpenVMS events to a Windows NT event log, to the Advanced Server for OpenVMS event log, and to a log file on the OpenVMS system. You can view OpenVMS Events using any of the following:

- Windows NT event viewer

- Advanced Server for OpenVMS event viewer
- OpenVMS event log file

By default, the system logs DCOM events generated by COM for OpenVMS. In addition to recording COM for OpenVMS events, the system can also log COM application events for COM applications that you create. The COM for OpenVMS kit includes sample code that shows you how to generate an application event using Win32 APIs.

Threading model

On Windows, COM supports two threading models: STA (single-threaded apartment) model and MTA (multithreaded apartment) model. The STA model allows Windows to run legacy applications or new applications that are not thread-safe.

COM for OpenVMS supports only the multithreaded apartment (MTA, also known as free threads) model for application servers. The multithreaded apartment model allows a component to have more than one thread. OpenVMS implemented only the MTA model because server applications need to be thread-safe for efficiency.

Windows NT has its own threading APIs. OpenVMS uses the POSIX threads APIs.

Security

On Windows NT systems, authorization and authentication between COM objects is handled using the NTLM (NT LanManager)

The OpenVMS engineers enhanced OpenVMS security to allow OpenVMS to understand the NTLM security model. For a detailed discussion of OpenVMS security, see Chapter 6.

Graphical vs character-cell interface

Windows provides a graphical interface to almost all applications, tools, and utilities. When the OpenVMS engineers ported the COM and registry tools and utilities (such as dcomcnfg and regsvr32) to OpenVMS, the engineers created character cell or command line interfaces to these tools and utilities. For more information on these utilities, see Chapter 7.

Service Control Manager (SCM)

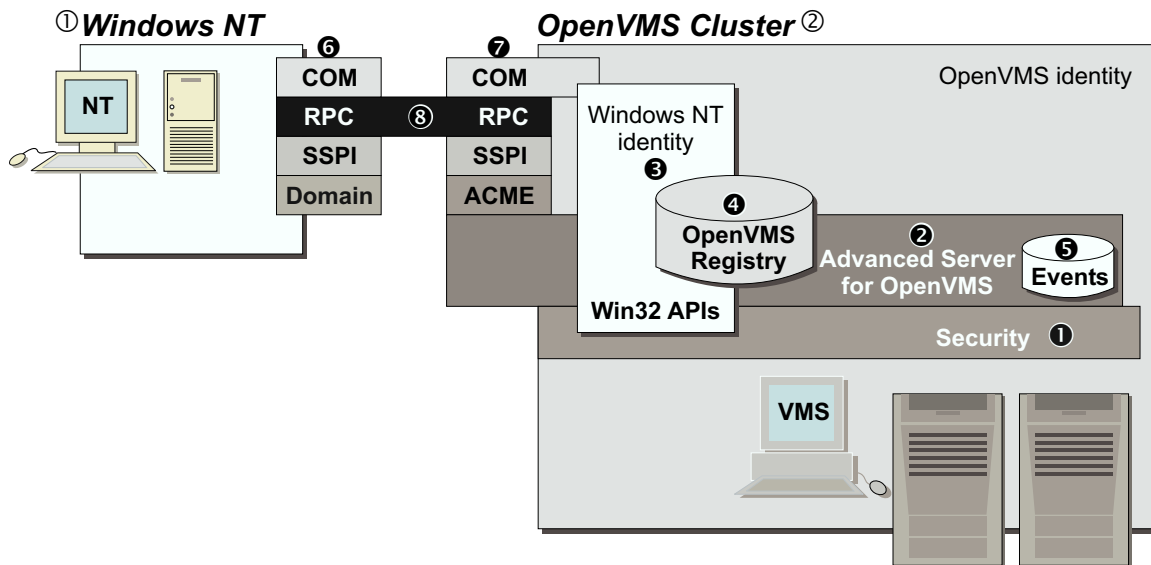
On Windows NT systems, the Service Control Manager (SCM) is an RPC server that unifies and provides secure management of Windows NT services. The SCM allows you to stop, start, pause, and resume all services running on the Windows NT system. Windows NT starts the Service Control Manager service automatically at startup.

The OpenVMS engineers chose not to implement a Windows NT-like SCM on OpenVMS. On an OpenVMS system, applications that depend on server services (such as stop, start, pause, and resume) rely on the OpenVMS features that provide similar functionality. For example, you use the OpenVMS site-specific startup and shutdown command procedures to implement automatic starting of services at system startup and automatic shutdown of services at system shutdown. That is, OpenVMS doesn't provide service APIs such as RegisterServiceCtrlHandler, ChangeServiceConfig, and so on.

OpenVMS infrastructure architecture

As I described earlier in this chapter, to support COM for OpenVMS, Compaq ported Windows NT infrastructure to OpenVMS, including a registry, events logger, NTLM (NT LanManager) security, and Win32 APIs. COM for OpenVMS is layered on The Open Group's Distributed Computing Environment (DCE) RPC. COM for OpenVMS supports communication among objects on different computers on a local area network (LAN), a wide area network (WAN), or the Internet. Figure 20 shows a schematic view of the OpenVMS infrastructure. The numbers in the figure correspond to numbered descriptions that follow.

Figure 20. OpenVMS infrastructure that supports COM for OpenVMS



① **Windows NT system**

The smaller box represents the Windows NT system.

② **OpenVMS Cluster**

The large box represents the OpenVMS system. Within and around this box you can see several other boxes labeled with numbers. The following list describes those numbered items:

❶ **OpenVMS security**

This is the standard OpenVMS security (login, authentication, ACLs, and so on) available with all OpenVMS systems.

❷ **Advanced Server for OpenVMS**

The Advanced Server for OpenVMS provides authentication of Windows NT users to OpenVMS and provides a connection to the OpenVMS Registry and events viewer for Windows NT users.

❸ **Windows NT identity/Win32 APIs**

The OpenVMS Security, MSV1_0 ACME agent, Advanced Server for OpenVMS, OpenVMS Registry, event logger, and Win32 APIs (COM APIs) all contribute to the creation of a Windows NT identity within the OpenVMS system.

❹ **OpenVMS Registry**

The OpenVMS Registry, like the registry on Windows NT systems, allows you to store system, software, and hardware configuration information on OpenVMS. COM for OpenVMS uses the OpenVMS Registry to store information about COM applications.

❺ **Event logger**

Like the event logger on Windows NT systems, the event logger on OpenVMS records informational, warning, and error messages about COM events.

❻ **Windows NT COM stack**

On the Windows NT system, COM requests and responses pass through the COM, RPC, SSPI (security), and Domain layers.

❼ **OpenVMS COM stack**

The OpenVMS system mirrors the Windows NT COM stack, with some additions. On the OpenVMS system, COM requests and responses pass through

the COM, RPC, SSPI (security) , MSV1_0 ACME agent, and Advanced Server for OpenVMS layers. The MSV1_0 ACME agent is an extension to the Authentication and Credential and Management (ACM) authority.

⑧ Connection through RPC layer

The COM connection between the Windows NT system and OpenVMS is always through the RPC layer.

COM for OpenVMS releases

Compaq released COM for OpenVMS in two phases.

- Phase 1 implemented **unauthenticated COM** for OpenVMS.

Compaq delivered this phase with COM Version 1.0 for OpenVMS in OpenVMS Version 7.2 (January 1999). In COM Version 1.0 for OpenVMS, a COM process executes with an OpenVMS security identity only; OpenVMS does not authenticate COM requests from Windows NT clients or process any Windows NT credentials.

- Phase 2 implemented **authenticated COM** for OpenVMS.

Compaq delivered this phase with COM Version 1.1 for OpenVMS in OpenVMS Version 7.2-1 (July 1999). In COM Version 1.1 for OpenVMS, a COM server process executes in the security context of the requesting Windows NT client. The COM for OpenVMS server process includes Windows NT credentials that OpenVMS can use for OpenVMS Registry access and outbound COM requests. COM Version 1.1 for OpenVMS is the full implementation of NTLM security for COM for OpenVMS.

Table 4 summarizes the security implementation differences between COM Version 1.0 for OpenVMS and COM Version 1.1 for OpenVMS.

Table 4. Summary of differences

Area	COM Version 1.0 for OpenVMS	COM Version 1.1 for OpenVMS
Client requests	Authenticated on Windows NT; not authenticated on requests to OpenVMS.	Authenticated on Windows NT and OpenVMS.

continued

Table 4 (con't). Summary of differences

Area	COM Version 1.0 for OpenVMS	COM Version 1.1 for OpenVMS
Security	Servers can run with the client's identity on Windows NT and run with a pre-specified OpenVMS identity on OpenVMS.	Servers can run with the client's identity on Windows NT and on OpenVMS.
	Per-method security is allowed on Windows NT but only process-wide security is allowed on OpenVMS.	Per-method security is allowed on Windows NT and on OpenVMS.
Outbound COM requests	Authenticated on Windows NT only.	Authenticated on Windows NT and OpenVMS.
Registry access	<i>On Windows NT:</i> controlled by NT credentials. <i>On OpenVMS:</i> relies on OpenVMS security controls such as privileges or rights identifiers.	<i>On Windows NT:</i> controlled by NT credentials. <i>On OpenVMS:</i> controlled either by Windows NT credentials or by OpenVMS security controls.
Event logging	Windows NT only.	Windows NT and OpenVMS.

Delivery of COM for OpenVMS

COM for OpenVMS is delivered as follows:

- Run-time environment

COM for OpenVMS provides a free run-time environment on OpenVMS Alpha for the deployment of COM for OpenVMS client and server applications. This COM for OpenVMS run-time environment is licensed with and included as part of OpenVMS Version 7.2-1.

- Developer's kit

For developers, the COM for OpenVMS developer's kit provides a Microsoft Interface Definition Language (MIDL) compiler and C-style header files for application development. This COM for OpenVMS developer's kit ships with OpenVMS Version 7.2-1 and has a separately orderable license.

Chapter 6. COM for OpenVMS security

How do COM clients from Windows systems interact securely with COM servers on OpenVMS?

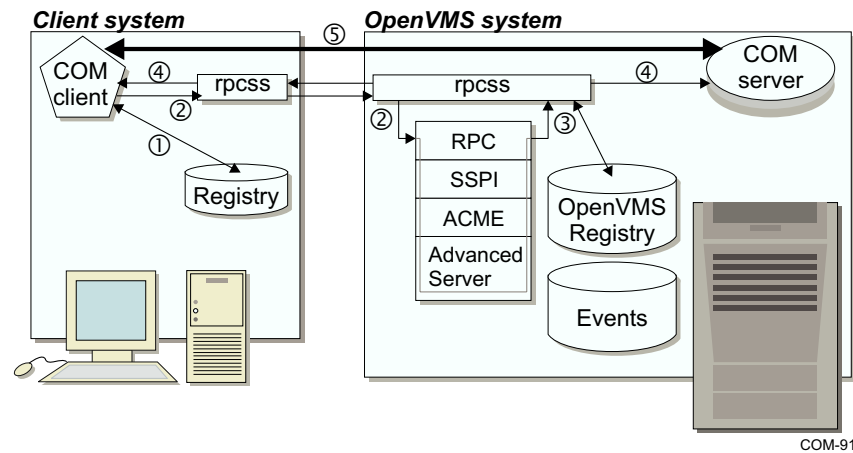
OpenVMS is famous for its security, from controlling break-ins to using ACLs (access control lists) to limit access to directories, images, and files. Windows NT has a similar, but different, security scheme. How do the two systems work together to ensure that the correct Windows user is starting the correct COM server and is allowed to view the resulting data?

As I described in Chapter 5, COM for OpenVMS sits at the top of the OpenVMS infrastructure. That means that COM for OpenVMS is integrated with existing OpenVMS security, Advanced Server security, the OpenVMS Registry, the ACMEs (Authentication and Credential Management Extensions) and the RPC (remote procedure call) process. The following sections describe how all these pieces work together to create a secure environment for running COM objects on OpenVMS.

Starting a COM server securely on OpenVMS

This section explains step-by-step how a COM client application from a Windows system is authenticated, authorized, and starts an COM server application on OpenVMS. The numbers in Figure 21 correspond to the steps in the description.

Figure 21. How a COM client starts a COM server on OpenVMS



On a Windows system, a user starts a COM-enabled application. This application needs to get data from another COM application on OpenVMS. The following events then take place:

1. The COM client application checks in the local registry to see on what system the other COM application (the COM server) resides. The registry shows that the COM server is on an OpenVMS system.
2. The COM client makes a request through the `rpcss` process from the client system to the `rpcss` process on the OpenVMS system. On the OpenVMS system, `rpcss` authenticates the COM client and assumes the identity of (impersonates) the client system user. To do this, the request passes through the RPC (remote procedure call) layer and the SSPI (security support provider interface) layer to the ACMEs (Authentication and Credential Management Extensions). The ACME agents—which allow OpenVMS to authenticate a user and acquire the user’s credentials for non-OpenVMS security environments—process the request as follows:

- NT ACME agent

Using the COM client’s username and other authentication information, the NT ACME authenticates the client through the Advanced Server’s SAM database. This step provides the `rpcss` process with the NT credentials it needs to impersonate the COM client.

- VMS ACME agent

Using the OpenVMS username mapping for the COM client (which it gets from Advanced Server through the NT ACME), the VMS ACME provides the `rpcss` process with the VMS credentials associated with the COM client.

3. The now *authenticated* request passes back to the `rpss` process, which is now impersonating the COM client. The `rpss` process checks the OpenVMS Registry for the following:
 - Is the server application registered?
 - What is the server application's OpenVMS image name?
 - What are the security settings for this server application? (Is the client *authorized* to start this server?)
 - What account (identity) should run this server application? That is, has the server been specified to run as the launching user or to run as a specified user?
4. When the `rpss` process confirms that the client process is both authenticated and authorized, the `rpss` process does the following:
 - Starts the COM server on OpenVMS. The `rpss` process uses the NT and VMS credentials it acquired in Step 2 to start the COM server identified in Step 3.
 - Notifies the COM client that the server is now available.
5. The COM client application and the COM server application now communicate directly over RPC.

Authentication

The time-honored movie cliché “Halt! Who goes there?” is an *authentication* request: you're being challenged to tell someone who you are.

In the everyday world, you authenticate yourself by presenting identification—a business card, a driver's license, picture ID, a credit card, and so on. In cases where someone else requires a more secure form of identification, you might be asked to provide a piece of information only you should know—your employee ID number, social security number, mother's maiden name, or some agreed-on password.

On a computer system, authentication is the act of verifying a user's identity before permitting entrance or access to the system. Typically you respond to an authentication request by providing a user name that identifies you to the system, followed by some secret password that only you should know. Your identity can be in the form of a username and password, or a secure card, or a biometric identity such as a thumbprint or retina scan.

Authorization

“Who said you could do that?” is an *authorization* request: you're being asked to show that you have permission from some authority to do something or go somewhere.

In the everyday world, you prove that you are authorized to do something by presenting some form of credentials—an auto registration, a letter of introduction, a certificate or diploma, a legal document (like a power of attorney), and so on. All of these documents list or describe your authority to be somewhere or perform some specific activity.

On a computer system, after a system authenticates you, it binds your credentials—your authorization information—to your user process. The system uses these authorization credentials to determine whether to grant or deny access to system resources.

Impersonation

In the everyday world, you probably don't impersonate other people very often (I'm talking about impersonation in the legal definition of the word, not those bad impressions of actors or politicians you do at the water cooler)—in fact, there are laws against impersonation. However, there are cases in which you grant others the authority to act as agents in your name—for example, when you contract with someone to buy real estate for you at a distant location, or have someone rent your house when you are an absentee landlord.

When you request that a computer system start a new task or process for you, the new process may *impersonate* you—that is, the system starts the process with your identity and credentials. In this way, if the process needs other system resources, it can show your credentials to the system; the system can then grant or deny the process (that is impersonating you) access to the additional resources as if you had requested the resources directly. In this way, the system treats you and your impersonated process (like a COM application) as the same thing.

Authentication, impersonation, and authorization on OpenVMS

OpenVMS provides both native (SYSUAF-based) and Windows NT-compatible authentication and authorization capabilities as follows:

- OpenVMS authentication and authorization (native)

The system authenticates a user based on password information stored in the SYSUAF.DAT file. The authorization information consists of UIC, privileges, and rights identifiers.

- Windows NT authentication

The system authenticates a user based on password information stored in a SAM database managed by the domain controllers. This authorization information consists of primary SID, group SIDs, session key, and privileges obtained from the user's account information in the SAM database.

After OpenVMS successfully authenticates a user (either using OpenVMS or Windows NT), OpenVMS attaches the user's OpenVMS (native) credentials to the process using a structure called a *persona*. If the system authenticated the user through Windows NT, OpenVMS also attaches the user's Windows NT credentials to the process (as an extension to the persona).

When a Windows COM client requests that OpenVMS start a COM server, the request is passed to OpenVMS through *rpcss*. At this point, *rpcss* impersonates the Windows NT user from whose account the request originates—that is, the user request logs in to the OpenVMS system through the Advanced Server. OpenVMS grants the user request the same Windows NT credentials and OpenVMS credentials as if the user logged in directly.

With the user's credentials, the request can now impersonate the client.

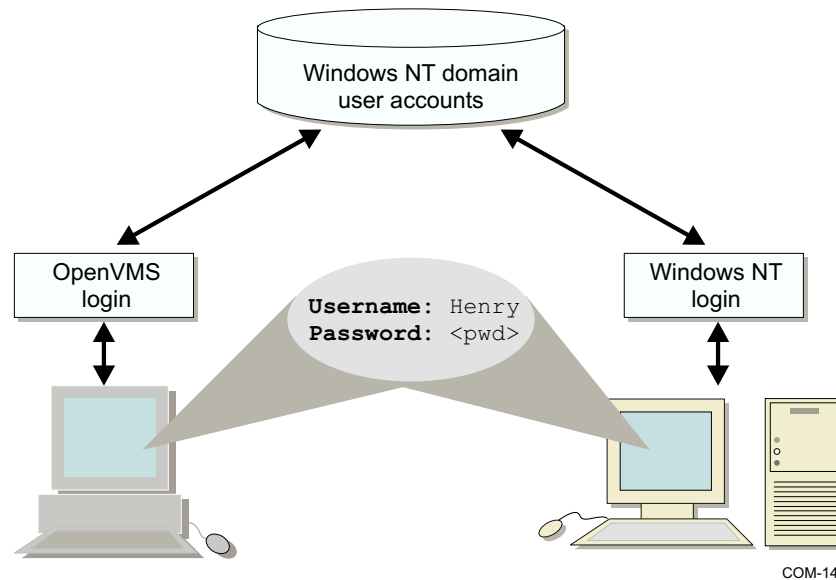
Authentication, impersonation, and authorization in action: an example

The ACME services allow Windows NT users (or Windows NT applications) to log on to OpenVMS using a Windows NT password (or RPC/SSPI-based NTLM challenge/response). Both Windows NT and OpenVMS use the Windows NT database to authenticate the user.

For example, a COM client application on Windows NT wants to access a COM server on OpenVMS. The client application has to be able to connect to the OpenVMS system. But the only login information the COM client application has is the Windows client system's user name (Henry) and password ("Jake sent me"). When the COM client application comes to the OpenVMS login door and knocks, OpenVMS slides the peephole open and asks, "Who are you and what's the password?"

The COM client application presents its Windows user name and password to OpenVMS: "I'm Henry. Jake sent me." OpenVMS checks the Windows NT domain user accounts for the client application's account (Henry) and confirms that the client application's password really is "Jake sent me." OpenVMS unbolts the door and lets the client application in. Figure 22 shows how this authentication works.

Figure 22. Authentication: the shared user account database

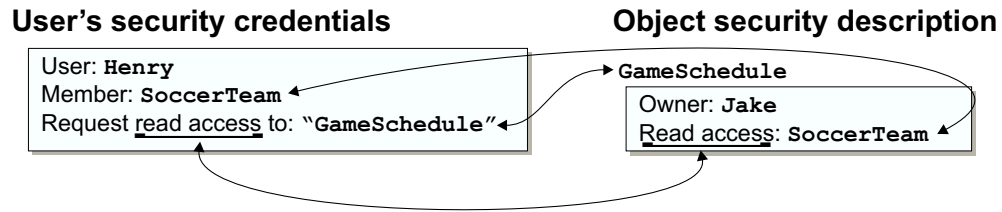


At this point, the OpenVMS process is impersonating the COM client using a persona extension. The persona extension services allow OpenVMS to store a user's Windows NT credentials on OpenVMS. The system uses these additional credentials to grant or deny access to Windows NT services, such as COM objects, and to impersonate a COM client (or Windows NT user).

For example, the COM client passed the first OpenVMS checkpoint by telling OpenVMS the COM client's Windows name (Henry) and password ("Jake sent me"). But if the client wants to access a COM server on OpenVMS, the client application needs more than just a matching password—the client application needs to present its Windows NT credentials to OpenVMS to prove that the client really is who he says he is.

OpenVMS keeps a record of both the client's Windows persona and the client's OpenVMS persona. So when OpenVMS authenticated the client application, OpenVMS provided the client with a set of OpenVMS credentials and with a set of Windows NT credentials. In this example, the client application (impersonating Henry) wants to see the COM server (GameSchedule) that his friend, Jake, owns. Jake's COM server knows the soccer game schedule. The client's credentials show that Henry is a member of SoccerTeam, so the client (impersonating Henry) will be allowed to read the game schedule. Figure 23 shows an example of the client's persona extension.

Figure 23. Persona extensions: Windows NT credentials on an OpenVMS system



COM-15

Authentication and Credential Management (ACM) authority details

The Authentication and Credential Management authority authenticates users and determines the user security profile for OpenVMS and Windows NT. The ACME_SERVER process provides these ACM services. The ACME_SERVER process uses plug-in modules called ACME agents. ACME agents perform the actual work of responding to authentication requests, query requests, and event requests.

The OpenVMS ACME agent (VMS\$VMS_ACMESHR.EXE) provides OpenVMS native services. The MSV1_0 ACME agent (PWRK\$MSV1_0_ACMESHR.EXE, an Advanced Server for OpenVMS product component) provides Windows NT connectivity services.

The MSV1_0 ACME agent forwards Windows NT connectivity service requests from NTA\$LOGON and SSPI/NTLM to an Advanced Server for OpenVMS process (PWRK\$LMSSRV.EXE) running on one or more systems in the cluster.

The PWRK\$ACME_SERVER logical name contains a comma-delimited list of cluster node names to which the MSV1_0 ACME can forward requests. Running the Advanced Server for OpenVMS process on more than one cluster node and including the node names in the PWRK\$ACME_SERVER logical name allows the MSV1_0 ACME agent to fail over a request automatically if a connection is interrupted.

The ACME_SERVER process must be present on any system running RPC or COM for OpenVMS. However, the Advanced Server for OpenVMS process needs to be present on only one node in the cluster.

Rules for Windows NT authentication on OpenVMS

Because the ACME_SERVER returns a complete OpenVMS persona with the requested attached Windows NT persona extension, the VMS ACME agent enforces the following rules:

- Every Windows NT user must be mapped to a local OpenVMS user name. The MSV1_0 ACME provides this mapping through the Advanced Server for OpenVMS HOSTMAP database.
- The mapped OpenVMS user name must be a valid (and not disabled) account in the SYSUAF.DAT. The account's access restrictions must allow access during the specified days and times. COM for OpenVMS and RPC typically require NETWORK access during authentication.
- The mapped OpenVMS user name must be an account with the EXTAUTH flag set. EXTAUTH allows the system manager fine control over which OpenVMS accounts can be used for mapping. You can use the IGNORE_EXTAUTH bit (bit number 11 [decimal]) in the SECURITY_POLICY system parameter to override this per-account feature. If you set the IGNORE_EXTAUTH bit to 1, OpenVMS allows you to map to any account, regardless of the account's EXTAUTH setting.

Using Windows NT and OpenVMS security together

The following example shows how you can use Windows NT and OpenVMS together to create a secure, restricted-use COM application.

The N&L Company uses a two-tier computer architecture to run its import/export business: Windows on the desktop and OpenVMS systems in the datacenter. One of its key applications is the payroll/personnel system. N&L's IS department created this application, which was written as COM components.

Eileen, the N&L Company's CFO, needs to limit access to its payroll system. Eileen identified only two people in the payroll group—Marie and Dennis—who can have access to the payroll system. To further restrict access, the Eileen wants Marie and Dennis to have access to the payroll system only between the hours of 10:00am and 11:30am on Fridays. Also, because the payroll database is shared with the personnel organization, Theresa, the N&L Company's HR manager, wants to be sure that Marie and Dennis have access to only employee payroll records and not to the full employee database.

To implement this set of requirements, Alice, the N&L Company's IS manager, does the following:

1. On the Windows NT system, Alice creates an account called PAYROLL and provides a password. This is the account through which Marie and Dennis will access the payroll application on OpenVMS.
2. On the OpenVMS system, Alice creates an account called PAYROLL and provides the same password as on the Windows NT system. This is the account in which the payroll application COM component will run.

3. Within Advanced Server for OpenVMS, Alice maps the Windows NT PAYROLL account to the corresponding OpenVMS PAYROLL account. This procedure ties the Windows NT and OpenVMS accounts together for authentication and authorization (as described earlier in this chapter).
4. On the OpenVMS system, Alice now sets SYSUAF restrictions on the OpenVMS PAYROLL account. These restrictions limit the login hours to between 10:00am and 11:30am on Fridays.
5. Alice now runs the DCOM\$SETUP command file, and from the Application Identity submenu, specifies that the OpenVMS account should use the security context of the Launching User.
6. On the OpenVMS system, Alice now sets the UIC information for the database—that is she sets OpenVMS ACL (access control lists) for the PAYROLL user so that the PAYROLL user can read and modify only payroll-specific database fields—effectively locking the PAYROLL account users out of the HR areas of the database.

Alice has now configured the systems so that Marie and Dennis can run a COM client on a Windows NT system and access the payroll COM server only on Fridays between 10:00am and 11:30am. If Marie or Dennis tries to start the payroll process at any other time, the OpenVMS system rejects the Windows NT attempt to connect to OpenVMS. In addition, the OpenVMS PAYROLL account ACLs restrict Marie and Dennis from viewing or changing any data in the database not related to the payroll account.

Chapter 7. Getting COM for OpenVMS installed and running

Installing COM for OpenVMS

The COM for OpenVMS installation kit contains a single POLYCENTER Software Installation file. You must install the COM for OpenVMS files on an OpenVMS Alpha Version 7.2-1 (or higher) system. Before you install the kit, you need the following:

- For OpenVMS systems
 - ♦ OpenVMS Version 7.2-1 or higher
 - ♦ Compaq C Version 5.6 or higher and Compaq C++ Version 5.6 or higher (for COM for OpenVMS application development)
 - ♦ Compaq TCP/IP Services for OpenVMS Version 5.0 or equivalent
 - ♦ Compaq Advanced Server for OpenVMS Version 7.2A or higher
- For Windows® NT™ systems
 - ♦ Windows NT 4.0 with Service Pack 3 or higher installed
 - ♦ Microsoft® Visual C++ (for Windows NT client development and information about MIDL compiler). See the Microsoft website for compiler version requirements.
 - ♦ TCP/IP enabled (needed for OpenVMS connectivity)

Configuring and running COM for OpenVMS

COM for OpenVMS includes several utilities that help you configure and run COM for OpenVMS. These character cell interface tools replace, and, in some cases, extend similar Windows GUI tools. The utilities are as follows:

- DCOM\$SETUP, which helps a system manager configure the COM for OpenVMS system environment.
- DCOM\$CNFG, which helps an application developer configure and examine COM applications.
- DCOM\$REGSVR32, which allows an application developer to register and unregister in-process server applications.

The following sections describe these utilities in more detail.

DCOM\$SETUP

DCOM\$SETUP is a collection of tools to help a system manager configure the COM for OpenVMS system environment. Use DCOM\$SETUP to populate the OpenVMS Registry database with COM for OpenVMS keys and values. Figure 24 shows the DCOM\$SETUP menu.

Figure 24. DCOM\$SETUP OpenVMS COM Tools Menu

```

-----
                        OpenVMS COM Tools

1) DCOMCNFG, COM Configuration Properties
2) GUIDGEN, Globally Unique Identifier Generator
3) Populate the Registry database for COM
4) Start the COM server
5) Stop the COM server
6) Register a COM application
7) Create the DCOM$GUEST account and directory
8) Configure the DCOM$RPCSS accounts
H) Help
E) Exit
Please enter your choice:
-----

```

The options are as follows:

- 1) DCOMCNFG, COM Configuration Properties
Use to query information and manipulate properties of COM for OpenVMS applications.
- 2) GUIDGEN, Globally Unique Identifier Generator
Generate CLSIDs (class IDs) (or GUIDs [globally unique identifiers]) in various formats (for example, the OpenVMS Registry or Windows NT Registry format).

The CLSID tags each application with a unique identifier. This version of DCOM\$SETUP generates GUIDs in OpenVMS Registry and Windows NT Registry formats only.

3) Populate the Registry database for COM

Set up the OpenVMS Registry database. COM for OpenVMS requires that specific keys and values be added to the OpenVMS Registry database. You must have both write access to the OpenVMS Registry and Windows NT Administrator privileges.

4) Start the COM server

Start the COM for OpenVMS server control server process (DCOM\$RPCSS). DCOM\$SETUP calls the SYS\$STARTUP:DCOM\$STARTUP procedure to start the server.

5) Stop the COM server

Shut down the COM for OpenVMS server control server process (DCOM\$RPCSS). DCOM\$SETUP calls the SYS\$STARTUP:DCOM\$SHUTDOWN procedure to stop the server.

6) Register a COM application

Register a COM for OpenVMS server application. You can register the following types of servers:

- In-process server

When you register an in-process server, the system prompts you for the server's location.

- Local server or out-of-process server

When you register a local server or out-of-process server, the system prompts you for the following information:

- ♦ Full path information (location of the server)

This is a required value. Use the following syntax:

device:[directory]file-name.ext

- ♦ Application title

This is an optional value. If you do not supply a title, the system uses a default title.

- ◆ CLSID (GUID)

This is a required value. If the server does not have a CLSID, the system generates one automatically.

After you complete the registration process, the system generates the following files:

A Windows NT Registry file (*server-name*.REG_NT) that you can use to register the application on a Windows NT system.

An OpenVMS command procedure (*server-name*.REG_VMS) that you can use to register the server on an OpenVMS system.

When you use these files on other systems, you must modify the path statement to point to the server's current location.

7) Create the DCOM\$GUEST account and directory

You must create the DCOM\$GUEST account before you can use COM for OpenVMS without NTLM authentication.

8) Configure the DCOM\$RPCSS accounts

Configure and create the DCOM\$RPCSS Advanced Server for OpenVMS user and SYSUAF accounts. The COM for OpenVMS server control server process (DCOM\$RPCSS) requires these accounts for authentication.

H) Help

Display help about each menu option.

E) Exit

Exit the menu.

DCOM\$CNFG

DCOM\$CNFG is a utility to help COM developers configure and manage COM for OpenVMS applications on OpenVMS. Use the DCOM\$CNFG utility to query information and manipulate properties of COM for OpenVMS applications.

To use the DCOM\$CNFG utility, choose option 1 from the DCOM\$SETUP menu. Figure 25 shows the DCOM\$CNFG menu.

Figure 25. DCOM\$CNFG Main Menu

```

-----
                        DCOM$CNFG Main
1 - Applications List
2 - System-wide Default Properties
3 - System-wide Default Security
(E to Exit)
(H for Help)
Enter <CTRL-Z> or 'E' to return to the previous menu at any time
Please enter your choice:
-----

```

The options are as follows:

- 1 - Applications List
 - Lists all applications registered on this machine.
- 2 - System-wide Default Properties
 - Allows you to set systemwide machine properties.
- 3 - System-wide Default Security
 - Allows you to set systemwide security parameters.

DCOM\$REGSVR32

All COM components (implemented as either an out-of-process server or as an in-process server) must be registered in the OpenVMS Registry before you can use them.

Out-of-process servers, which are implemented as executable programs (.EXE files), usually contain code to register and unregister the components contained within them. The advantage an out-of-process server has over an in-process server is that you can run the executable and automatically create the necessary registry keys.

In-process servers, which are usually implemented as dynamic link libraries (.DLL files) on Windows NT or as shareable images on OpenVMS, also contain code to register and to unregister the components within them automatically. However, these in-process servers cannot be run the same way as an executable image because they do not contain a main entry point. As a result, you must manually register the components contained within a .DLL, or create a command procedure to perform the registration.

Microsoft provides the REGSVR32 utility that you can use to register the components contained within a DLL. REGSVR32 takes as a command line argument the following:

- DLL name
- Switches to register or unregister the components

When registering a DLL's components, REGSVR32 searches the specified DLL for the `DllRegisterServer` symbol and, if found, calls it. When unregistering a DLL, REGSVR32 calls `DllUnregisterServer`. This means that all in-process components that you want to register automatically must include these two entry points in their export files.

To facilitate the registration of components contained within shareable images on OpenVMS systems, Compaq created the `DCOM$REGSVR32` utility. The `DCOM$REGSVR32` utility does the same things that the Microsoft REGSVR32 utility does. Any shareable images that contain components to be registered must also include the `DllRegisterServer` and `DllUnregisterServer` universal symbols in their symbol vectors. Both the `DCOM$REGSVR32` and the REGSVR32 utilities use the same command line syntax.

During the COM for OpenVMS installation, the system places the `DCOM$REGSVR32.EXE` file in the `SYS$SYSTEM` directory.

Before you use the `DCOM$REGSVR32` utility, you must define a symbol that allows the utility to accept foreign command lines. For example:

```
$ regsvr32 ::= $DCOM$REGSVR32
```

Alternatively, you can activate the `DCOM$REGSVR32` utility as follows:

```
$ MCR DCOM$REGSVR32
```

You can use either method to activate the utility, and register or unregister components contained in shareable images.

To display help for `DCOM$REGSVR32`, enter the following:

```
$ regsvr32 -?
```

Developing a COM application for OpenVMS

The following sections describe the basic steps for developing a COM application on OpenVMS.

Step 1: Generate unique identifiers

Use the `DCOM$GUIDGEN` utility to generate 16-byte globally unique identifiers (GUIDs). A GUID uniquely identifies each interface; you must include these GUIDs in the IDL file used in Step 2. COM uses these GUIDs to identify COM servers; COM ultimately writes

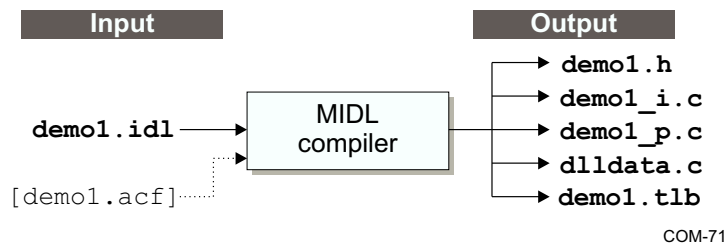
these GUIDs as keys in the OpenVMS Registry. For more information about using this utility see the *OpenVMS Connectivity Developer Guide*.

Step 2: Build the app using the MIDL compiler

The Interface Definition File (IDL) contains code to describe the interfaces, methods, parameters, and type libraries. This code is processed by the MIDL compiler to generate header files, proxy and stub files, and the type library (.TLB) file.

Use the MIDL compiler to generate the header file for the functions, the IID file, the proxy file, the DLLDATA files, and the type library file. When processing the IDL file, the MIDL compiler also checks for an optional ACF (attribute configuration file). You can use the ACF to specify additional attributes for the application interface. Figure 26 shows the MIDL compiler input and output files.

Figure 26. MIDL compiler input and output



The output from the MIDL compiler is as follows:

- Header file (demo1.h): defines the data structure functions.
- IID file (demo1_i.c): contains GUIDs (CLSIDs) for interfaces.
- Proxy (demo1_p.c): handles marshalling and unmarshalling.
- DLLDATA file (dlldata.c): used to make DLL or shareable image library.
- Type library file (demo1.tlb): compiled version of IDL file that describes the interfaces in a format usable by OLE automation.

For more information about running the MIDL compiler see the *OpenVMS Connectivity Developer Guide*.

Step 3: Compile the COM app

Use the Compaq C compiler to compile the proxies and stubs generated by the MIDL compiler. Use the Compaq C++ compiler to compile the files containing the client and

server implementations. Table 5 shows the files you need to create COM clients and servers.

Table 5. Files used to generate clients and servers

Client	In-proc server	Proxy DLL	Out-proc server
(ui)	DLLDATA	DLLDATA	(main)
IID	IID, Proxy	IID, Proxy	IID
DCOM RTL	DCOM RTL	DCOM RTL	DCOM RTL
	(functions)		(functions)

You must build all four images. Why? Because the client doesn't know until run time whether it needs an in-proc server or an out-proc server. If the client chooses an in-proc server, the DCOM RTL merges the in-proc DLL into the client's address space. If the client chooses an out-proc server, the DCOM RTL loads the proxy DLL into the client's address space. The client then uses the proxy DLL to communicate with the remote proxy DLL and the out-proc server.

To create a client you need the following files:

- User interface file
- IID file (generated by MIDL compiler)
- DCOM RTL

To create a Proxy DLL you need the following files:

- DLLDATA file (generated by MIDL compiler)
- IID files (generated by MIDL compiler)
- Proxy files (generated by MIDL compiler)
- DCOM RTL

To create an in-proc server you need the following files:

- DLLDATA file (generated by MIDL compiler)
- IID files (generated by MIDL compiler)
- Proxy files (generated by MIDL compiler)
- Functions file
- DCOM RTL

To create an out-proc server you need the following files:

- Main file
- IID files (generated by MIDL compiler)
- Functions file
- DCOM RTL

For more information about this process and the compiler switches used for compiling the files, see the *OpenVMS Connectivity Developer Guide*.

Step 4: Link the COM app

As described in Step 3, you must build both client and component images. Because you can implement a component as either an in-process component or an out-of-process component, you must build a shareable image and an executable image.

If you are creating a new interface, you must also build a proxy shareable image. The proxy shareable image provides an interface-specific object that packages parameters for that interface in preparation for a remote method call. A proxy runs in the sender's address space and communicates with a corresponding stub in the receiver's address space. For more information see the *OpenVMS Connectivity Developer Guide*.

Part III: COM for Tru64 UNIX

This part includes the following chapters:

- Chapter 8. Overview of COM on Tru64 UNIX**
- Chapter 9. COM on the Compaq Tru64 UNIX platform**
- Chapter 10. COM run-time environment on Tru64 UNIX**
- Chapter 11. COM for Tru64 UNIX APIs**
- Chapter 12. The COM registry on Tru64 UNIX**
- Chapter 13. Security considerations (authentication)**
- Chapter 14. Utilities**
- Chapter 15. Configuring Tru64 UNIX for COM**
- Chapter 16. Developing COM applications on Tru64 UNIX**

Chapter 8. Overview of COM on Tru64 UNIX

To get right to the heart of the matter, it is fairly easy to build and run COM applications in a heterogeneous environment of OpenVMS, Tru64 UNIX, and Windows NT servers and clients.

Running existing COM applications where Tru64 UNIX is added to the client/server mix requires only that you install a COM run-time environment on the UNIX machine and register the COM applications.

Building new COM applications on COM-enabled Tru64 UNIX requires only that you install the COM run-time and the software development environment, which is an optional installation subset.

With the exception of GUI-based APIs, COM programmers will find the OpenVMS and Tru64 UNIX COM programming environment as familiar as Windows NT.

Tru64 UNIX, however, is not Windows NT and the differences in architecture do cause differences in the implementation of COM, especially in the areas of security, compiling COM applications, use of the Registry, and platform configuration. That said, the user will find that COM is COM, no matter which operating system is used.

If you are a Tru64 UNIX system administrator who is familiar with the operating system's programming environment, you may find the general COM requirements for security, compilation, Registry, and platform configuration to be new. If you are an NT user who is familiar with COM application requirements, you may find the implementation of those requirements on Tru64 UNIX to be new.

This section should help both the UNIX administrator and the NT programmer understand the little that has changed with the addition of Tru64 UNIX to the set of COM platforms. The chapters in this part provide information about:

- How to secure remote activation of an application.
- How to compile COM shared libraries or executables on UNIX.
- How to register that application on a Tru64 UNIX machine.

These chapters help you see both the similarities and the differences of COM on the Tru64 UNIX platform. They don't provide you with all the details of installing and using COM on Tru64 UNIX. That job is best left to the product documentation.

The historical context

It might be helpful to look at the historical context to understand how easily a COM-enabled Tru64 UNIX server or client fits into distributed COM environment.

Over the past several years, Digital Equipment Corporation (and, later, Compaq Computer Corporation) collaborated with Microsoft Corporation, Hewlett-Packard Company, and Siemens Nixdorf Information Systeme AG to implement and test distributed COM on 32-bit, and 64-bit operating systems. The collaboration resulted in a COM reference platform that runs on a 32-bit UNIX system (represented by Sun Solaris) and a 64-bit UNIX system (represented by Compaq Alpha running Tru64 UNIX). The Open Group, for a cost, makes this reference platform implementation available to the public as a starting point for a platform-specific COM implementation.

Of more importance to the readers of this book is a collaboration between Compaq and Microsoft formed to implement COM on Compaq's 64-bit Alpha platforms. The result is the COM for Tru64 UNIX product delivered as an Associated Product with the operating system.

Industry analysts predict that, within the next year, 95% of all large organizations will have a heterogeneous environment. And many of those will include a high performance Compaq Alpha platform, such as OpenVMS or Tru64 UNIX, working together with Windows NT operating systems.

Compaq desktop systems running NT combined with Tru64 UNIX in a heterogeneous environment offer the following benefits:

- Reduces the risk in development and deployment of new and existing applications.
- Leverages existing knowledge and experience across heterogeneous platforms.
- Expands the options and capabilities for deployed systems.

Organizations can use COM on Compaq platforms to leverage their investments in Windows-based systems and extend the productivity of their familiar applications to an enterprise system.

The Compaq partnership with Microsoft did not end with the first implementation of COM for Tru64 UNIX. Compaq and Microsoft continue to collaborate on the future directions of COM and on the port of more sophisticated COM features from Windows NT to Tru64 UNIX.

The strategic context

The Compaq AllConnect Program defines *interoperability* as the ability for dissimilar systems to work together, while recognizing that there are different levels of interoperability, including coexistence, integration, and migration. The program provides a strategic framework against which the company delivers an array of products for interoperability between Tru64 UNIX and Windows NT. Among these products is COM for Tru64 UNIX.

Under the umbrella of AllConnect, the Tru64 UNIX products include:

- Management tools for a mixed UNIX and Windows NT environment.
- Resource sharing tools for common access to files and printers.
- Software development tools for the creation and deployment of applications.

The goal of the AllConnect Program is to make the development, deployment, and management of enterprise solutions across Tru64 UNIX and Windows NT as easy and seamless as possible. The strategy starts with application access by means of COM on the Tru64 platform and extends to a shared infrastructure of security, directory services, application management and cross-platform software development.

As an existing Tru64 UNIX user, you can view Windows NT and Tru64 UNIX interoperability as a way to preserve existing and future investment in UNIX while introducing Windows servers (or clients) into your computing environment.

Compaq recommends that critical customer applications use a three-tier division of labor to take advantage of the strengths unique to Tru64 UNIX and Windows NT. That is, you construct a heterogeneous environment in such a way that the first tier manages the interface, the second tier provides the business logic, and the third tier manages the database with each tier based on the operating system most appropriate to the job.

The design center for Windows NT/Tru64 UNIX interoperability includes Windows servers in the second tier, managing business logic, and providing a front end for the Tru64 UNIX system. Tru64 UNIX applies its strengths of performance, scalability, reliability, and availability to the management of third-tier databases.

In such an environment, the application programmer:

- Writes client applications on Windows that communicate directly with Tru64 UNIX COM servers, just as though they were NT servers.
- Writes second-tier applications on NT just as though the third tier was also on NT (when, in fact, it is on Tru64 UNIX).

- Writes third tier applications on Tru64 UNIX just as though they were being written for an NT server.

The reality

By the first quarter of 2000, the first version of COM for Tru64 UNIX has been in customers' hands for six months.

It is a risky proposition to discuss software releases before they happen. However, Compaq engineers continue to collaborate with Microsoft and they continue to develop enhancements for the existing product.

This section describes the planned enhancements that Compaq is implementing for its COM product, a proposition which is very nearly risk free. The section will leave the discussion of long-term enhancements to marketing and others who live on the edge.

- COM for Tru64 UNIX Version 1.0 – This Compaq release ported COM, minus the GUI-specific APIs, to Tru64 UNIX. It also implemented the run-time system needed to support COM applications and the development tools needed to create COM server applications on UNIX. The largest parts of the porting task were the implementation of NT services in the foreign land of UNIX and creation of a COM-specific Registry. The largest hole in this first release was security, or the lack of authenticated COM.
- COM for Tru64 UNIX Versions 1.1 and 2.0 – These Compaq releases fill the current hole and provide security in the form of authenticated COM. Compaq is committed to the follow-on work for COM and the features planned for near-term releases (authentication, surrogate process support, encryption libraries, and ATL support) will be available with the publication of this book. To avoid instant irrelevance, this book contains descriptions of security, encryption, and ATL as they will be implemented.

Table 6 provides a summary of the COM for Tru64 UNIX releases.

Table 6. Summary of COM for Tru64 UNIX releases

Area	COM for Tru64 UNIX Version 1.0	COM for Tru64 UNIX near-term enhancements
Client requests	Authenticated on Windows NT only.	Authenticated on Windows NT and Tru64 UNIX.
Security	On Windows NT, servers can run with the client's identity. On Tru64 UNIX, security is disabled. Client and Server applications must include a CoInitializeSecurity call with authentication set to NONE and impersonation set to ANONYMOUS.	On Tru64 UNIX, security authentication is enabled. Because Tru64 UNIX does not allow thread-level security, impersonation is not allowed. The NT LanManager Security Support Provider is supported by means of a password-protected pass-through to a Windows NT Primary Domain Controller.
Outbound COM requests	Authenticated on Windows NT only.	Authenticated on Windows NT and Tru64 UNIX.
Registry access	On Windows NT, Registry access is controlled by NT credentials. On Tru64 UNIX, Registry access is controlled by application credentials. At the system level, an NT daemon controls Registry access.	Same.
Event logging	Windows NT only.	Windows NT only.
ActiveX Template Library	Not supported.	Supported.
Encryption	Not supported.	Supports 40-bit encryption.
Surrogate Processes	Not supported.	Supported.

Implementation details

The Tru64 UNIX COM port is fully compliant with Microsoft COM, fully 64-bit, fully international, and fully integrated with the Compaq Tru64 UNIX operating system.

COM on Compaq platforms allows you to:

- Produce or consume COM objects on Tru64 UNIX or OpenVMS that are usable by COM applications on any platform.
- Port COM server applications from Windows NT operating environments to Tru64 UNIX or OpenVMS.
- Create wrappers for existing Tru64 UNIX or OpenVMS applications, providing COM access to those applications by clients running Windows.
- Develop new, distributed, applications that take advantage of the COM distribution mechanism. These applications can make the most of COM reuse, version independence, and language independence capabilities.

COM ported to the Compaq Tru64 UNIX platform includes:

- MIDL, the Microsoft Interface Definition Language Compiler that you use to create the component object interface.
- ATL (ActiveX Template Library) Version 2.1.
- The interfaces and APIs defined by Microsoft as those needed to support COM on non-Windows platforms.
- Support for the creation of server applications in C and C++ and support for client applications in C, C++, Java, and Visual Basic.
- Support for COM capabilities such as Monikers, OLE Automation, Uniform Data Transfer (UDT), Connectable Objects, structured storage, surrogate processes (including custom surrogates), and type libraries.
- Remote Procedure Call System Services (RPCSS), which includes Service Control Manager (SCM), Object Exporter, and Running Object Table.
- Multi-Threaded (MTA) and Single-Threaded Apartment (STA) threading model.
- Microsoft Remote Procedure Call (MS RPC) that provides transparent communication so that remote clients appear to directly communicate with the server. MS RPC is wire-level compatible (as opposed to call-level compatible) with the Open Software Foundation (OSF) implementation of Distributed Computing Environment RPC (DCE RPC).
- Registry, the database of COM components and relevant configuration information, and Registry tools that allow you to modify Registry contents.
- Security, in the form of call security that allows a client or server to apply an appropriate security level to method calls.

- Security in the form of the Security Support Provider Interface (SSPI) standard that defines security providers that can be accessible to COM server applications. Microsoft NT uses the Windows NT LanManager Security Support Provider (also called NTLM SSP) . COM for Tru64 UNIX supports NTLM SSP calls for client authentication.
- Internationalization capability, including Unicode support of wide characters. COM for Tru64 UNIX converts platform-specific differences between the 64-bit implementation, which uses a 4-byte wide character, and other implementations, which use a 2-byte wide character.
- Support for localized messaging, or the localization of system messages in COM libraries and utility applications.
- Error handling conventions that allow COM objects in different environments to share status information.

Chapter 9. COM on the Compaq Tru64 UNIX platform

This chapter describes the implementation of COM on Tru64 UNIX. While there are differences between COM on its native Windows NT platform and on Tru64 UNIX, COM users from NT will find the Tru64 UNIX implementation remarkably close to their environment.

This part does not describe how to use COM on the Tru64 UNIX platform. That information is too specific to the platform and, since Compaq ships documentation free with the COM for Tru64 UNIX product, it is more appropriate for you to download the software, use the documentation, and see for yourself.

The documentation for COM on Tru64 UNIX describes how to install and use the COM run-time, and discusses the idiosyncrasies of compiling, building, and registering COM applications on the Tru64 UNIX platform.

You can view the Tru64 UNIX COM documentation at

www.tru64unix.compaq.com/com/

The COM for Tru64 UNIX product also contains a demonstration software package that illustrates the use of COM across heterogeneous platforms of NT clients and UNIX and OpenVMS servers. The software is very similar to the demonstration software included with this book and described in the COM demo CD-ROM appendix. In both cases, the demonstration software simulates an order entry and inventory system in a multi-tier environment with components that operate across the platforms.

The implementation of COM on Tru64 UNIX

Table 7 summarizes the implementation of COM on the Compaq Tru64 UNIX platform. The remaining chapters in this part contain details about this implementation.

Table 7. The Implementation of COM features on the Tru64 UNIX platform

COM attribute on NT	On Tru64 UNIX
Run-time Environment, NT Services, rpcss, and Endpoint Mapping	<p>The COM for Tru64 UNIX port implemented COM services, such as the Service Control Manager, on UNIX.</p> <p>To resolve environmental differences between the platforms, Compaq also implemented an NT daemon (ntd) and libmutant library to provide COM applications on UNIX with required NT services.</p> <p>Finally, the port included a privileged helper application (coolrip) to map server/client communication to a privileged communications port.</p>
COM APIs and Interfaces	<p>Except for Windows-specific (GUI) APIs and Interfaces, Compaq ported the COM APIs and Interfaces to Tru64 UNIX. These APIs and Interfaces meet the Component Object Model Specification interoperability requirements.</p>
Registry	<p>Compaq implemented a COM Registry on Tru64 UNIX that is similar to the Registry on Windows NT.</p> <p>The major difference between the registries lies in the area of Registry management. Also, unlike the Registry on Windows NT, the Tru64 UNIX Registry does not control the entire system.</p>
Event Logging	<p>Event logging is implemented on OpenVMS and described in Part II. It is not implemented on Tru64 UNIX. Events are an NT feature, not a COM feature, and were not included in the port of COM to Tru64 UNIX.</p>
Security	<p>Compaq implemented a security subsystem that provides authentication between the Tru64 UNIX server and a Windows NT Primary Domain Controller. The subsystem is password protected and supports the NTLM SSPI.</p>
Utilities	<p>Compaq implemented a number of COM-based utilities on Tru64 UNIX. The utilities assist you in configuring application security in the Registry, converting between directory tree structures and compound files, and viewing the current version of COM libraries and utilities.</p>

continued

Table 7 (con't). The Implementation of COM features on the Tru64 UNIX platform

COM attribute on NT	On Tru64 UNIX
UNIX Platform Configuration	Compaq implemented a configuration file, called dcomconfig, that automates the setting of Tru64 UNIX environment variables. You can also customize the file to suit your platform.
Compiling a Server Application	Compaq supports C, C++, Visual Basic, or Java client applications and supports the development of C and C++ server applications on Tru64 UNIX. The COM development environment implemented on Tru64 UNIX includes the standard MIDL compiler and the bundled C++ compiler and libraries to produce executables, DLLs, or type libraries.
Coding an Application	You code a COM application for execution on Tru64 UNIX just as you would for execution on Windows NT.

Implementation challenges

As Compaq ported COM to the 64-bit Tru64 UNIX platform, there were a number of generic issues that needed to be solved. The implementation details of Compaq's engineering solutions are described in subsequent chapters, but this list provides you with an overview of the problems and their resolutions.

Porting code and COM functions

The COM implementation on Tru64 UNIX is MS RPC based, which minimized compatibility issues with Microsoft's code. However, there were other issues in porting optimized Windows NT COM code to the 64-bit Tru64 UNIX environment. Some of these issues arose from interface (command line vs Windows GUI) and operating system differences and are discussed in Chapter 10. Other issues were more specific to the code and are described in the Porting Considerations appendix.

COM APIs and interfaces

With the exception of Windows-based GUI APIs, the port includes all of the interfaces and functions defined by Microsoft as those needed to support COM on non-Windows platforms.

In-process and out-of-process servers

As with COM on Windows NT, you can use the development environment on Tru64 UNIX to compile a COM application as a DLL (shared library), type library, or executable.

Security configuration

As described in Chapter 13, the COM for Tru64 UNIX implementation establishes application and system Registry entries with a command line utility called `olecnfg`, which is functionally similar to the Windows NT utility, `dcomcnfg`.

Service Control Manager

As described in Chapter 10, the COM for Tru64 UNIX implementation uses a daemon, called `rpcss`, to provide the services of SCM, the object exporter, running object table, and (with the aid of a root-owned helper application) endpoint mapping. The implementation also uses an NT daemon (`ntd`) and a libmutant library to provide the Win32 services that applications normally find in a Windows NT environment.

Threading model

The COM implementation on Tru64 UNIX includes support for the single-threaded apartment (STA) and multithreaded apartment (MTA) threading models.

Security

As described in Chapter 13, the current COM implementation on Tru64 UNIX supports the NT LanManager (NTLM) Security Support Provider by means of a method called NTLM pass-through. That is, a password-protected communication link is established with an NT Primary Domain Controller which is capable of authenticating NT clients.

Creating and running a server application on Tru64 UNIX

In general, the steps you follow to compile, link, and register a COM application for launch on a Tru64 UNIX remote server are:

1. Define the object's interfaces with IDL (Interface Definition Language) and create an IDL file (`.idl`). Also create a registration file (`.reg`) containing globally unique identifiers (GUIDs) for the interfaces. Most programmers use tools such as Visual C++® that automate much of the writing of IDL and registration files, or they take existing files and modify them for new uses.
2. Use the MIDL compiler to process the IDL file. The MIDL compiler generates C/C++ compatible modules (client and server source files, header files, and, optionally, an application configuration file) that you can compile into a COM

executable or shared library. Depending on the MIDL switches you specify and the content of the IDL file, MIDL can also generate a type library.

3. For a proxy/stub library (.so), create a Module Definition File (DEF file) that specifies the library name that exports the `DllRegisterServer`, `DllUnregisterServer`, `DllCanUnloadNow`, and `DllGetClassObject` functions. Use the `makedef` Utility to compile the DEF file into a file suitable for input to the C++ compiler.
4. Use the C++ compiler to compile and link the source and header files and required libraries into proxy and stub libraries or executables. By default, the compiler produces an executable. You can choose to have the compiler produce a shared library (equivalent to a Microsoft Dynamic Link Library), which can be used in an in-process server or, with a surrogate process, as a remote application DLL.
5. Make sure the COM run-time system is installed and running on Tru64 UNIX and register the COM application on the server. You can use the `sermon` utility to import an existing registration file, or create a new registration file, containing the GUIDs for the proxies and stubs or type libraries. Use the `sermon commit` command to merge the file into the server COM Registry. To register a shared library, use the `regsvr` utility.
6. Register the application on the Registry of each client that calls the COM server application.

After the server application is registered, a client application can activate it remotely.

Running COM Applications on Tru64 UNIX

The steps for running a COM application in a heterogeneous Windows NT/Tru64 UNIX environment are familiar ones for NT application programmers.

The remaining chapters of this part provide you with enough detail to gauge the level of COM's integration with Tru64 UNIX and to see for yourself how familiar the environment is.

Before running an application:

- The COM run-time must be active on the Tru64 UNIX system.
- There must be a network connection between the UNIX and NT machines, with TCP/IP configured on both the client and server platforms and the server node names and IP addresses registered with the Windows NT client platforms.
- Applications must be registered on the client and server machines.

To start a server or client application on Tru64 UNIX:

- Invoke the application name from the command line prompt. Optionally, append the switches defined for your application in its code. Some of the traditional COM application switches are `-r` for registering the server application and `-e` for running the application interactively.

To start the application on NT:

1. Register the application.
2. Enable DCOM (Distributed COM is enabled by default on the Tru64 UNIX system).
3. Unless security is enabled, set the Default Authentication Level to NONE.
4. Unless security is enabled, set the Default Impersonation Level to ANONYMOUS.
5. If the Windows NT machine is a server, set the application security properties such that launch and access permissions are checked. If the Windows NT machine is a client, you can skip this step.

Chapter 10. COM run-time environment on Tru64 UNIX

If you want to run an existing COM application on a Tru64 UNIX server or client, there are only two things you need to do:

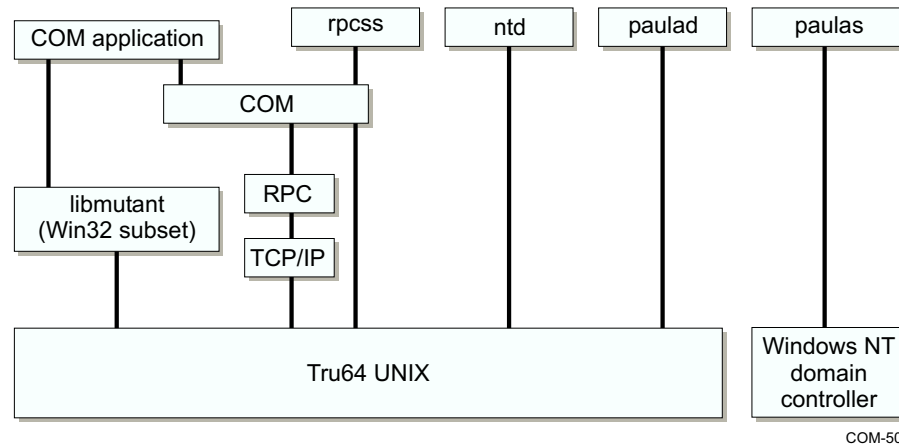
1. Make sure that your Tru64 UNIX platform has the following characteristics:
 - An Alpha AXP™ Workstation with a CD-ROM drive. Compaq suggests that the platform have 128MB RAM.
 - The Version 4.0D (or later) or the Version 5.0 (or later) Tru64 UNIX operating system. The Version 4.0D and 4.0E operating system products were called Digital UNIX prior to the merger of Digital and Compaq. After the merger, the operating system was renamed to Compaq Tru64 UNIX. This book uses the current name, Tru64 UNIX.
2. Install, configure, and activate the COM run-time environment that Compaq has tailored for Tru64 UNIX. This chapter describes that run-time environment and how it compares to the native Windows NT COM platform.

The COM run-time environment on Tru64 UNIX duplicates the Windows NT services and infrastructure that support the execution of COM applications. The COM run-time bridges the gap between Windows NT and UNIX and provides a compatible environment for the execution of COM applications.

There is another thing you must do if you wish to develop new COM applications on a Tru64 UNIX server in addition to running existing applications. You must install the COM Software Development Kit when you install the COM run-time environment. The development kit contains the MIDL compiler and other tools you must have to build COM applications. Chapter 16 describes how to use the compilers and utilities of the Software Development environment.

To illustrate how the components of the run-time environment operate together, consider Figure 27 and the following two sections that inventory the COM run-time and programming environments provided on Tru64 UNIX.

Figure 27. Inventory of COM for Tru64 UNIX



An inventory of the COM run-time environment

The COM for Tru64 UNIX run-time environment consists of services and daemons that provide the functions used by all COM applications on the system.

Win32 Subsystem. The Win32 subsystem implements the Win32 services that COM applications require, such as file input and output and dynamic library loading. The Win32 Subsystem includes:

- **ntd** — The NT daemon that emulates Windows NT services, maintains the COM Registry, and is responsible for activating requested server applications.
- **libmutant** — The shared library that resolves Win32 calls.
- **paulad and paulas** — The security subsystem that authenticates communication between the Tru64 UNIX server (paulad) and the Windows NT Primary Domain Controller (paulas).

COM run-time. The COM run-time includes the services that provide basic COM functions, such as NT service manager functions. The COM run-time consists of:

- **rpcss** — The daemon process that provides the functions of the Service Control Manager (SCM), Object Exporter, and Running Object Table
- **The TCP/IP well-known port (port 135)** — The daemon **rpcss** (by means of a root-owned helper application called **coolrip**) uses port 135, the reserved TCP/IP well-known port, as the communication link for offering its services between the Tru64 UNIX server and COM clients. TCP/IP and **coolrip** are automatically enabled on Tru64 UNIX systems. However, if your COM for Tru64 UNIX system is in a

heterogeneous environment with a Microsoft Windows NT client, be sure that your Windows NT client has TCP/IP enabled.

Run-time Utilities. The implementation of COM for Tru64 UNIX includes a set of command-line utilities and a configuration file you use to start, stop, or modify run-time services and daemons:

- **dcomsetup** — The COM Setup file used to initialize the system Registry and start and stop ntd, paulad, and rpcss. Compaq implemented the COM Setup file on Tru64 UNIX as a convenience; behind the scenes, dcomsetup calls ntwopper and sermon to perform its functions.
- **ntwopper** — The utility used to start ntd, paulad, and rpcss.
- **sermon** — The utility used to stop ntd and modify the Registry.
- **regsvr** — The server registration utility used to add or remove Registry information on shared libraries.
- **olecnfg** — The utility used to configure the authentication level required for activating a server application.
- **stgview** — The utility used to display the contents of a structured storage compound file.
- **df2t** — The utility that converts a directory tree structure to a structured storage compound file (and vice versa).
- **dcomver** — The utility that displays version information.

COM Configuration File. The file `/etc/dcomconfig` defines environmental variables and configures COM services on the Tru64 UNIX operating system. For example, the value you attach to the `COOL_PERSISTREG` variable in `dcomconfig` defines the location of Registry files on disk.

COM Registry. The UNIX system database that contains information about COM objects and enables a client application to find and execute the server application. An initialized Registry on UNIX supports COM applications in the same manner as the Windows NT system Registry, but without the implied responsibility of being a system-wide Registry.

COM applications on Tru64 UNIX cannot execute without the ntd, paulad, and rpcss processes running on the machine. Also, the Registry on Tru64 UNIX must be initialized and the application must be registered. The `dcomsetup` configuration file initializes the Registry and starts the daemons automatically.

An inventory of the COM programming environment

The COM for Tru64 UNIX programming environment (or Software Development Kit) consists of the compilers, tools, and libraries you must have to create and compile a COM application on Tru64 UNIX. COM development is an optional installation subset on Tru64 UNIX, which means that a system administrator can install the COM run-time and configure the UNIX machine as a “run-only” environment.

The components that make up the application programming environment include:

Programming Tools.

- **MIDL** — The Microsoft Interface Definition Language compiler. For the first version of COM for Tru64 UNIX, Compaq ported Version 3.00.44 of the MIDL compiler. Compaq has contractual arrangements with Microsoft to ship an updated version of the MIDL compiler at a later date.
- **ATL** — The Microsoft ActiveX Template Library, Version 2.1.
- **C++** — The Tru64 UNIX C++ compiler. The C++ compiler requires a license. However, you can also use the C compiler, which is bundled with the operating system, to create server applications on Tru64 UNIX.
- **makedef** — A Bourne shell script that you use to compile a Windows NT Module Definition File.

Shared Libraries. A shared library on Tru64 UNIX is very similar to a Dynamic Link Library (DLL) on Windows NT. The COM shared libraries perform specific functions for the server application. The programmer links the shared libraries to the server application when it is compiled as an executable. Depending on the calls the server application makes, some shared libraries are required (must be linked during compilation) and some are optional.

Type Libraries. Two type libraries, `stdole32` and `stdole2`, are standard libraries describing automation interfaces and are implemented on Tru64 UNIX to support interfaces derived from `IDispatch`, `ITypeInfo`, and `IEnumVARIANT`. Programmers include references to these type libraries within an application’s IDL file.

Chapter 16 discusses the process you use to compile and link COM applications and describes the required and optional libraries.

The Win32 services on Tru64 UNIX

On a Windows system, the supporting infrastructure for COM applications is always there as integrated functions of Windows NT. However, on a non-Windows server, such

as Tru64 UNIX, the infrastructure is built into the COM run-time, which must be installed and running.

At a minimum, the run-time must provide NT services, Service Control Manager functions, a communications link, and a Registry. All of these prerequisites (ntd, rpcss, coolprip, and the Registry, respectively) are implemented in the COM for Tru64 UNIX run-time. Once installed on Tru64 UNIX, you start and stop the run-time system with individual command line commands or, collectively, with the COM Setup script (dcomsetup).

There are a few UNIX operating system caveats that you need to be aware of when starting and stopping these services. (This is not an issue when using dcomsetup, which automatically accommodates these caveats.) The idiosyncrasies of ntd and the Win32 services can also give you some insight into the marriage of Windows NT services and the Tru64 UNIX operating system.

- You must start the services from a non-root account.

The UID where you start ntd (whether by using dcomsetup or ntwoppper) becomes the UID for ntd and the owner and group settings of that UID become the owner and group settings for ntd.

When an authenticated client COM application requests a server object, rpcss fields the request and uses ntd to launch the object under the UID that ntd is configured to run as. The ntd daemon fulfills the request based on the UID settings for ntd.

For example, if the UID you start ntd from allows execution by group “com,” ntd applies that same set of permissions to the remotely activated object (server application). If a client launch or access request does not meet the permissions set for the server application, the client request fails with an “Access Denied” error.

- There should be only one set of services running under a single owner.

Because the Tru64 UNIX operating system places restrictions on system-wide processes and to ensure that system-wide COM resources (Registry, daemons, and services) are properly shared, Compaq recommends that there be only one set of Win32 services active on the Tru64 UNIX machine. For example, the system administrator might want to start the services from a password protected account.

On a development server, where a number of programmers are debugging applications, single ownership of services can become a nuisance. In this case, programmers can set the value of the COOL_NTD_TVTUNER environment variable to allow multiple ntd processes under different accounts on the same machine. But in a production environment, be safe and have only one set of Win32 services active. (See Chapter 15 for more information.)

- Be aware of the order in which you start and stop services: start ntd, then paulad, then rpcss. Stop rpcss, then paulad, then ntd.

If paulad terminates, stop all applications and restart ntd. You must restart ntd because the context of the secure channel that was established between ntd and paulad, is lost upon termination. If you use the dcomsetup menu to start and stop services, it will take care of the start and stop order for you.

Starting and stopping Win32 services from the command line

You can start and stop COM run-time services (ntd, paulad, and rpcss) on Tru64 UNIX using the ntwopper and sermon utilities from the UNIX command line. You use ntwopper to start ntd, paulad, and rpcss. You use sermon to stop these services. However, to make your life easier and to avoid errors when starting and stopping Win32 services, Compaq recommends you use the COM Setup menu, dcomsetup, described in the next section.

Starting and stopping Win32 services automatically

The Compaq implementation of the COM run-time environment on Tru64 UNIX includes a COM Setup file called dcomsetup. You run dcomsetup from a non-root account and use the dcomsetup menu to initialize the COM Registry and start and stop ntd, paulad, and rpcss.

The COM Setup menu automates the startup and shutdown processes and ensures that all the services start and stop cleanly. The configuration file checks for a conflict with other running daemons and services and performs a cleaner shutdown than if you manually stop the services. The configuration file also ensures that daemons and services start and stop in the proper order.

The dcomsetup menu options include:

- Initializing the Registry.
- Displaying status of the COM services.
- Starting and stopping the services.
- Verifying the correct installation of COM for Tru64 UNIX.

The NT daemon (ntd)

The ntd daemon is the process that emulates Windows NT services and maintains the COM Registry on the server. COM for Tru64 UNIX cannot execute unless the ntd process is running. (COM also requires a running rpcss and paulad process.)

The System Administrator initially starts ntd (from a non-root account) as part of the COM for Tru64 UNIX installation. Once started, ntd continues to run even after COM users log off the system.

Unless you alter the COM configuration file on Tru64 UNIX, the COM environment can have only one running ntd. If you attempt to start ntd by selecting the *Initialize* or *Start* options from the dcomsetup menu and there is an active ntd running, dcomsetup will fail to establish a running COM environment even though the utility seems to stop and start the daemon. This is the most common problem that users encounter when starting the COM run-time environment on UNIX.

UNIX account management, which includes user identity, permissions, and process ownership differs greatly from process ownership on Windows NT. A user can run dcomsetup and attempt to start ntd under a non-root account on Tru64 UNIX, but if ntd is running under another user identifier (UID) or is running under root, the user's attempt will fail.

It can get messy. If the ntd process that you are stopping is owned by another user, you will have to log in as root and use a sermon shutdown command to stop the daemon. Also, if you use the operating system kill command to stop ntd, the process stops but the Registry is not updated and ntd leaves files behind that can prevent an ntd restart. To fix it, you must remove COOL_NTD files from the UNIX machines /tmp directory.

All in all, you will find it easier to start and stop ntd (and the other services) with dcomsetup.

The Remote Procedure Call system services (rpcss)

RPCSS (Remote Procedure Call System Services) is the collective term for the Service Control Manager (SCM), Object Exporter, and Running Object Table (ROT).

The daemon process, rpcss, provides these services on Tru64 UNIX in a manner similar to that of the OLE Service Control Manager on Windows NT. The Tru64 UNIX daemon responds to a call from the SCM on a remote client system and launches a requested server application. Just as on Windows NT, the Running Object Table keeps track of active server applications and provides client applications with the means to connect to a running server application rather than to launch it.

Endpoint mapping (coolprip)

On Windows NT, rpcss also includes the Endpoint Mapper that listens to the TCP/IP well-known port (port 135) for calls.

On COM for Tru64 UNIX, however, there are issues that cause rpcss to use another process to perform Endpoint Mapper functions. Specifically, rpcss on Tru64 UNIX must

use shared libraries that are outside the system library directory. These libraries are restricted to non-root processes in order to perform Endpoint Mapper functions. However, because the TCP/IP well-known port is a reserved port, the process that establishes access to the port (the Endpoint Mapper function) must be a superuser, or root-owned process.

To resolve this conflict, rpcss uses a surrogate, root-owned helper process, called coolprip as the Endpoint Mapper. The coolprip process provides access to the privileged TCP/IP port and listens for incoming object calls.

The COM installation process on Tru64 UNIX installs coolprip under root ownership with the appropriate UID and path. The coolprip process becomes active when rpcss starts. No other action is required.

Network protocols, Windows NT, and Tru64 UNIX

Windows NT can use a number of different network protocols. Be aware, however, that Tru64 UNIX uses only TCP/IP (Transmission Control Protocol/Internet Protocol). You need simply to make sure that the Windows NT client has TCP/IP enabled in order for applications to communicate between the two systems. Knowing this bit of trivia can help you to avoid slow communication between the two systems.

If you use a COM for Tru64 UNIX system as a server in a Tru64 UNIX server and Microsoft Windows NT client environment, and your NT client uses multiple protocols, you can speed the search for the TCP/IP protocol made by COM calls to Tru64 UNIX by using TCP/IP as the first protocol entry in the client Registry.

To do so, enter ncacn_ip_tcp (which translates to the TCP/IP transport protocol) as the first entry in the following Registry key on the Windows NT client machine:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc\DCOM Protocols
```

Reboot the machine to have the change take effect.

Chapter 11. COM for Tru64 UNIX APIs

API support with COM for Tru64 UNIX

Compaq ported all of the standard, non-Windows, COM APIs to Tru64 UNIX. By “standard, non-Windows,” Compaq means that it did not port the GUI-based APIs, such as drag-and-drop, which make no sense on a Tru64 UNIX machine. However, Compaq did port all of the APIs and interfaces that programmers require to write COM applications that conform, in every way, to the COM standard.

The interface and API categories that are available to an application developer using the COM implementation on Tru64 UNIX are defined in the Component Object Model Specification, available at www.microsoft.com/com/comdocs.asp. Because the list of supported APIs grows with each version of COM on the Compaq platforms, refer to the documentation for COM on Tru64 UNIX for the most up-to-date list.

The APIs supported on Compaq platforms are a superset of the APIs that Microsoft requires for conformance to the interoperability standards.

Microsoft worked closely with Compaq to port COM to the Compaq platforms. To ensure that the implementation of COM is consistent across the various UNIX systems, Microsoft provides a conformance test that COM implementations must pass. (The Tru64 UNIX implementation passes these tests. The tests are also known as Interoperability Tests and are available from The Open Group.)

The tests verify that the COM implementation fulfills a basic set of performance requirements and that the level of API support is standardized. These standards and tests make sure that applications can be relied upon to run across platforms.

The COM for Tru64 UNIX product documentation contains a complete list of the API functions and interfaces that were ported to the UNIX platform. The following list summarizes the supported categories and the APIs and interfaces within each category.

Custom Services

IClassFactory, IClassFactory2, CoCreateInstance, CoCreateInstanceEx, CoGetClassObject, CoGetInstanceFromFile, CoGetInstanceFromIStorage, CoGetPSClsid, CoRegisterClassObject, CoRegisterPSClsid, CoReleaseServerProcess, CoResumeClassObjects, CoRevokeClassObject, CoSuspendClassObjects, DllGetClassObject.

DLL Server Management

CoFreeAllLibraries, CoFreeLibrary, CoFreeUnusedLibraries, CoLoadLibrary, DllCanUnloadNow.

Enumerators and Enumerator Interfaces

IEnum*, IEnumString, IEnumUnknown.

Error Handling

ICreateErrorInfo, IErrorInfo, IErrorLog, ISupportErrorInfo, CreateErrorInfo, GetLastErrorInfo, SetErrorInfo.

Generic

IConnectionPoint, IConnectionPointContainer, IEnumConnectionPoints, IEnumConnections, IProgressNotify, IProvideClassInfo, IUnknown.

Helper Macros

FACILITY_NT_BIT, FAILED, HRESULT_CODE, HRESULT_FACILITY, HRESULT_FROM_NT, HRESULT_FROM_WIN32, HRESULT_SEVERITY, IS_ERROR, MAKE_HRESULT, MAKE_SCODE, SCODE_CODE, SCODE_FACILITY, SUCCEEDED.

Initialization and Memory Management

IMalloc, IMallocSpy, CoCreateStandardMalloc, CoGetMalloc, CoInitialize, CoInitializeEx, CoRegisterMallocSpy, CoRevokeMallocSpy, CoTaskMemAlloc, CoTaskMemFree, CoTaskMemRealloc, CoUninitialize.

Miscellaneous COM Functions

CLSIDFromProgID, CLSIDFromString, CoDosDateTimeToFileTime, CoFileTimeNow, CoFileTimeToDosDateTime, CoGetCurrentProcess, IIDFromString, ProgIDFromCLSID, StringFromCLSID, StringFromGUID2, StringFromIID.

Naming (Monikers)

IBindCtx, IClassActivator, IEnumMoniker, IEnumString, IEnumUnknown, IMoniker, IOleContainer, IOleItemContainer, IParseDisplayName, IROTData,

IRunningObjectTable, BindMoniker, CoGetObject, CreateAntiMoniker, CreateBindCtx, CreateClassMoniker, CreateFileMoniker, CreateGenericComposite, CreateItemMoniker, CreatePointerMoniker, GetClassFile, GetRunningObjectTable, MkParseDisplayName, MkParseDisplayNameEx, MonikerCommonPrefixWith, MonikerRelativePathTo.

National Language Support

CompareString, GetLocaleInfo, GetStringType, GetStringTypeEx, GetSystemDefaultLangID, GetSystemDefaultLCID, GetUserDefaultLangID, GetUserDefaultLCID, LCMapString.

Notification/Events

IAdviseSink, IAdviseSink2, IAdviseSinkEx, IPropertyNotifySink.

OLE Automation

IDispatch, IEnumVARIANT, BstrFromVector, CreateStdDispatch, DispGetIDsOfNames, DispGetParam, DispInvoke, DosDateTimeToVariantTime, GetActiveObject, GetAltMonthNames, RegisterActiveObject, RevokeActiveObject, SafeArrayAccessData, SafeArrayAllocData, SafeArrayAllocDescriptor, SafeArrayCopy, SafeArrayCopyData, SafeArrayCreate, SafeArrayCreateVector, SafeArrayDestroy, SafeArrayDestroyData, SafeArrayDestroyDescriptor, SafeArrayGetDim, SafeArrayGetElement, SafeArrayGetElemSize, SafeArrayGetLBound, SafeArrayGetUBound, SafeArrayLock, SafeArrayPtrOfIndex, SafeArrayPutElement, SafeArrayRedim, SafeArrayUnAccessData, SafeArrayUnlock, SysAllocString, SysAllocStringByteLen, SysAllocStringLen, SysFreeString, SysReAllocString, SysReAllocStringLen, SysStringByteLen, SysStringLen, SystemTimeToVariantTime, VarDateFromUdate, VarUdateFromDate, VariantChangeType, VariantChangeTypeEx, VariantClear, VariantCopy, VariantCopyInd, VariantInit, VariantTimeToDosDateTime, VariantTimeToSystemTime, VarNumFromParseNum, VarParseNumFromStr, VectorFromBstr.

Persistent Objects

IPersist, IPersistFile, IPersistMemory, IPersistStorage, IPersistStream, IPersistStreamInit.

Registry

RegCloseKey, RegConnectRegistry, RegCreateKeyEx, RegDeleteKey, RegDeleteValue, RegEnumKeyEx, RegEnumValue, RegFlushKey, RegGetKeySecurity, RegLoadKey, RegNotifyChangeKeyValue, RegOpenKeyEx, RegQueryInfoKey, RegQueryMultipleValues, RegQueryValueEx, RegReplaceKey,

RegRestoreKey, RegSaveKey, RegSetKeySecurity, RegSetValueEx,
RegUnLoadKey.

Remote Debugging

IORPCDebugNotify, DllDebugObjectRPCHook, DebugORPCClientGetBufferSize ,
DebugORPCClientFillBuffer , DebugORPCClientNotify,
DebugORPCServerGetBufferSize, DebugORPCServerFillBuffer,
DebugORPCServerNotify.

Remoting

IExternalConnection, IMarshal, IMessageFilter, IMultiQI, IPSFactoryBuffer,
IRpcChannelBuffer, IRpcProxyBuffer, IRpcStubBuffer, IStdMarshalInfo, ISurrogate,
CoAddRefServerProcess, CoCreateFreeThreadedMarshaler, CoDisconnectObject,
CoGetInterfaceAndReleaseStream, CoGetMarshalSizeMax, CoGetStandardMarshal,
CoIsHandlerConnected, CoLockObjectExternal, CoMarshalHresult,
CoMarshalInterface, CoMarshalInterThreadInterfaceInStream,
CoRegisterMessageFilter, CoRegisterSurrogate, CoReleaseMarshalData,
CoUnmarshalHresult, CoUnmarshalInterface.

Security

IClientSecurity, IServerSecurity, CoCopyProxy, CoGetCallContext,
CoImpersonateClient, CoInitializeSecurity, CoQueryAuthenticationServices,
CoQueryClientBlanket, CoQueryProxyBlanket, CoRevertToSelf,
CoSetProxyBlanket, CoSwitchCallContext.

Service Registration

CoCreateGuid, CoGetTreatAsClass, CoTreatAsClass, DllRegisterServer,
DllUnregisterServer, IsEqualGUID, IsEqualCLSID, IsEqualIID, OleDoAutoConvert,
OleGetAutoConvert, OleSetAutoConvert.

Structured Storage

IEnumSTATPROPSETSTG, IEnumSTATPROPSTG, IEnumSTATSTG,
IFillLockBytes, ILayoutStorage, ILockBytes, IPropertySetStorage, IPropertyStorage,
IRootStorage, ISequentialStream, IStorage, IStream, IPersistPropertyBag,
IPropertyBag, CreateILockBytesOnHGlobal, CreateStreamOnHGlobal,
FreePropVariantArray, GetConvertStg, GetHGlobalFromILockBytes,
GetHGlobalFromStream, OleConvertIStorageToOLESTREAM,
OleConvertIStorageToOLESTREAMEx, OleConvertOLESTREAMToIStorage,
OleConvertOLESTREAMToIStorageEx, PropVariantClear, PropVariantCopy,
ReadClassStg, ReadClassStm, ReadFmtUserTypeStg, SetConvertStg,
StgCreateDocfile, StgCreateDocfileOnILockBytes, StgGetIFillLockBytesOnFile,
StgGetIFillLockBytesOnILockBytes, StgIsStorageFile, StgIsStorageILockBytes,

StgOpenAsynchDocfileOnIFillLockBytes, StgOpenLayoutDocfile, StgOpenStorage, StgOpenStorageOnILockBytes, StgSetTimes, WriteClassStg, WriteClassStm, WriteFmtUserTypeStg.

Type Libraries

ICreateTypeInfo, ICreateTypeInfo2, ICreateTypeLib, ICreateTypeLib2, IProvideClassInfo2, ITypeComp, ITypeInfo, ITypeInfo2, ITypeLib, ITypeLib2, CreateDispTypeInfo, CreateTypeLib, CreateTypeLib2, LHashValOfName, LHashValOfNameSys, LoadRegTypeLib, LoadTypeLib, LoadTypeLibEx, QueryPathOfRegTypeLib, RegisterTypeLib, UnRegisterTypeLib.

Uniform Data Transfer (UDT)

IDataAdviseHolder, IDataObject, IEnumFORMATETC, IEnumSTATDATA, CreateDataAdviseHolder, ReleaseStgMedium.

International support

COM for Compaq Tru64 UNIX supports internationalization (commonly called I18N). If you are unfamiliar with I18N, see the `i18n_intro` man page on Tru64 UNIX, which provides an introduction to I18N concepts and pointers to other relevant man pages.

In COM, data often is processed internally using the wide character (`wchar_t`) data type. The ISO/ANSI C standard allows vendors wide latitude in specifying `wchar_t`, and that latitude is apparent in Microsoft and Tru64 UNIX implementations. Microsoft defines `wchar_t` to be 16 bits and to contain only Unicode. Tru64 UNIX defines `wchar_t` to be 32 bits and, while it may contain Unicode, it may also contain data encoded in other ways.

When an application marshals data of type `wchar_t`, COM truncates the data to the lower 16 bits. Any data the application attempts to pass in the upper 16 bits is lost. Unicode data is not lost because, currently, all Unicode data occupies only the lower 16 bits of a wide character. To marshal more than four bytes of non-Unicode character data in your application, use an array of characters or a long IDL data type.

Tru64 UNIX internationalization support includes logic to transparently convert `wchar_t` data between the Microsoft and Tru64 UNIX forms.

The Tru64 UNIX internationalization support contains logic for processing multi-byte characters making it possible to enter single-byte data (ASCII, Latin-1, etc.) anywhere within COM. The software also supports Asian multi-byte data in many places. United States ASCII data is, of course, acceptable everywhere in COM.

COM for Tru64 UNIX software can run under a wide variety of locales. However, if an application queries the locale, the only result returned is the Microsoft Locale ID (LCID)

for the American English locale. For example, if a server queries the locale in which a client is running, the result is 0x0409 (American English).

Applications can choose to ignore this return value. If they do, and if both client and server are running the same locale, the results should be consistent with that locale.

Because COM for Tru64 UNIX requires that data be processed internally as Unicode, you must use Unicode locales (those with a @ucs-4 extension) on Tru64 UNIX.

Chapter 12. The COM registry on Tru64 UNIX

The COM Registry on the Tru64 UNIX server is a multi-user database that contains information about COM applications (including globally unique identifiers [GUIDs] and paths). You add application information to the server Registry and to the client registries, which provides a way for client applications with the appropriate permission to locate and execute your server application.

The basic steps for implementing a Registry on Tru64 UNIX are:

1. Install the COM for Tru64 UNIX run-time environment.
2. Run `dcomsetup` to initialize the COM Registry and start up the `ntd` daemon.
3. Register applications.

Usually, the System Administrator runs `dcomsetup` to initialize the Registry at installation.

If you use the `dcomsetup initialize` option as a postinstallation task, `ntd` reinitializes the Registry from the COM for Tru64 UNIX defaults in `master.reg`. The file, `master.reg`, is the basic COM Registry and contains unique identifiers for all of the common interfaces used by COM for Tru64 UNIX.

Initialization has advantages and drawbacks.

You can see the advantages to initialization on Tru64 UNIX by comparing it to NT, where the Registry contains system configuration information in addition to object identifiers and paths. A corrupted Registry on NT can mean a corrupted system that you have to rebuild.

Because the COM Registry on Tru64 UNIX only contains application information, you can reinitialize a corrupted Registry from the defaults in `master.reg` without affecting the operating system.

The drawback to initialization of the Tru64 UNIX COM Registry is that the `dcomsetup Initialize` option erases any application information you might have registered. In other words, you are left with a basic COM Registry minus any subsequent application

registration information. There is a workaround to this problem, however, that provides an alternative to the initialize operation. On Tru64 UNIX, you can save a backup copy of `regdata.reg` that contains all of your registered applications (and is, hopefully, uncorrupted). You then use the backup copy to replace the corrupted Registry.

Registering COM applications on Tru64 UNIX

To make your server application known to clients, you must register the application (whether DLL library or executable) on the host server and on the systems that host the client applications. Once the information is in the Registry databases of the client and server machines, the client application is able to call the server and cause it to launch, or execute, the server application.

You can register an application in any of the following ways:

- The application can register itself. Because self-registering applications are self-contained, easier to maintain, and more portable, it is the preferred method.
- Manually register the application using a registration file.
- If the application is a shared library, use the `regsvr` utility.

This chapter does not discuss application self-registration, which is usually platform and application dependent. For information about coding a self-registering application, see the Component Object Model Specification description of the Reg family of APIs (`RegSetValue`, `RegSaveKey`, etc.) at the MSDN website:

<http://msdn.microsoft.com/developer/>

The following sections describe how to register an application's registration file and how to register a shared library.

Creating an application registration file

A registration file (`.reg`) is an ASCII text file that contains your application's configuration information.

To register COM application information in the Registry, either by adding the information to a registration file that you import into the Registry or by editing the Registry, you must provide the logical name and the location of the application and a GUID (globally unique identifier).

On Tru64 UNIX, you use the `uuidgen` command line utility to generate one or more GUIDs. (The `uuidgen` utility is similar to the `guidgen` and `uuidgen` GUI tools on Windows NT.)

Normally, when you create a registration file on Tru64 UNIX, you copy lines from an existing registration file into a new file and modify that new file to meet your application's requirements.

Consider the following example of the registration file for the simple server sample application installed with COM for Tru64 UNIX. To modify this file, you can use a text editor to change the following:

- Key values to values appropriate for your application's GUID.
- The level of security (LaunchPermission).
- The executable path.

You then use the `sermon reg import` command to merge the registration file into the COM Registry. The `sermon` utility uses the key and subkey information you provide in the registration file to determine where to place the data in the Registry. The following code fragment provides an example.

```
REGEDIT4
;
;Registry file for simple server on Windows/NT (4.0) and Tru64
;UNIX
;Simple DCOM Application for Local and Remote Activation
;and Access via the IStream Interface

[HKEY_CLASSES_ROOT\CLSID\{5e9ddec7-5767-11cf-beab-00aa006c3606}]
@="Simple Object Server"
[HKEY_CLASSES_ROOT\CLSID\{5e9ddec7-5767-11cf-beab-00aa006c3606}\LaunchPermission]
@="Y"
[HKEY_CLASSES_ROOT\CLSID\{5e9ddec7-5767-11cf-beab-00aa006c3606}\LocalServer32]
@="/usr/examples/com/mssimple/server/sserver"
; Use the following for NT
;@="C:\\msdev\\projects\\mssimple\\sserver\\debug\\sserver.exe"
```

If you use a Windows NT system to create your COM application's IDL file and define GUIDs for the interfaces, you can use one of the following methods to register the application on Tru64 UNIX:

- Use `regedit` to add the GUIDs to the Registry on NT. Use the `regedit export` command to create a registration file for your application from the appropriate portion of the NT Registry. Copy that registration file to Tru64 UNIX and use the `sermon reg import` command to merge those entries into the UNIX Registry.

- Use a tool such as Visual Basic that generates a registration file in conjunction with the IDL file. Use the sermon reg import command to merge that registration file's contents into the UNIX Registry.

You will likely modify the NT registration file for use on Tru64 UNIX, especially for changes in path names. A Windows NT path uses a double backslash while Tru64 UNIX uses a single forward slash. You can use either type of slash in the registration file to delimit a path for either platform, but Compaq recommends that you use the delimiter that is consistent with the platform on which you use the registration file.

Using sermon to add a registration file to the registry

On the Tru64 UNIX implementation of COM, you use a command line utility called sermon to do the following:

- List objects currently in the Registry database.
- Merge application registration files into the Registry database.
- Create a copy of an application's Registry information for export to another machine's Registry.
- Shut down ntd. While you use sermon to shut down ntd, you must use ntwoppper to start it.

The sermon utility provides a set of top-level commands, including a help command, and a set of Registry subcommands. You use the Registry subcommands, which must be preceded by the top-level command, reg, to examine and change the contents of the Registry.

The most useful sermon commands for manipulating the Registry are:

- sermon reg import filename — Copies a registration file (filename) into the system Registry.
- sermon reg export filename — Creates a copy of a registration file (filename) from the current Registry. You can then take this copy and import it to a Registry on another server.
- sermon reg commit — Writes the Registry to persistent (disk) storage. You usually execute this command after a sermon reg import command to “save” the updated Registry.

To register COM application on Tru64 UNIX, use sermon commands to update regdata.reg with the registration file containing application information. Once registered, you can remotely launch the application by means of a valid application request. You must register your application on each server machine.

The sermon import command merges a registration file with the Registry (regdata.cur) on UNIX. The utility compares the key and value data in the Registry with the data in the registration file. If sermon finds an exact match, it replaces the data in the Registry with the matched data from the registration file and adds any unmatched data from the registration file to the Registry.

The sermon commit command writes the contents of regdata.cur to regdata.reg, thus updating regdata.reg and committing the Registry data to disk.

Using regsvr to register a shared library in the registry

If you are using a proxy/stub shared library (DLL), you must register the library on both the client machine and the server machine.

On Tru64 UNIX, you use the regsvr utility to register and unregister proxy and stub shared libraries. The utility is based on the Windows NT regsvr32.exe program and accepts the library name as input. The regsvr utility searches the named library for the DllRegisterServer entry point. The utility calls the entry point and registers the library on the host machine.

For example:

```
% regsvr library
```

where library is the library name. If the library is successfully registered, the utility displays the message “ok.”

Chapter 13. Security considerations (authentication)

Implementing COM application security on Tru64 UNIX presents interesting problems, not the least of which is that COM authentication on Tru64 UNIX involves security-specific communication between the Tru64 UNIX machine and an NT machine (called the Primary Domain Controller) to implement NTLM SSPI.

The implementation of security with COM for Tru64 UNIX was not complete with Version 1.0. That is, Version 1.0 did not include authentication; applications had to contain a `CoInitializeSecurity` call with authorization set to `NONE` and impersonation set to `ANONYMOUS`. The most recent version of COM for Tru64 UNIX is security-enabled and supports encryption.

Security

There are several levels of COM application security on Tru64 UNIX:

- Security in the form of call security that allows a client or server application to apply an appropriate security level to method calls.
- Security in the form of the Security Support Provider Interface (SSPI) standard that defines security providers that can be accessible to COM server applications. Microsoft NT uses the Windows NT LanManager Security Support Provider (also called NTLM SSP). COM for Tru64 UNIX supports NTLM SSPI calls for client authentication.
- Security in the form of the security subsystem that authenticates communication between the Tru64 UNIX server (paulad) and the Windows NT Primary Domain Controller (paulas).

The basic steps to establish COM application security on the Tru64 UNIX operating system are:

- Make sure that the security subsystem is enabled; that is, that paulad is active on your server and paulas is installed and active on the Windows NT Primary Domain Controller.

- Make sure that the Domain Controller address is defined in dcomconfig and that a password file and password are shared between the Domain Controller and the UNIX server.
- Make sure that the appropriate level of security is reflected in the application's Registry entry.

Chapter 12 discusses the application's Registry entries. This chapter discusses the security subsystem and Domain Controller password protection.

The security subsystem (paulad and paulas)

Lacking the Windows NT Access Control List and thread-level security, the Tru64 UNIX machine must rely on an associated Windows NT machine to ensure that the client is, indeed, who he says he is. Thus, when a client application calls a server application on Tru64 UNIX, UNIX invokes a system service called paulad that communicates with a service called paulas on an NT machine to authenticate the call. This communication between the Tru64 UNIX server and the NT Primary Domain Controller is password protected.

The Windows NT Primary Domain Controller is the NT server that manages domain accounts for all of the client machines in that domain and, thus, is in a position to authenticate client application calls (where the client can be either an NT machine or a Tru64 UNIX machine).

To give you a sense of how COM application security was implemented between Windows NT clients and Tru64 UNIX servers, consider how the two primary components (paulad and paulas) are installed.

Both paulad and paulas are shipped as part of the COM for Tru64 UNIX software. Once, installed, you would start paulad on the UNIX machine with the dcomsetup menu, or with an ntwoppper command.

As for paulas, it must be copied from the software distribution to the NT Primary Domain Controller and password protection established between paulad and paulas. The paulas service is then started on the NT machine and the Domain Controller address is entered as the value for the COOL_PAULA_DC_ADDRESS variable in the UNIX dcomconfig file. These steps are usually performed by the System Administrator as part of the COM for Tru64 UNIX software installation.

To run applications using the security subsystem, each client and server application user must be registered on the NT Primary Domain Controller. Security is enforced so that each COM user who calls a remote object on the server must have his or her user name and password registered on the NT Primary Domain Controller.

Password protection between paulad and paulas

Secure communication between paulad and paulas is enabled by means of a shared password between the Tru64 UNIX server and the Windows NT Primary Domain Controller. (This password is in addition to the user names and passwords that each COM user must have registered on the NT Primary Domain Controller.)

On Tru64 UNIX, the System Administrator stores the password in the file paula.txt located in an account defined in the COM Configuration file COOL_PAULA_RPC_PASSWORD_FILE environment variable. The UID that owns paula.txt must be the same UID that starts ntd and paulad. The owner should be the only UID with read/write access to the file to protect the integrity of the password.

On the Windows NT Primary Domain Controller, the System Administrator stores the password in %SystemRoot%\System32\paula.txt. To ensure secure communication, the Domain Controller file system must be NTFS and the paula.txt ACL (Access Control List) must allow file access only to the System Administrator.

The password stored in the UNIX and NT paula.txt file must be the same.

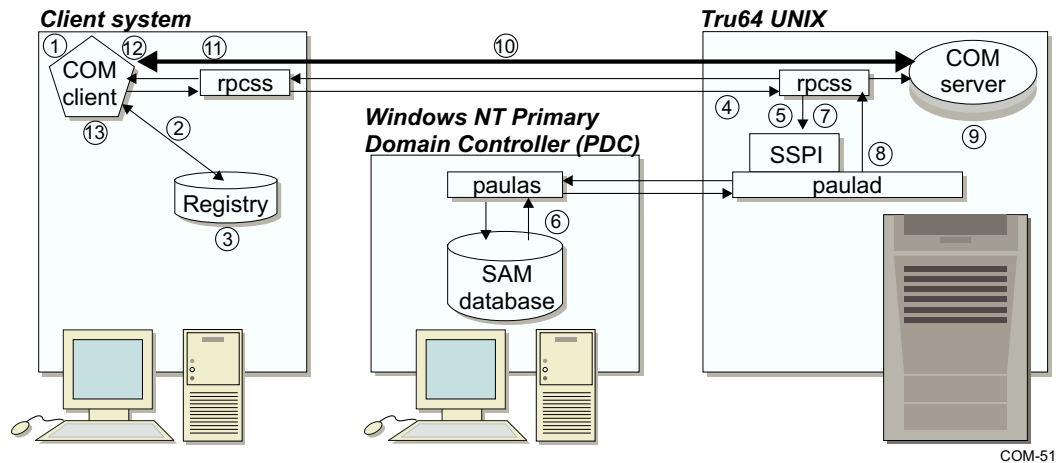
Remote object activation

COM remote activation establishes a secure link between the client application and the remote server application and then steps out of the picture so that authenticated RPC calls can flow directly between the client and the server. To accomplish this purpose, COM for Tru64 UNIX performs a number of steps, accesses information in various databases, and runs through a series of security checks.

The System Administrator configures some aspects of remote server application, such as establishing Domain Controller security. There are also aspects of remote activation that you define on the Tru64 UNIX server, such as synchronizing the COM user names and passwords with those on the NT Primary Domain Controller and updating client and server registries to recognize your application.

Figure 28 and the accompanying text describe the COM remote activation topology on Tru64 UNIX; that is, how calls flow between client and server and how security is imposed on that flow.

Figure 28. Remote activation on Tru64 UNIX



A COM client application on NT makes a request for connection to an object interface on Tru64 UNIX and the following events are set in motion:

1. The COM Run-Time Library (RTL) uses a local Remote Procedure Call (RPC) to forward the request to the client Object Exporter.
2. The client Object Exporter forwards the request to the client Service Control Manager (SCM).
3. The client SCM looks up the object interface ID in the client Registry. If the object is local to the client, the SCM makes the interface available to the application. However, in this case, the SCM determines that the object is on another node.
 - The local and remote Registries are the means whereby SCM finds the server application and associates it with the client application. To allow SCM to make that association, you must register the server application's executable path and the globally unique identifiers (GUIDs) for the server object interfaces with the Registry of each client and server machine.
4. The client SCM, masquerading as the client application, makes an authenticated call to the SCM on the remote node requesting server application activation.
5. From the client side, the client SCM calls the client proxy, which marshals the call arguments and passes them to the client RPC RTL. The client RPC RTL calls the server RPC RTL, which calls the server stub. The server stub unmarshals the argument and passes it to the Object Exporter, which passes it to the server SCM.
 - Remote Procedure Call System Services (RPCSS) includes the SCM, Object Exporter, and the Running Object Table. On the Tru64 UNIX server, the rpcss

daemon, which provides the functions of RPCSS, fields the client call and, with the ntd daemon, processes the call.

- In terms of security, the client RPC call uses authenticated RPC to connect to the server and cause rpcss to launch the server application. On a Windows NT client, NT LanManager security (NTLM) controls this launch by means of the Access Control List in the Registry, which identifies who can start a server.
6. On the Tru64 UNIX server (which does not have Access Control Lists), COM uses a technique called NTLM pass-through to perform the security checks on requests between client and server. The players in this technique are rpcss and paulad on the Tru64 UNIX server, the NTLM server and paulas on the Windows NT Primary Domain Controller, and RPCSS on the NT or Tru64 UNIX client.
- In response to a client call, rpcss on the server calls the Private Authentication LAYER Daemon (paulad) to authenticate the client user name. paulad sends a call to paulas, which searches for the user name in the Security Account Manager (SAM) of the NT Primary Domain Controller.
 - If the user name is found (and the password matches), the client is authenticated. Authentication is sent back to paulad and rpcss on the Tru64 UNIX server.
 - Depending on the type of call, rpcss requests that the NT daemon (ntd) launch the server application or allow the client application to access a running server application.
 - rpcss, using ntd, responds to the call and reverses the communication path to transmit the requested data to the client.
 - The paula.txt files provide password protection across the NTLM pass-through link. On Tru64 UNIX, the System Administrator creates paula.txt in the directory specified in the COOL_PAULA_RPC_PASSWORD_FILE environment variable. The file contains the same password defined in paula.txt on the NT Primary Domain Controller and establishes password protection between paulad and paulas. The paula.txt password must be non-null and non-blank. Also, NTLM pass-through security requires at least one NT Primary Domain Controller in the COM environment that has paulas installed and has all the server user names registered with it. This Domain Controller must be registered in the /etc/dcomconfig file COOL_PAULA_DC_ADDRESS environment variable on the server.
 - If there is no NT Primary Domain Controller on the network, the client can only make non-secure calls. By default, the Tru64 UNIX server is configured for secure calls. If there is no NT Primary Domain Controller on the network, the System Administrator must configure the COM for Tru64 UNIX server with an

authentication level of zero and client and server applications must issue `CoInitializeSecurity` calls that set authentication to `NONE` and impersonation to `ANONYMOUS`.

7. The server SCM looks up the server application in the server Registry. The SCM verifies that the requested object interface resides in the Registry and that the client application is authorized to access the interface. The SCM also checks the Registry's Running Object Table (ROT) to see if the server application is currently active.
8. The `rpcss` daemon fetches the Registry information needed to activate the server application.
9. The server `ntd` (NT daemon) launches the server application on behalf of `rpcss` and `rpcss` communicates object identity information to the client application. Because the server application runs under `ntd`'s UID (the UID under which `ntd` started), the System Administrator must ensure that the UID has execution permissions that allow client requests to launch the application.
10. The `rpcss` daemon sends the application's RPC binding data to the server Object Exporter, obtains an Object Identifier (OXID), returns the OXID to the server SCM, which returns the OXID to the client SCM.
11. The client SCM returns the OXID to the client application.
12. The client application uses COM RTL to send the OXID to the client Object Exporter. The Object Exporter uses the OXID to obtain the server application binding information from the server Object Exporter.
13. The client Object Exporter returns the binding data to the client. The client uses the binding data to fill in the proxy and make authenticated RPC calls directly to the remote server application. At this point, COM steps out of the way.

NTLM pass-through

The previous section describes remote activation with security. A key element of that security is NTLM (or Windows NT LanManager) security, which is provided to compensate for the fact that the Windows NT Primary Domain Controller is not ported to Tru64 UNIX.

This alternative does not affect the coding of COM applications. That is, your application implements NTLM security with COM for UNIX in the same way that it is implemented with COM on Windows NT; the workaround is that UNIX uses a pass-through system to route authentication requests between the UNIX server and the NT Primary Domain Controller.

All of this is illustrated in the steps of **Remote object activation** section and is invisible to the client application.

Setting authentication levels with olecnfg

The olecnfg utility functions are similar to the functions of the GUI tool, DCOMCNFG, found on Windows NT. (In fact, olecnfg is a port of the obsolete predecessor of DCOMCNFG.)

The olecnfg utility command line syntax on Tru64 UNIX is:

```
olecnfg
  [DisplayGlobalSettings]
  [EnabledDCOM <y,n>]
  [DefaultLaunchPermission <y,n>]
  [DefaultAccessPermission <y,n>]
  [LegacyAuthenticationLevel <0,1,2,3,4,5,6>]
  [LegacyImpersonationLevel <1,2,3,4>]
  [[ProgID [Description]] [CLSID [Description]]
    [InprocHandler32 [Path]]
    [InprocServer32 [Path]]
    [LocalServer32 [Path]]
    [LocalService [Path]]
    [RemoteServerName [MachineName]]
    [RunAs [UserName [Password]] ]
    [ActivateAtStorage <y,n>]
    [LaunchPermission <y,n>]
    [AccessPermission <y,n>]
  ]
```

As you look at the settings that you can establish with olecnfg on UNIX, notice the similarities with the functions carried out by DCOMCNFG on NT. For example, both DCOMCNFG and olecnfg:

- Enable DCOM.
- Set default launch and access permissions.
- Set default authentication and impersonation level.
- Enable server specific attributes.

While DCOMCNFG uses a graphical user interface to step you through these functions, the UNIX olecnfg utility uses a command line interface.

olecnfg system-wide registry settings and authentication levels

The olecnfg utility supports six system-wide Registry settings that set default server properties and default server security:

```
olecnfg
    [DisplayGlobalSettings]
    [EnableDCOM <y,n>]
    [DefaultLaunchPermission <y,n>]
    [DefaultAccessPermission <y,n>]
    [LegacyAuthenticationLevel <0,1,2,3,4,5,6>]
    [LegacyImpersonationLevel <1,2,3,4>]
```

- To display the current settings for EnableDCOM, DefaultLaunchPermission, DefaultAccessPermission, LegacyAuthenticationLevel, and LegacyImpersonationLevel, use the olecnfg DisplayGlobalSettings option.
- EnableDCOM — Determines whether any remote client can launch, or connect to, a server application on this machine. By default, distributed COM is enabled when COM is installed on Tru64 UNIX.
- DefaultLaunchPermission — The rpcss daemon uses this setting to determine if the client can launch an application on this machine.
- DefaultAccessPermission — The rpcss daemon uses this setting to determine if the client can access a running application on this machine.
- LegacyAuthenticationLevel — Authentication verifies that the client really is the client and that the call did originate from the client.
- Impersonation, on Windows NT, allows the server application to masquerade as the client. The various levels of impersonation determine the level of authority given to the server as it impersonates the client. Tru64 UNIX does not have security at the process level, which means that impersonation is not supported. COM for Tru64 UNIX does support impersonation levels of ANONYMOUS and IDENTIFY, but does not support IMPERSONATE or DELEGATE. That is, if a client application contains an IMPERSONATE or DELEGATE impersonation level, COM for Tru64 UNIX interprets it as IDENTIFY.

COM for Tru64 UNIX uses underlying RPC functions to provide authentication. RPC allows six levels of authentication in addition to the default level offered by COM for Tru64 UNIX. These are the same authentication levels provided with COM and COM+ security.

The argument values are cumulative. That is, a value includes the authentication levels of the lower numbered values. The setting is stored in the server Registry under the following key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc\SecurityService

Table 8 describes each of the authentication levels.

Table 8. COM application authentication levels on Tru64 UNIX

Value	RPC Function	Meaning
0	RPC_C_AUTHN_LEVEL_DEFAULT	Use the default authentication level offered by the operating system. COM for Tru64 UNIX provides connect authentication by default.
1	RPC_C_AUTHN_LEVEL_NONE	Use no authentication.
2	RPC_C_AUTHN_LEVEL_CONNECT	At the first method call, authenticate the client's credentials at the server. NTLM authenticates credentials with a challenge and response mechanism between the server and the client.
3	RPC_C_AUTHN_LEVEL_CALL	Authenticate the header of the first network packet of each RPC call between the client and server. (This level of authentication protects against call replay, in which intruders capture validated network transmissions and replay them.)
4	RPC_C_AUTHN_LEVEL_PKT	Authenticate the header of each network packet of each RPC call between the client and server.
5	RPC_C_AUTHN_LEVEL_PKT_INTEGRITY	Perform a checksum on the marshalled parameters of the call to ensure that the packet contents have not been tampered with during transmission.
6	RPC_C_AUTHN_LEVEL_PKT_PRIVACY	Not supported.

Table 9 describes each of the impersonation levels.

Table 9. Impersonation levels

Impersonation level	Value and RPC function	Meaning
1	RPC_C_IMP_LEVEL_ANONYMOUS	The client is anonymous.
2	RPC_C_IMP_LEVEL_IDENTIFY	The server can obtain the client's identity, but impersonation is not allowed.
3	RPC_C_IMP_LEVEL_IMPERSONATE	Not supported on Tru64 UNIX. Reverts to IDENTIFY.
4	RPC_C_IMP_LEVEL_DELEGATE	Not supported on Tru64 UNIX. Reverts to IDENTIFY.

Olecnfg application-specific registry settings

The olecnfg utility supports nine system-wide Registry settings that establish Registry settings specific to an application. However, the current implementation of COM on Tru64 UNIX ignores LocalService, RunAs, LaunchPermission, and AccessPermission when the server is Tru64 UNIX.

Olecnfg

```
[[ProgID [Description]] [CLSID [Description]]
    [InprocHandler32 [Path]]
    [InprocServer32 [Path]]
    [LocalServer32 [Path]]
    [LocalService [Path]]
    [RemoteServerName [MachineName]]
    [RunAs [UserName [Password]] ]
    [ActivateAtStorage <y,n>]
    [LaunchPermission <y,n>]
    [AccessPermission <y,n>]
]
```

You would probably establish application-specific settings with a self-registering application, but if you choose to use olecnfg, note the following:

- The CLSID and path identify the application.
- The first three application-specific settings (InprocHandler32, InprocServer32, and LocalServer32) apply to Registry settings on the Tru64 UNIX server. These

applications are on the same machine and are either in-process servers or local servers.

- ◆ InprocHandler32 tells rpcss that the application uses a custom handler and specifies the path to the library.
- ◆ InprocServer32 tells rpcss to start an in-process server application.
- ◆ LocalServer32 tells rpcss to start a local server application (an executable).
- ◆ The fourth setting (LocalService) is not supported on a Tru64 UNIX server.
- The last five application-specific settings (RemoteServerName, RunAs, ActivateAtStorage, LaunchPermission, and AccessPermission) apply to Registry settings on a client communicating with a server. These applications are remote.
 - ◆ RemoteServerName tells the client SCM to request a specific server for remote activation.
 - ◆ ActivateAtStorage tells the client SCM to instantiate the application on the machine where the application was initialized.
 - ◆ The Tru64 UNIX server ignores the RunAs, LaunchPermission, and AccessPermission settings and uses the system-wide default settings.

Chapter 14. Utilities

In addition to the daemons and utilities described earlier (ntd, rpcss, paulad, olecnfg), the COM implementation on Tru64 UNIX also includes:

- stgview and df2t — Utilities you use to view and convert structured storage documents.
- dcomver — A utility that provides you with versioning information that can be used for debugging.

These utilities are similar to those used on Windows NT, but have been slightly modified on Tru64 UNIX.

Structured storage functions, stgview and df2t

Structured storage is a file system within a file that allows COM applications to treat a single file as a structured collection of directories and files that can be shared among processes. These directories and files are called storage objects and stream objects in the COM dialect. Thus, objects running in-process or out-of-process have equal access to data stored on disk.

COM for Tru64 UNIX includes two run-time functions you can use with structured storage. You can use the stgview function as a window into the compound file to verify its structure. You can use the df2t function to convert a directory tree structure into a compound file or convert a compound file into a directory tree structure.

The command line function, stgview, displays the contents of a compound file. Depending on the stgview switch you use, the display can include the names of embedded storage objects and stream objects and information about the compound file itself. The file you examine with stgview must be a structured storage compound file with the extension .dfl.

The df2t function, also called the DocFile to Directory Tree Conversion Program, converts an existing directory tree structure into a compound file or converts the contents of a compound file into a directory tree structure.

A forward conversion converts the structured objects in the compound file to subdirectories in the directory tree and converts the stream objects in the compound file to files in the directory tree. A backward conversion does the reverse (subdirectories convert to structured objects and files convert to stream objects).

As implemented on Tru64 UNIX, you can also use the COM df2t utility to assign an STGM enumeration value to the conversion. That is, you can specify that either STGM-TRANSACTIONED mode or STGM-DENY-WRITE mode be applied to the conversion. Under the COM standard, TRANSACTIONED mode buffers any storage element changes until the client issues an explicit commit. DENY-WRITE mode prevents other users from opening the object in write mode, which protects against copying.

Displaying version numbers with the dcomver utility

The dcomver utility displays the COM for Tru64 UNIX version for various installed COM utilities and shared libraries.

You invoke the dcomver utility from the command line and provide the name of a COM for Tru64 UNIX utility, daemon, or executable.

Because dcomver displays the version number of the installed product, it can be useful in future (post Version 1.0) troubleshooting situations. For example, if you are on the phone to Compaq and the support person asks you for the version number of the recalcitrant COM utility, you can use dcomver to obtain the version and patch level identifier.

Chapter 15. Configuring Tru64 UNIX for COM

The COM for Tru64 UNIX configuration file, `/etc/dcomconfig`, contains the COM environment variables and values that set the conditions for each COM application that starts on the Tru64 UNIX server. These conditions include the level of errors that get recorded, the location of the Registry, and the location of security password files. Utilities, such as `ntd`, use the values defined in `dcomconfig` and the absence of `dcomconfig` or erroneous settings can cause COM applications to fail.

During installation, the COM Setup configuration file establishes initial values for most of the environment variables. However, the System Administrator must add values for `COOL_PAULA_DC_ADDRESS` and `COOL_PRIMARY_DOMAIN`. The System Administrator must also create the `paulad` password file and ensure that the file's contents are the same as the password file on the NT machine.

There probably won't be much reason for you to change the `dcomconfig` file, but if you do, be aware that non-default settings can have unforeseen consequences.

To ensure that all users share the same environment variable settings, `/etc/dcomconfig` is installed under root ownership and permissions are set so that only the System Administrator has write access while all other users have read access. This prevents other users from changing `dcomconfig` to define an application-specific environment.

Table 10 contains a list of the COM configuration variables and the values established by the COM Setup configuration file at installation.

Table 10. COM-related environment variables on Tru64 UNIX

Environment variable	Description	Value set at installation
COOL_NTD_TVTURNER	<p>The value of the ntd tuner used for communication between processes, including the communication that occurs between ntd and rpcss.</p> <p>If you are in a development environment and you find the use of a single ntd process on the system too restrictive, you can specify a different value for this variable, which allows you to run a second ntd under a different UID on the same system. But this can be prone to error and Compaq recommends against it.</p>	=0xc001babe
COOL_VMLOGGING	Enables or disables virtual memory logging. Zero disables logging. One enables logging. A log is useful in locating memory leaks.	=0
COOL_VMLOGFILE	<p>The path and file name of the logfile used to store VM logging output in the form of either stderr (standard error) or stdout (standard output).</p> <p>To log stdout, specify a dash (-) after the equal sign. To log stderr, specify a second equal sign after the first. Any characters after the dash or equal sign are defined as the path and file name.</p>	==
COOL_LOGLEVEL, COOL_LOGFILE	LOGLEVEL is the set of events that COM logs by default. The default is to log errors. The LOGFILE value is the path and name of the file containing LOGLEVEL output.	=2 =
COOL_PERSISTREG	<p>The directory containing the persistent (disk resident) image of the Registry.</p> <p>Be sure that the UID that started ntd can write to this file.</p>	=/etc/com

continued

Table 10 (con't). COM-related environment variables on Tru64 UNIX

Environment variable	Description	Value set at installation
COOL_DFLT_ENVIRONMENT	The directory containing private shared libraries. The run-time loader uses this path when an application executes.	=LD_LIBRARY_PATH: PATH:LOGNAME
COOL_PAULA_DC_ADDRESS	The name or network IP address of the primary Windows NT Primary Domain Controller where the paulas service is running.	=
COOL_PRIMARY_DOMAIN	The name of the Windows NT Primary Domain Controller that paulad uses to authenticate client requests.	=
COOL_PAULA_RPC_PASSWORD_FILE	The Tru64 UNIX path and file name of the password text file that secures communication between paulas and paulad.	=/etc/com/paula.txt
COOL_PAULA_DC_ENDPOINT	The port number for communication with paulas on the Windows NT Primary Domain Controller. By default, port 666 is the endpoint; use of another port can jeopardize interoperability.	=666

Chapter 16. Developing COM applications on Tru64 UNIX

COM application development programmers use COM tools to encapsulate a function into an object and make that object callable by other, client, applications. The server that hosts the object executes the function. Client application calls to the object are actually made to the object's interface. If an object's interface identifier is registered on multiple client machines, multiple client applications can access the function simultaneously.

The COM programmer typically uses the Interface Definition Language (IDL) to create an IDL file that defines the COM object interface (that is, the methods and parameters, interface data types, structure members, and array sizes). The Microsoft IDL compiler (MIDL) reads those definitions and generates code suitable for input to a C or C++ compiler, which generates proxies, stubs, or type libraries.

COM on Tru64 UNIX implements this same basic application development environment. The COM ported to Tru64 UNIX supports client applications written in the standard application languages of C, C++, Java, and Visual Basic. It also supports your use of C and C++ to create COM server applications on Tru64 UNIX.

As described earlier, the basic steps in compiling a COM application on Tru64 UNIX include:

- Defining object interfaces with IDL and creating the registration file for that interface.
- Using MIDL to process the IDL file into C or C++ compatible modules.
- For a proxy/stub library, creating a module definition file with the COM for Tru64 UNIX `makedef` utility.
- Using the C++ compiler to compile and link MIDL output (client and server source files, header files, application configuration file) with required libraries to produce an application (in the form of an executable, shared library [DLL], or type library).
- Registering and running the application.

This chapter describes how these basic steps are implemented on Tru64 UNIX.

Using MIDL to compile an IDL file

Use the MIDL compiler to compile an IDL file on Tru64 UNIX. The MIDL compiler generates C++ compatible source and header files that you then compile with the C++ compiler to produce COM executables and shared libraries.

The Tru64 UNIX platform currently implements a version of the MIDL compiler that corresponds to MIDL Version 3.00.44 on Microsoft Windows NT. In future releases, Compaq will be implementing support for an updated version of MIDL.

Keep the following in mind when using the MIDL compiler on Tru64 UNIX:

- Use the MIDL compiler required switches. Your use of only the required switches allows the compiler to apply the appropriate defaults and produce stubs and proxies suitable for Tru64 UNIX.
- MIDL-generated files are platform specific, which means that you must compile an object's IDL file with MIDL on each platform that hosts that object's interfaces. You cannot copy MIDL output files generated for one platform onto a different platform.
- MIDL commands and switches are case sensitive.
- The `-Os` switch (mixed-mode marshaling) is the only MIDL compiler optimization switch currently supported on Tru64 UNIX.
- By default, the MIDL compiler on Tru64 UNIX uses the C++ preprocessor. Because C++ is binary compatible with COM, Compaq recommends that you do not override the MIDL default. Use of the C preprocessor causes the MIDL compiler to generate code that yields uncertain results on Tru64 UNIX.
- By default, the MIDL compiler on Tru64 UNIX generates input for the C++ compiler that results in MS RPC compatible stubs.
- Use a dash (-) and not a slash (/) to specify a compiler switch.
- If the command line contains an absolute path name, precede the initial slash in the path name with a backslash. For example:

```
midl -I\usr/com/files filename.idl
```

The COM implementation on Tru64 UNIX supports most of the MIDL compiler switches (except for those with only a Windows GUI-oriented purpose). These switches, their arguments, and their purpose are described in the product documentation.

However, producing MIDL output that is suitable for compilation on Tru64 UNIX, does not require a lengthy series of switches to modify the compile. The following list is the minimal set of MIDL switches needed to produce C++ input that results in a Tru64 UNIX COM server application:

- Zp8 — Specifies the packing level of structures in the target system. You must specify the same packing level for the MIDL compiler and the C++ compiler. The default packing level is 8 bytes.
- char unsigned — Sets the default character type to ensure that the MIDL compiler and the C++ compiler operate together correctly for all char and small data types. Use the -char unsigned switch.
- ms_ext — Enables support of Microsoft extensions to OSF DCE such as interface definitions for OLE objects and enum initialization. This is the default mode for the MIDL compiler on Tru64 UNIX.
- c_ext — Allows Microsoft C extensions in the IDL file.
- Os — Specifies mixed-mode marshalling of stub code passed between client and server.

By and large, the MIDL switches used for compilation on Windows NT were ported to Tru64 UNIX and the few differences should not confuse a COM developer working across these two platforms.

Type libraries

Type libraries are binary files describing an object (interface, methods, and type information). Type libraries can be deployed as stand-alone files or referenced in the `importlib` IDL statement of a COM executable, thus making COM object interface and method information available to other COM-enabled client systems.

As with executables and shared libraries, you also use the MIDL compiler on Tru64 UNIX to produce type libraries. As implemented on Tru64 UNIX, the MIDL compiler can parse either ODL or IDL input and generates the type library and an optional C++ header file.

The MIDL compiler generates a type library and an optional C++ header file and uses `ICreateTypeLib` and `ICreateTypeInfo` interfaces to create the type library. The type library can then be accessed by tools and compilers that use those interfaces.

The only operating system dependency on the generation of type libraries is the requirement that the `liboleaut32.so` automation library be installed on Tru64 UNIX.

Compiling a module definition file with `makedef`

The implementation of COM on Tru64 UNIX includes the `makedef` utility, which is a Bourne shell script written for non-Windows COM, that you use to compile a Windows NT-specific Module Definition File (`.def`).

A Module Definition File is a text file containing statements that define exported functions in a proxy or stub shared library. The DEF file specifies the library name that exports function definitions, including the `DllRegisterServer`, `DllUnregisterServer`, `DllCanUnloadNow`, and `DllGetClassObject` functions.

The `DllRegisterServer` and `DllUnregisterServer` functions must be exported to provide self-registration of the proxy or stub library. The `DllGetClassObject` function must be exported to expose the library's class object to COM. COM uses the `DllCanUnloadNow` function to find libraries that it can unload. (Refer to COM programming documentation for information about these functions and the contents and use of a Module Definition File.)

The `makedef` utility generates a file suitable for input to the C++ compiler from Module Definition File input. You compile the C++ file to produce an object (.o) file, which you can then link to a proxy or stub shared library (.so). The object file ensures that the required definitions can be loaded using Windows' `LoadLibrary()` and `GetProcAddress()` functions. You can also use preprocessor statements (for example, `#include`, `#if`, `#ifdef`, `#else`, `#endif`) to do conditional compiles when parsing the DEF file.

For information about the `makedef` utility and the flags and switches you can include in the command line, see the COM for Tru64 UNIX product documentation.

Why use C++?

The implementation of COM on Tru64 UNIX currently requires that you write server applications in C or C++, although it supports client applications in C, C++, Java, and Visual Basic. The restriction on the use of C and C++ for server applications will ease in the future. But even if you had a choice, there are a number of reasons why you might choose to write server and client applications in C++.

C++ is binary compatible with COM. This means that C++ and COM operate in ways that complement each other. For instance, the Interface Description Language (IDL), which COM uses as an interface descriptor for APIs, easily translates to C++. Of more importance, however, is the fact that COM method calls use a `vtbl` structure to index an array of pointers to functions that service the method requests. This `vtbl` structure matches the structure that C++ uses for virtual methods (polymorphism).

Basically, the greater a language's object orientation, the greater its ability to integrate with COM. Binary compatibility with the COM standard provides the structure for interoperability that makes the language in which applications are written and the functions that applications perform, irrelevant to their ability to connect and communicate. This is a very powerful capability of COM and, by extension, of C++.

The Tru64 UNIX C++ compiler (cxx) is ANSI compliant, produces strong typing, and is a superset of the C compiler. Compaq recommends that you use it to compile and link MIDL output to produce COM libraries and executables.

Compaq also recommends that you accept MIDL's default use of the C++ preprocessor. If you override the MIDL default and specify use of the C preprocessor, the generated code yields uncertain results.

Use of C++ gives you access to the benefits provided by C++ compatible tools. For instance, Visual C++ provides the capability to automatically generate registration files.

Also, you can use the Enterprise Toolkit to develop, debug, build, and tune C++ server applications for the Compaq platforms. The Enterprise Toolkit supporting functions include:

- Wizards for setting up remote projects (establishing remote and local file sets and access to a remote host).
- Functions for remotely building applications on Tru64 UNIX.
- Visual Studio search functions.
- The Ladebug Debugger with a graphical interface.
- Access to online documentation.

The current version of COM for Tru64 UNIX requires C++ Version 6.2, which is the default C++ compiler on the most current versions of Tru64 UNIX.

Compiling COM executables and shared libraries

This section and Table 11 list the C++ compiler and linker optional and required switches you use to produce COM proxies and stubs from MIDL-generated source code on Tru64 UNIX. By default, the C++ compiler produces executable proxies and stubs. To specify that the compiler produce shared library proxies and stubs, use the -shared linker switch.

Consider the following:

- You design the COM sever application to execute remotely (that is, the client and server execute in different processes on different machines). If the server application is designed to run as an executable, you compile it with the call-shared switch (the default). If the server application is designed as a DLL, you compile it using the -shared switch and associate it with a surrogate process in the registry.

The COM for Tru64 UNIX implementation of surrogate processes is the same as Microsoft's, with the exception of security.

The COM installation installs the surrogate executable (/usr/bin/dllhost) that you can associate with the class ID of a DLL server application. The surrogate process creates a process within which the DLL executes. The surrogate process also aggregates and exposes that DLL's interfaces to client calls. The surrogate process will support more than one DLL if they have the same ID and the same security requirements. COM for Tru64 UNIX also supports custom surrogate processes.

Because COM for Tru64 UNIX does not support a EOAC_APPID definition in a CoInitializeSecurity call, the surrogate process implements default security as defined in the Registry.

- You design the COM application to execute locally, or out-of-process (that is, the client and server are on the same machine, but execute in different processes). The out-of-process application must be designed to run as an executable and you compile it using the call-shared option (the default).
- You design the COM application to execute as an in-process server (that is, the client and server are on the same machine and execute in the same process). The in-process server application must be designed to run as a library and you compile it using the -shared option.

Table 11. C++ switches for compiling COM applications on Tru64 UNIX

Switch	Meaning
-shared	<p>Linker switch to produce a shared library.</p> <p>The compiler creates all of the tables for run-time linking and resolves references to other specified shared libraries. The library created may be used by the linker to make dynamic executables.</p>
-call_shared	<p>Linker switch to produce an executable file. This is the default.</p> <p>The run-time loader (/sbin/loader) is invoked to bring in all required shareable libraries and to resolve any symbols that remained undefined during static link time.</p>

continued

Table 11 (con't). C++ switches for compiling COM applications on Tru64 UNIX

Switch	Meaning
-depth_ring_search	Linker switch that must be used when generating an executable with the -call_shared switch. The switch causes the linker to use the depth_ring_search method for symbol resolution and avoids a conflict with RPC libraries.
-error_unresolved	Linker switch to produce an error message and cause the link to fail if an unresolved symbol is encountered. This is the default when linking executables.
-g	Linker switch to produce symbol table information for full symbolic debugging.
-inline_manual	Linker switch that specifies inline expansion of function calls that have an explicit inline keyword.
-ms	Compiler switch that allows certain ANSI constructs that Microsoft C++ allows. Ordinarily, the Tru64 UNIX C++ compiler returns a warning on these constructs. This switch is useful for porting Microsoft applications to Tru64 UNIX.
-pthread	Optional compiler and linker switch that directs the linker to use the threadsafe version of any library specified with the -l switch when it links programs. Use this switch only if your COM application makes pthread calls.
-u <i>name</i>	Linker switch that enters <i>name</i> into the symbol table as an undefined symbol. A symbol table is initially empty and this switch forces loading of the first routine.
-unsigned	Linker switch that ensures that all char declarations have the same machine representation and value set as unsigned char declarations.

Required and optional shared libraries

The libraries and the COM run-time environment must be installed on the server and on any client that is running the application. The libraries resolve the API and Win32 subsystem calls in your COM executables and libraries. Each application that executes in the Windows NT client/Tru64 UNIX server COM environment must link to these libraries.

On a Tru64 UNIX server machine, you install the libraries as part of the COM installation. If the client system is a Tru64 UNIX machine, you must install the COM for Tru64 UNIX run-time environment. If the client is an NT machine, it already contains the appropriate libraries.

The linker returns an “undefined symbol” error if it encounters a call to an interface that is not in the COM libraries implemented on Tru64 UNIX. This can occur if the interface is not part of the COM standard. Table 12 lists and describes the libraries.

Table 12. Libraries for COM on Tru64 UNIX

Library name and location	Description	Required or optional
/usr/shlib/libmutant.so	Resolves calls to the Win32 system services and threading functions implemented as part of COM for Tru64 UNIX.	Required.
/usr/shlib/libole32.so	Provides basic COM API support such as structured storage.	Required.
/usr/shlib/liboleaut32.so	Resolves OLE automation run-time calls (such as SysAllocString, VariantInit, and LoadTypLib).	Optional. Required only for type library generation or if your COM application contains automation APIs.
/usr/shlib/librpcrt4.so	Provides the transport layer for Distributed COM.	Required.
/usr/lib/libmutantstubs.a	Static link library that provides the functions libmutant.so requires for Win32 client behavior.	Required.
/usr/shlib/libntrtl.so	Provides NT functions such as process structure, thread scheduling, and interprocess communication. Also acts as the NT Windows executive.	Required.
/usr/shlib/libcoolmisc.so	Provides miscellaneous NT functions.	Required.
/usr/shlib/libcrypt.so /usr/shlib/libbasecrypt.so /usr/shlib/libsecur40.so	Supports security and 40-bit encryption.	Dynamically linked; user does not specify.

Compiler and linker switches and flags

To correctly build a COM application on Tru64 UNIX, there are a number of compiler and linker flags that you must include. Table 13 describes the required compiler flags.

Table 13. Required compiler flags for COM on Tru64 UNIX

Switch or flag	Meaning	Required or Optional
-DSAG_COM=1	Enables platform-specific code sections.	Required.
-DACD_DIGITAL_UNIX	Enables public code specific to Tru64 UNIX.	Required.
-DCE_TAXPOSF1	Enables platform-specific code sections.	Required.
DWIN32=100	MIDL requirement for compiling proxies and stubs.	Required.
-D_WIN32_WINNT=0x400	Specifies Windows NT Version 4.0.	Required.
-DINC_OLE2	Enables OLE automation.	Required.
-DREGISTER_PROXY_DLL	Enables conditional compilation of default definitions for DllMain, DllRegisterServer, and DllUnregisterServer functions.	Required for DLLs.
-DCOBJMACROS	Enables vtable functions implemented in C struct.	Required for DLLs.
-DUNICODE	Enables wchar_t Unicode conformance. If omitted, the compiler assumes received data is ASCII. Microsoft requires that wide character data contain only 2-byte Unicode coded data. Although this is not a Tru64 UNIX requirement, it is a requirement of COM for Tru64 UNIX and ensures that data can be marshalled between objects in a heterogeneous COM environment.	Optional.
-D_WIN32_DCOM	Enables DCOM-related functions such as CoInitializeEx.	Optional.
-D_WCHAR_T_DEFINED	Enables wchar definitions.	Optional.

Part IV: Appendices

This part includes the following:

Appendix A. The COM demo CD-ROM

**Appendix B. Porting 32-bit Windows applications to a 64-bit Tru64
UNIX platform**

Appendix C. Acronyms

Glossary

Index

Appendix A. The COM demo CD-ROM

This appendix describes the contents of the Compaq COM demo CD-ROM. The CD-ROM contains a demo of a COM client and server warehouse application that runs on OpenVMS, Tru64 UNIX, and Windows NT systems.

The CD-ROM includes the following:

- Pre-built executable files for all three platforms.

The pre-built version of the COM demo software consists of the executables and required files that demonstrate COM on OpenVMS and Tru64 UNIX servers with Windows NT clients. This appendix describes how to install and deploy the files on your OpenVMS, Tru64 UNIX, and Windows NT systems.

You need the COM run-time environment installed on your OpenVMS or Tru64 UNIX system to run the pre-built version of the demo. The COM run-time environment ships as part of the OpenVMS and Tru64 operating system. You do not need the COM developer kits for OpenVMS or Tru64 UNIX to run the pre-built demo files.

- Source files for all the demo software.

The source code includes all the sources for the demo. You can use these sources to modify and rebuild the demo or you can use these files as templates when creating your own COM applications. This appendix describes how to build the sources.

The source files are the same for both OpenVMS and Tru64 UNIX. If you understand COM, enjoy reading code, or both, feel free to work with the sources. You must have the COM developer kit installed on your OpenVMS or Tru64 UNIX system before you can build images from the COM demo sources.

- A README file describing how to install and configure the systems.

What does the COM demo do?

The COM demo software is an order entry and fulfillment system that works across a multi-tier client/server environment. The COM demo software simulates a salesperson ordering parts that are held in a remote warehouse.

You can run the demo in the following configurations:

- Windows NT client to OpenVMS server
- Windows NT client to Tru64 UNIX server
- Windows NT client to OpenVMS server and Tru64 UNIX server

From the Windows NT client you can do the following:

- Use an online form to connect to the server.
- Enter one or more orders.
- Fill orders and track inventory in real time.
- View the status of back orders.
- Generate statistics on orders filled vs back orders.

The Compaq OpenVMS or Tru64 UNIX server provides the business logic that drives the order entry system. As you enter the order from the Windows client, the COM server application (called `inventorycontroller`) does the following:

- Receives orders from the client and dispatches orders to the warehouse.
- Maintains statistics on order quantities filled and back ordered.
- Resets statistics on request from the client.

The Compaq OpenVMS or Tru64 UNIX server also provides the inventory application (called `warehouse`), which works in concert with the `inventorycontroller` application. Together, the applications respond to requests from the order-entry server and maintain the inventory.

Installation prerequisites

Your client and server platforms must meet the following hardware and software requirements before you can install and successfully run the COM demo software.

COM for OpenVMS prerequisites

Before you can install and run the COM demo software on OpenVMS, you must do the following:

- Install and configure OpenVMS Alpha Version 7.2-1 or later.
- Install and configure Advanced Server Version 7.2 or later.
- Install and configure the COM for OpenVMS run-time. The COM for OpenVMS run-time is bundled with the OpenVMS Version 7.2-1 operating system. COM for OpenVMS can be installed only on an Alpha system.

For more information about installing COM for OpenVMS, see the *OpenVMS Connectivity Developer Guide*.

Because the COM demo consists of pre-built images, you need only the COM for OpenVMS run-time. If you want to build the demo from the COM demo sources, you must install the COM for OpenVMS developer kit, available under a separate license. For more information on COM for OpenVMS, see the following website:

<http://www.openvms.compaq.com/openvms/products/dcom/>

Configuring OpenVMS security settings

COM Version 1.1 for OpenVMS uses NTLM security and requires specific security settings. Before you can run the COM demo between Windows NT and OpenVMS systems, you must determine the domain and trust relationship between the two systems and configure COM for OpenVMS appropriately.

- If the Windows NT client is in the same domain as the OpenVMS server, or if the systems are in different, but trusted, domains, configure the OpenVMS server with default settings. By default, the server runs in the context of the launching user. When you run the client from a domain account on Windows NT, the server runs on the OpenVMS system using the same client domain account.

For the configuration procedure, see the **In same domain or in different but trusted domains** section.

- If the Windows NT client and the OpenVMS server are not in the same domain, or if you have not established a trust relationship between the domains, you must modify the COM on OpenVMS configuration setup.

For the configuration procedure, see the **In different domains without a trust relationship** section.

In same domain or in different but trusted domains

Use the following procedure on the OpenVMS system:

1. Create an OpenVMS account (for example, DEMO).
2. Create an Advanced Server account (also named DEMO).
3. Map the Advanced Server account to the OpenVMS account using the following command:

```
$ ADMIN add hostmap demo demo
```

4. Log on to the OpenVMS system account and start the DCOM\$SETUP utility as follows:

```
$ @SYS$STARTUP:DCOM$SETUP
```

The system displays the OpenVMS COM Tools menu.

5. From the OpenVMS COM Tools menu, choose **DCOMCNFG**. The system displays the DCOM\$CNFG Main menu.
6. From the DCOM\$CNFG Main menu, choose **Applications List**.
7. Check the applications list for the warehouse server and enter the warehouse server's Index number. The system displays the Application Properties submenu.
8. From the Application Properties submenu, choose **Security**. The system displays the Application Security submenu.
9. From the Application Security submenu, choose **Edit Custom Access permission**. The system displays the Registry Value Permissions submenu.
10. From the Registry Value Permissions submenu, choose **A to Add to List**. The system displays the Add Registry Value Permissions submenu.
11. From the Registry Value Permissions submenu, choose **Add Specific User or Group**. The system prompts you for a user name and password. Enter the user name (for example, DEMO) and password. Toggle the value to **Allow**.
12. From the Application Security submenu, choose **Edit Custom Launch permission**. The system displays the Registry Value Permissions submenu.
13. From the Registry Value Permissions submenu, choose **A to Add to List**. The system displays the Add Registry Value Permissions submenu.
14. From the Registry Value Permissions submenu, choose **Add Specific User or Group**. The system prompts you for a user name and password. Enter the user name (for example, DEMO) and password. Toggle the value to **Allow**.

15. Exit to the Applications List submenu and repeat steps 7 through 14 for the inventorycontroller server.

For more information about adding an Advanced Server account and mapping it to an OpenVMS account see the *OpenVMS Connectivity Developer Guide*.

In different domains without a trust relationship

Use the following procedure:

Systemwide settings

1. Log on to the OpenVMS system account and enter the command:
\$ @SYS\$STARTUP:DCOM\$SETUP
2. From the OpenVMS COM tools menu, choose **DCOMCNFG**. The system displays the DCOM\$CNFG Main menu.
3. From the DCOM\$CNFG Main menu, choose **System-wide Default Properties**. The system displays the System-wide Default Properties submenu.
4. From the System-wide Default Properties submenu, choose **Default Authentication Level**. The system displays the Default Authentication Level submenu.
5. From the Default Authentication Level submenu, choose **None**. The system displays the Default Authentication Level submenu.
6. Log on to the Windows NT system Administrator account.
7. From the **Start** menu, choose **Run...** The system displays the *Run* dialog box.
8. Enter **dcomcnfg** in the command box and click OK. The system displays the *Distributed COM Configuration Properties* dialog box.
9. From the dialog box, click the *Default Properties* tab. From the *Default Authentication Level* list box, choose **(None)**.

Application settings

1. Log on to the OpenVMS system account and enter the command:
\$ @SYS\$STARTUP:DCOM\$SETUP
2. From the OpenVMS COM tools menu, choose **DCOMCNFG**. The system displays the DCOM\$CNFG Main menu.
3. From the DCOM\$CNFG Main menu, choose **Applications List**. The system displays the Applications List submenu.
4. Check the applications list for the warehouse server and enter the warehouse server's Index number. The system displays the Application Properties submenu.

5. From the Application Properties submenu, choose **Identity**. The system displays the Application Identity submenu.
6. From the Application Identity submenu, choose **NTLM Account**. The system prompts you to enter the username and password for the OpenVMS Advanced Server account used to run the server applications. Enter the user name (for example, DEMO) and password.
7. Exit to the Applications List submenu and repeat steps 4 through 6 for the inventorycontroller server.
8. Start the DCOM\$SETUP utility as follows:

```
$ @SYS$STARTUP:DCOM$SETUP
```

The systems displays the OpenVMS COM Tools menu.

9. From the OpenVMS COM Tools menu, choose **DCOMCNFG**. The system displays the DCOM\$CNFG Main menu.
10. From the DCOM\$CNFG Main menu, choose **Applications List**.
11. Check the applications list for the warehouse server and enter the warehouse server's Index number. The system displays the Application Properties submenu.
12. From the Application Properties submenu, choose **Security**. The system displays the Application Security submenu.
13. From the Application Security submenu, choose **Edit Custom Access permission**. The system displays the Registry Value Permissions submenu.
14. From the Registry Value Permissions submenu, choose **A to Add to List**. The system displays the Add Registry Value Permissions submenu.
15. From the Registry Value Permissions submenu, choose **Add Everyone**. Toggle the value to **Allow**.
16. From the Application Security submenu, choose **Edit Custom Launch permission**. The system displays the Registry Value Permissions submenu.
17. From the Registry Value Permissions submenu, choose **A to Add to List**. The system displays the Add Registry Value Permissions submenu.
18. From the Registry Value Permissions submenu, choose **Add Everyone**. Toggle the value to **Allow**. Enter **E** until you return to the Applications List submenu.
19. Exit to the Applications List submenu and repeat steps 11 through 18 for the inventorycontroller server.

For more information about adding an Advanced Server account and mapping it to an OpenVMS account, see the *OpenVMS Connectivity Developer Guide*.

Running the COM servers on OpenVMS

Use the following procedure to run the warehouse and inventorycontroller servers.

On the Windows NT system:

- Log into the DEMO account on Windows NT to run the client.

On the OpenVMS system:

1. To run the warehouse server, log into the DEMO account on OpenVMS and obtain NT credentials for the account using the following command:

```
$ MCR NTA$LOGON demo
```
2. Set default to the [.warehouse] subdirectory and enter the following command:

```
$ run warehouse
```
3. To run the inventorycontroller server, log into the DEMO account on OpenVMS and obtain NT credentials for the account using the following command:

```
$ MCR NTA$LOGON demo
```
4. Set default to the [.invctr] subdirectory and enter the following command:

```
$ run inventorycontroller
```

COM for Tru64 UNIX prerequisites

Before you can install and run the COM demo software on Tru64 UNIX, you must install and configure the Compaq COM for Tru64 UNIX software on a Version 4.0D or later Tru64 UNIX operating system and initialize COM services.

COM for Tru64 UNIX is available on the Tru64 UNIX Version 5.0 Associated Products CD or as a free download from the COM for Tru64 UNIX website (<http://www.tru64unix.compaq.com/com/>). You can download a kit from the website and use the instructions in the COM for Tru64 UNIX Installation Guide to install, configure, and initialize the software.

Because the COM demo consists of pre-built images, only the COM for Tru64 UNIX run-time is required. To modify and build the COM demo sources, you must install the COM for Tru64 UNIX development environment.

Windows NT client prerequisites

To run the COM demo software on the Windows NT client, you must have a Microsoft Windows NT 4.0 with Service Pack 3 or later installed. If you want to run the Visual Basic statistics application, you must have Microsoft Excel installed.

You can connect the Windows NT system to a network with one or more Compaq Tru64 UNIX Version 4.0D (or later) Alpha systems or Compaq OpenVMS Alpha Version 7.2-1 (or above) systems, or both. You must have at least one Alpha server.

The network connection requires that you configure TCP/IP on both the client and server systems.

Installing the COM demo software

The COM demo software is on the CD-ROM included with this book. To install the COM demo software, use the following procedure:

1. Install COM on the OpenVMS or Tru64 UNIX server and initialize COM services.
2. Load the COM demo CD-ROM on the Windows NT system and run **setup.exe**. The setup program launches Installshield, which installs the client files and registers the client applications. Installshield also copies the OpenVMS and Tru64 UNIX server files to the Windows NT system in compressed form.
3. Use binary ftp to transfer the OpenVMS and Tru64 UNIX compressed server files from the Windows NT system to the corresponding server system.
4. Expand the server files and run an OpenVMS command file or Tru64 UNIX script to install and register the server applications. Use one of the following:
 - On OpenVMS, run the OVMSDEMOKIT.EXE file to expand the server files. Run the VMS_REGISTER_DEMO.COM file to register the server applications.
 - On Tru64 UNIX, untar tru64dcomdemo.tar.gz to expand the server files and run the /src/dunix/bin/installdemo script to install and register the server applications.
5. Run the COM demo software from the Windows NT system.

The following sections contain details on each of these steps.

Windows NT installation

You must have administrator privileges on the Windows NT system to install the COM demo software on Windows NT. Use the following procedure:

1. Load the CD-ROM on your Windows NT system.

2. Run the setup.exe program.

- From the **Start** menu choose **Run...** and locate the setup.exe file on the CD-ROM.
- From Windows Explorer, display the contents of the CD-ROM and double-click setup.exe.

The setup.exe program starts Installshield, which installs the COM demo files.

The system displays the Installshield Setup dialog box.

3. Accept all of the defaults. By default, Installshield installs the files in \Program Files\Compaq DCOM Demo on the C: drive, adds the Compaq DCOM Demo to the **Start** menu, and registers the Windows NT client application.

Installation copies the following files:

- The executables and libraries needed to run the COM demo software on Windows NT and Compaq OpenVMS and Tru64 UNIX systems.
- The compressed files containing the executables, command files, and scripts required for installing the COM demo software on OpenVMS or Tru64 UNIX.
- The Visual Basic client components for generating statistical spreadsheets.

For a complete list of installed files, see Table 15, Table 16, and Table 17.

When the installation completes, use an ftp binary transfer to move the appropriate compressed server files from the Windows NT system \Program Files\Compaq DCOM Demo\Remote directory to the OpenVMS or Tru64 UNIX system.

OpenVMS installation

Before installing the COM demo software on an OpenVMS server, you must do the following:

- Install COM for OpenVMS.
- Start COM for OpenVMS.
- Create a directory on the OpenVMS system for the COM demo files.

To install the COM demo software on OpenVMS do the following:

1. From the Windows NT system, use binary ftp to transfer the OVMSDEMOKIT.EXE file to the OpenVMS [.demo] directory.
2. On OpenVMS, set your default directory to the [.demo] directory and run the OVMSDEMOKIT.EXE file. The OVMSDEMOKIT.EXE file is a self-extracting file

that unpacks the COM demo software. For a list and description of the installed directories and files, see Table 16.

3. To register the inventorycontroller and warehouse server applications, run the VMS_REGISTER_DEMO.COM file. This command procedure registers the COM demo server applications in the OpenVMS Registry.

You are now ready to run the demonstration. See the **Running the COM demo software** section.

Tru64 UNIX installation

Before installing the COM demo software on a Tru64 UNIX server, you must have COM for Tru64 UNIX installed and running on the server.

Among the COM demo files installed on the NT machine is a UNIX zipped tar file, tru64dcomdemo.tar.gz, that contains the COM demo software binaries for the server platform. Using binary ftp, transfer tru64dcomdemo.tar.gz to a directory on the Tru64 UNIX machine and use the gunzip and tar commands to unpack and expand the file.

For example:

```
# gunzip tru64dcomdemo.tar.gz
# tar xvf tru64dcomdemo.tar
```

The expansion of the tar file creates a /CpqDcomDemo directory that contains all the required files for running the COM demo in a Tru64 UNIX/Windows NT environment. Table 17 describes the installed directories and files.

Execute the /src/dunix/bin/installdemo script from a non-root account to install the COM demo server applications (inventorycontroller and warehouse) in the appropriate directories and to register the COM demo server applications in the Tru64 UNIX COM Registry.

The server applications include CoInitializeSecurity calls in which RPC_C_AUTHN_LEVEL is set to NONE and RPC_C_IMP_LEVEL is set to ANONYMOUS. These settings for authentication and impersonation disable security and allow the COM demo software to run between server and client with no user authentication and with no Primary Domain Controller.

With the COM demo software successfully installed on the Windows client and the Tru64 UNIX server, you can run the demonstration as described in the **Running the COM demo software** section.

Configuring the COM demo software

If you have followed the installation instructions, you should have already configured the OpenVMS and Tru64 UNIX servers. The servers are configured if you:

- Successfully installed and configured the COM run-time on OpenVMS or on Tru64 UNIX (or both). These steps are described in each operating system *Installation Guide*.
- Successfully installed and registered the COM demo software inventorycontroller and warehouse server applications, as described in the **OpenVMS installation** and **Tru64 UNIX installation** sections.

You are now ready to run the COM demo software.

Running the COM demo software

Before you start the COM demo, you must enable Distributed COM and disable security on the Windows NT client. Use the following procedure:

1. From the **Start** menu, choose **Run...**. The system displays the *Run* dialog box.
2. Enter **dcomcnfg** in the command box and click OK. The system displays the *Distributed COM Configuration Properties* dialog box.
3. From the dialog box, click the *Default Properties* tab. Set the following:
 - Check the Enable Distributed COM on this computer box.
 - From the *Default Authentication Level* list box, choose **(None)**.
 - From the *Default Impersonation Level* list box, choose **Anonymous**.

If you want to plot statistics from the demonstration, you must install Excel. If Excel is not installed when you request statistics, Windows NT displays the following error:

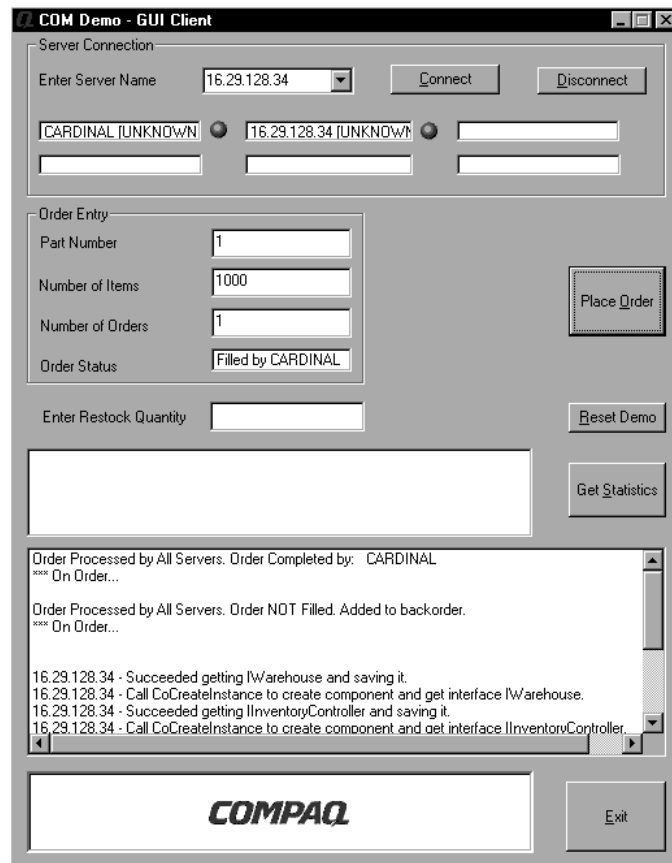
429 ActiveX Component Can't Create Object

Starting the demo

Run the demo from the Windows NT system. The Windows NT system automatically launches the server applications on the remote OpenVMS and Tru64 UNIX systems. With security disabled and Excel running, use the following steps to run the COM demo software and activate the COM server application remotely:

1. From the **Start** menu, choose **DcomClient**. The system displays the DCOM Demo menu shown in Figure 29.

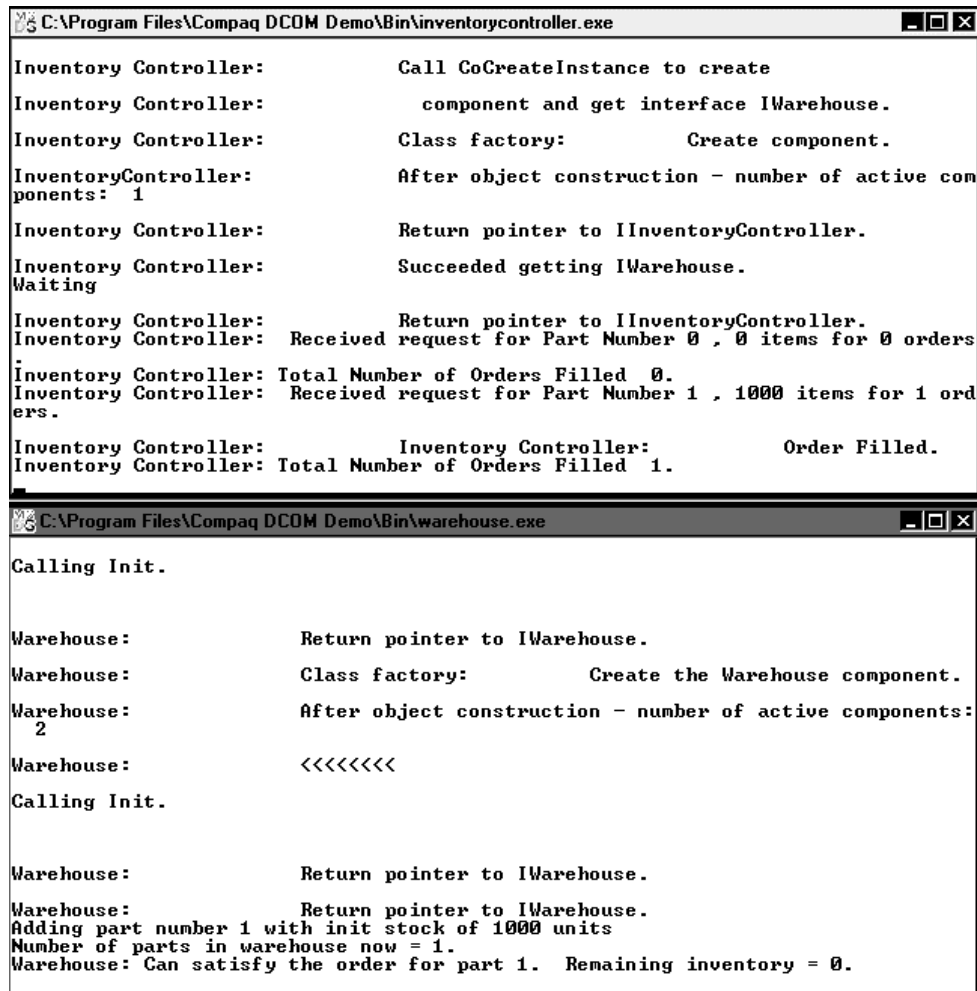
Figure 29. COM Demo - GUI client menu



2. In the *Enter Server Name* box, enter the OpenVMS or Tru64 UNIX system name or the system's IP address. Click **Connect**.

The demo launches two DOS windows on the Windows NT system. These windows display the client and server interactions. Figure 30 shows the DOS windows that display the interactions between the client and the server.

Figure 30. DOS windows showing server interaction



The image shows two DOS windows. The top window is titled 'C:\Program Files\Compaq DCOM Demo\Bin\inventorycontroller.exe' and the bottom window is titled 'C:\Program Files\Compaq DCOM Demo\Bin\warehouse.exe'. The top window displays a series of messages from the Inventory Controller, including calls to CoCreateInstance, returning pointers to IInventoryController, and receiving requests for parts. The bottom window displays messages from the Warehouse, including calls to Init, returning pointers to IWarehouse, and processing orders, including adding stock and satisfying orders.

```

C:\Program Files\Compaq DCOM Demo\Bin\inventorycontroller.exe
Inventory Controller:      Call CoCreateInstance to create
Inventory Controller:      component and get interface IWarehouse.
Inventory Controller:      Class factory:      Create component.
InventoryController:      After object construction - number of active com
ponents: 1
Inventory Controller:      Return pointer to IInventoryController.
Inventory Controller:      Succeeded getting IWarehouse.
Waiting
Inventory Controller:      Return pointer to IInventoryController.
Inventory Controller:      Received request for Part Number 0 , 0 items for 0 orders
Inventory Controller:      Total Number of Orders Filled 0.
Inventory Controller:      Received request for Part Number 1 , 1000 items for 1 ord
ers.
Inventory Controller:      Inventory Controller:      Order Filled.
Inventory Controller:      Total Number of Orders Filled 1.

C:\Program Files\Compaq DCOM Demo\Bin\warehouse.exe
Calling Init.

Warehouse:      Return pointer to IWarehouse.
Warehouse:      Class factory:      Create the Warehouse component.
Warehouse:      After object construction - number of active components:
2
Warehouse:      <<<<<<<
Calling Init.

Warehouse:      Return pointer to IWarehouse.
Warehouse:      Return pointer to IWarehouse.
Adding part number 1 with init stock of 1000 units
Number of parts in warehouse now = 1.
Warehouse: Can satisfy the order for part 1. Remaining inventory = 0.

```

The demo then displays an order menu.

Note:

The state of the network affects the speed with which the system displays the order menu. You might see a delay in the order menu display.

To connect to multiple servers, repeat Step 2 for each server. When you use multiple servers, the system forwards order requests to the next server when the previous warehouse server is empty. The system continues to forward order requests until all specified warehouse servers are empty.

Note:

If you leave the Server Name box blank and click **Connect**, the COM demo connects you to the local server (where the Windows NT versions of inventorycontroller and warehouse are installed).

By default, the demo initializes each part number in the inventorycontroller for an order quantity of 1000.

Running the demo

The following sections describe how to demonstrate the COM demo.

Ordering items

Do the following:

1. In the *Part Number* box, enter 1.
2. In the *Number of items* box, enter 1000.
3. In the *Number of orders* box, enter 1.
4. Click **Place Order**.

The COM demo menu displays the *Order Status* box as **Filled by** [server name].

Ordering items and exhausting the warehouse

Do the following:

1. In the *Part Number* box, enter 1.
2. In the *Number of items* box, enter 1000.
3. In the *Number of orders* box, enter 1.
4. Click **Place Order**.

Because the initial inventorycontroller server order quantity of 1000 was exhausted by the first order, the COM demo menu displays **Order NOT Filled**.

Click the *Get Statistics* button, which generates statistics on the previous orders. The statistics show two orders, with one order filled and one backorder.

Restocking the warehouse and reordering

Do the following:

1. In the *Restock Quantity* box, enter 1.
2. Click **Reset Demo**.

3. In the *Part Number* box, enter 1.
4. In the *Number of items* box, enter 1.
5. In the *Number of orders* box, enter 1.
6. Click **Place Order**.

Because the inventorycontroller server stock for Part Number 1 has been increased by one, the COM demo menu displays the *Order Status* box as **Filled by** [*server name*].

7. In the *Part Number* box, enter 1.
8. In the *Number of items* box, enter 1.
9. In the *Number of orders* box, enter 1.
10. Click **Place Order**.

Because the restocked inventory of one has been exhausted, the COM demo menu displays **Order NOT Filled**.

11. Click the *Get Statistics* button, which generates statistics on the previous two orders. The statistics show two orders, with one order filled and one backorder.

Plotting statistics

In addition to the statistics collected and displayed by the COM demo software, you can run a plotting package that generates a bar graph based on transactions with the warehouse.

Do the following:

1. From the **Start** menu, choose **DcomStats**. The system displays the DCOM Stats menu.
2. Enter a server name or IP address. Click **Add**. The server name appears in the display statistics field.
3. Click **Display Statistics**, which connects the client to the inventorycontroller server and generates a bar chart.

Exiting the demo

Do the following:

- Click **Exit**.

Note:

The servers do not include the ability to store transactions from one session to another. All transactions are lost if you exit from the COM demo.

Building the COM demo software sources

The CD-ROM included with this book contains source files for the Windows NT client and server applications. The source files are contained in the compressed files for each platform.

When you run setup.exe to install the COM demo, InstallShield installs the NT sources in the \DcomClient, \invctr, \VBClient, and \warehouse subdirectories of the \Compaq DCOM Demo\Src\ directory.

When you expand the compressed OpenVMS or Tru64 UNIX files on the appropriate Alpha platform, the source files are written to the /cpqdcomdemo/src/invctr and /warehouse subdirectories on Tru64 UNIX or the [.invctr] and [.warehouse] subdirectories on OpenVMS.

You can use these source files as templates for your own applications or you can modify them to suit your application needs. The following sections describe how to build the sources on their respective platform.

Building sources on Windows NT

The Windows NT client sources include Visual C++ sources for the client application and Visual Basic sources for the statistical display.

To build the client sources on Windows NT, you must have Visual Studio and Visual Basic installed and you must be able to run them from the DOS command line.

To rebuild the modified NT client sources, run DEMOBUILD.BAT from the source directory.

Building sources on Tru64 UNIX

To rebuild modified Tru64 UNIX COM sources, run the make files from the /dunix directory:

```
# cd /dunix
# make clean
# make
```

After the binaries are built, you must register the order-entry and inventory database servers in the Tru64 UNIX Registry. The servers are self-registering.

To activate self registration of the inventorycontroller and warehouse server application, execute the installdemo script:

```
# /dunix/bin/installdemo
```

Finally, to register the appropriate type libraries on the Tru64 UNIX server, you must run the inventorycontroller and warehouse server applications from their respective server directories. For example:

```
# cd /dunix/warehouse/
# warehouse &
warehouseclient
2
3
```

and:

```
# cd /dunix/invctr/
# inventorycontroller &
inventorycontrollerclient
2
3
```

Building sources on OpenVMS

To rebuild modified OpenVMS COM sources, run the VMS_DEMOBUILD command file as follows:

```
$ @VMS_DEMOBUILD
```

The command file builds and registers the OpenVMS inventorycontroller and warehouse servers.

Possible error messages

Error messages can occur when running the COM demo software. Table 14 lists the most common errors, their meaning, and areas to troubleshoot. For more complete troubleshooting information, see the COM product documentation for your server.

Table 14. Possible COM demo errors

Error	Meaning	Solution
80020005	Type mismatch	The application uses a data type that causes a data type mismatch between the 32-bit NT client and the 64-bit Alpha server.
80029C4A	Error loading type library/DLL	Indicates Registry corruption. Initialize the server registry and re-register applications.
80040002	No such interface supported	Generated by a CoCreateInstanceEx API when COM cannot find a specified GUID in the Registry. Re-register the server applications.
80040154	Class not registered	Represents Registry issues and generated when COM cannot find a CLSID, DLL, or other application information. Re-register the application.
80040155	Interface not registered	
80070005	Access denied	Usually a security issue.
800706BA	RPC server is unavailable	NT cannot communicate with the server. This could be a problem with an inactive run-time, problems with the network, or an attempt to run the server application in debug mode.
800706B6	No bindings	Usually a problem with TCP/IP and the network.
80080005	Server execution failed	The server is not running. Restart the server. This error also occurs when the client calls a machine where no interactive user is logged in and active. There must be at least one interactive user active on the target machine.

COM demo installed files

The following tables list and describe the files installed on each platform.

Table 15. Files installed on Windows NT system in C:\ProgramFiles\Compaq DCOM Demo

Path and file name	Use
Uninstall.isu	Removes the COM demo software from Windows NT.
\Bin\DcomClient.exe	Distributed COM Client executable.
\Bin\DcomStats.exe	Statistics charting executable.
\Bin\inventorycontroller.exe	Inventory controller server application.
\Bin\invctr.tlb	Inventory controller application type library.
\Bin\warehouse.exe	Warehouse server application.
\Bin\warehouse.tlb	Warehouse application type library.
\Bin\stats.exe	Statistics executable.
\Doc\Readme.*	A text and HTML version of this document.
\Remote\ovmsdemokit.exe	OpenVMS COM demo kit that you ftp to the OpenVMS server.
\Remote\tru64comdemo.tar.gz	Tru64 UNIX Demo kit that you ftp to the Tru64 UNIX server.
\Src\demobuild	Script for building the COM demo.
\Src\ntinstall	Script for installing the COM demo (automatically performed by InstallShield).
\Src\DcomClient*	Source files and graphics files for modifying and rebuilding the COM demo client.
\Src\invctr*	Source files for modifying and rebuilding the inventorycontroller server application.
\Src\VBClient*	Source files for modifying and rebuilding the statistics package.
\Src\warehouse*	Source files for modifying and rebuilding the warehouse server application.

Table 16. Files installed on the OpenVMS system in the [.demo] directory

Path and file name	Use
VMS_DEMOBUILD.COM	Command file for building the server applications.
VMS_REGISTER_DEMO.COM	Command file for registering the server applications.
[.INVCTR]BUILD_INVENTORYCONTROLLER.COM	Command file for rebuilding modified inventorycontroller sources.
[.INVCTR]INVENTORYCONTROLLER.*	Executable, map, and object files of inventorycontroller server application.
[.INVCTR]INVENTORYCONTROLLERCLIENT.*	Executable, map, and object files of inventorycontroller client application.
[.INVCTR]*DBG.*	Executable, map, and object files of the debug versions of inventorycontroller client and server application.
[.INVCTR]*.H, *.IDL, *.TLB, *.\$SHR	Source files, type libraries, and shared libraries for inventorycontroller.
[.INVCTR]NTCLIENT.*	Source files, executables, map and object files for the NT Client application.
[.INVCTR]DINVENTORYCONTROLLERCLIENT.*	Version of inventorycontroller using OLE automation.
[.WAREHOUSE]BUILD_WAREHOUSE.COM	Command file for rebuilding modified warehouse sources.
[.WAREHOUSE]WAREHOUSE.*	Executable, map, and object files of warehouse server application.
[.WAREHOUSE]WAREHOUSECLIENT.*	Executable, map, and object files of warehouse client application.
[.WAREHOUSE]*DBG.*	Executable, map, and object files of the debug versions of warehouse client and server application.
[.WAREHOUSE]*.H, *.IDL, *.TLB, *.\$SHR	Source files, type libraries, and shared libraries for warehouse.

Table 17. Files installed on the Tru64 UNIX system in the /CpqDcomDemo/src/ directory

Path and file name	Use
/dunix/bin/installdemo	The COM demo software script that registers the Demo server application in the COM Registry.
/dunix/bin/uninstalldemo	The COM demo software script that removes the server application entries from the COM Registry.
/dunix/bin/inventorycontroller	Client and server versions of inventorycontroller and the inventorycontroller type library.
/dunix/bin/warehouse	Client and server versions of warehouse and the warehouse type library.
/dunix/shlib/*	The directory that contains server application shared libraries. This directory is initially empty. It is populated when the makefiles are executed.
/dunix/warehouse/*	Makefiles for building the warehouse server application.
/dunix/invctr/*	Makefiles for building the inventorycontroller server application.
/invctr/*	The inventorycontroller server application C++ and .h source files that you use to modify the inventorycontroller application. These files are in NT format.
/warehouse/*	The warehouse server application C++ and .h source files that you use to modify the warehouse application. These files are in NT format.

Appendix B. Porting 32-bit Windows applications to a 64-bit Tru64 UNIX platform

This appendix discusses platform differences and the lessons that Compaq learned when it ported COM to the 64-bit Alpha, Tru64 UNIX platform. Of course, platform and operating system differences abound from one environment to another, but you may find the following information useful if you are porting applications from one environment to another. At a minimum, it may give you some food for thought as you try to debug that port.

Data type macros

The compiler preprocessors on Tru64 UNIX and Windows NT expand data type macros differently. If you are porting a Windows NT application to Tru64 UNIX, you must be sure that your code is consistent with the Tru64 UNIX data types. Table 18 describes the platform-specific data type and size for each macro.

Table 18. Data type macro differences between Windows NT and Tru64 UNIX

Data type macro	Windows NT data type	Windows NT size in bytes	Tru64 UNIX data type	Tru64 UNIX size in bytes
CHAR	Char	1	char	1
WCHAR	Wchar_t	2	wchar_t	4
SHORT	Short	2	short	2
LONG	Long	4	int	4
ULONG	Int	4	unsigned int	4
LONGLONG	Hyper	8	_int64	8

continued

Table 18 (con't). Data type macro differences between Windows NT and Tru64 UNIX

Data type macro	Windows NT data type	Windows NT size in bytes	Tru64 UNIX data type	Tru64 UNIX size in bytes
FLOAT	Float	4	float	4
DOUBLE	Double	8	double	8
DWORD	Int	4	int	4

The MIDL compiler on Tru64 UNIX translates data types in conformance with the Network Data Representation (NDR) specification, which is consistent with the Windows NT data types. If your application's IDL file contains 64-bit long data types (the Tru64 UNIX default), the MIDL compiler generates a 32-bit LONG data type that conforms to the NDR specification. To ensure that your application generates a 64-bit long data type, use `hyper` or `LONGLONG` as the 64-bit data type in your IDL file.

Marshalling pointers in embedded structures

To accommodate the requirement that the length of a pointer passed between client and server be 32 bits, the Tru64 UNIX platform modified MS RPC to adjust the relative offsets used to parse a complex structure (a structure with more than one atomic element) that contains one or more pointers.

In marshaling data for inclusion in an outgoing packet, MS RPC adjusts the offsets to be consistent with 32-bit pointers. In unmarshalling an incoming packet, MS RPC adjusts the offsets to be consistent with 64-bit pointers.

OXID key size

On the Tru64 UNIX platform, `rpcss` stores the OXID key as a 64-bit pointer. However, COM client and server applications require a 32-bit OXID for compatibility with the Microsoft Component Object Model Specification. The Tru64 UNIX platform added a method `CBlist::Lookup` that retrieves the 64-bit OXID, truncates it to 32 bits, and returns the truncated OXID to the caller.

The following code fragment uses `Lookup` to retrieve an OXID and truncate it to 32 bits as a `gpProcessList` object:

```
pProcess = (CProcess *)gpProcessList->Lookup(key);
```

Disabling UNIX permissions mask

When a client process is using a COM object created in a server process on the same system, they communicate by means of IPC (InterProcess Communication). In the case of Tru64 UNIX, the server process (an instance of the `NtcProcess` class) creates a file that is used by both the client and server processes. However, the client process may not be able to read or write the file if the UNIX umask variable defined for the user account that owns the server process alters the permissions of this file.

To ensure that a umask variable has no adverse effect on the client's ability to read or write the file, the Tru64 UNIX platform added a new member to the `NtcProcess` class that resets the umask value for an instance of `NtcProcess` when the process is constructed and restores the value of umask when the process is destroyed.

Wide character encoding and Unicode

The Unicode and ISO/IEC 10646 standards specify the Universal Character Set (UCS), a character set that allows character units to be processed for all languages with the same set of rules.

The COM sources on the Tru64 UNIX platform support the UCS-4 encoding of this character set, which means that the implementation parses characters in 32-bit units. Windows NT supports the UCS-2 encoding, which means that it parses characters in 16-bit units.

Wide character (`wchar_t`) encoding, which allocates a fixed number of bytes for each character, is typically used for passing Unicode-conformant data. However, the size of `wchar_t` is platform dependent. For example, `wchar_t` is a 16-bit value on Windows NT and a 32-bit value on the Tru64 UNIX platform. An application can use `sizeof(wchar_t)` to determine the size for the platform on which it is running.

Marshalling `wchar_t` data

When an application marshals data of type `wchar_t`, the Tru64 UNIX platform truncates the data to the lower 16 bits. Any data the application attempts to pass in the upper 16 bits is lost. Unicode data is not lost because currently, all Unicode data occupies only the lower 16 bits of a wide character.

Building a UUID (GUID)

The `GetNodeIdFromEthers()` routine, which is used to get a node ID to include in a UUID, was changed to be consistent with the Tru64 UNIX platform implementation of sockets. After opening a socket, the new code:

- Calls `ioctl`, using the request option `SIOCGIFCONF`, to get the configuration list for the interface.
- Calls `ioctl`, using the request option `SIOCGIFFLAGS`, to get the interface flags for each device.
- Uses the interface flags to determine if a device in the configuration list is a network device. If it is a network device, the code calls `ioctl`, using the request option `SIOCRPHYSADDR`, to retrieve the physical address of the device.

inet_addr return value

The function `inet_addr` translates an Internet network address string to an Internet address integer. On the Tru64 UNIX platform, `inet_addr` returns a 32-bit `in_addr_t` value. COM casts this value as `ULONG` to accommodate existing MS RPC code.

Appendix C. Acronyms

ACME	Authentication and Credential Management Extension
API	Application Program Interface
CLSID	Class ID
COM	Component Object Model
DCOM	Distributed Component Object Model
DLL	Dynamic Link Library
FMS	Forms Management System
GUI	Graphical User Interface
GUID	Globally Unique Identifier
MIDL	Microsoft Interface Definition Language
O-O	Object oriented
RPC	Remote Procedure Call
SMG	Screen Management Facility
SSPI	Security Support Provider Interface
UI	User Interface

Glossary

activation

The process of loading an object into memory. In COM terminology, this process is also called binding. The process resolves a client's call to execute a COM object by locating that object (application), putting it into a running state, and returning the object's interface pointer to the client. See also *remote activation*.

automation

The creation of an application that exposes its objects to other, outside, applications. Under the concept of automation, an application makes its services programmable. That is, the application exposes its objects by means of interfaces to clients written in simple languages such as Visual Basic.

class

The definition of an object in code. In C++, for example, the class of an object is defined as a data type.

class (registry class)

Registry element attribute that allows you to store additional descriptive information with a registry key or subkey.

class factory

COM terminology for a class object. An object whose state is shared by all the other objects in that class. In COM, a class factory (the *IClassFactory* interface) creates new instances of the object identified by a given CLSID.

class ID (CLSID)

A globally unique identifier (GUID) that is associated with a class object definition. Clients use the CLSID you register in the system registry to locate and load the executable code associated with the object. In effect, the CLSID identifies the COM server application so that it can be accessed and launched by clients.

client application

A COM object that requests services from another object. Contrast with **server application**.

encapsulation

The process of updating or extending the life of existing application code by leaving most of the code and its functionality intact, while including new or updated code (usually in a different programming language) at key entry points.

For example, you might add a Windows graphical interface to a character-cell application by writing some Visual Basic code that collects information from a Windows client, then formats and submits the data to the existing character cell application as if the data had come from the character cell interface.

endpoint map

A list of network protocols, network addresses, and end points (for example, socket numbers) that each server listens to.

executable

A server application that runs outside the process of the client that calls it. The application can run on the same machine as the client (local) or on another machine in the network (remote). Also known as an out-of-process server.

GUID

Globally unique identifier. An identifier that is associated with and unique to an interface. The GUID is used in the registry to provide a client application with a method for accessing the server object.

hive

A discrete set of keys, subkeys, and value entries contained in the registry.

IDL

Interface Definition Language. The language used to completely and precisely specify COM objects.

in-process server

An application that is located on the same system as the requesting client. On Windows NT systems, in-process servers are usually implemented as dynamic link libraries (DLLs). On OpenVMS systems, in-process servers are usually implemented as shareable images. On Tru64 UNIX systems, DLLs are implemented as shared libraries.

key (registry key)

Registry element that contains information specific to the computer, system, or user.

local server

An out-of-process server, implemented as an executable, that runs on the same machine as the client that calls it. See also *executable*.

marshalling

The packing, sending, and unpacking of interface methods across process boundaries. Specifically, the conversion of a request or response into a sequence of bytes.

moniker

A name that uniquely identifies a COM object. There are a number of different types, or classes, of monikers including absolute monikers (specify the absolute location of an object), file monikers, and item monikers. COM also supports a moniker provider, which is an application that manages objects and makes monikers available for those objects.

object

In COM, the encapsulation of data and function, defined as a single unit and accessible through a publicly specified interface.

out-of-process server

An application that is located on a different system than the requesting client. See also *executable*.

On Windows NT systems, out-of-process servers are usually implemented as .EXE files.

proxy

COM terminology for client-side marshaling code. In RPC terminology, this is called the client stub. The proxy can be an executable or a DLL that runs from the client and communicates with a corresponding stub on the server.

registry

A hierarchical database consisting of one or more files that stores configuration information about system hardware and software.

registry class

Registry element attribute that allows you to store additional descriptive information with a registry key or subkey.

remote activation

The use of distributed COM to resolve a client's call for an object across a network to a remote server.

remote server

A server application, implemented as an executable, that runs on a machine that is different from the machine on which the client application runs.

ROT

Running Object Table. A globally accessible table, on each COM-enabled computer, that tracks all COM objects in a running state. The ROT is used in the process of allowing client applications to identify and access a running server application.

server application

An application that can expose COM objects for use by one or more clients.

stub

COM terminology for server-side marshalling code. In RPC terminology, this is called a server stub. The stub can be an executable or a DLL that runs on the server and communicates with a corresponding proxy on the client.

subkey (registry subkey)

Registry element that is a child of a registry key. A registry key can have zero or more subkeys.

unmarshalling

The unpacking or conversion a sequence of bytes back into a request or response.

value (registry value)

Registry element that is the entry or value for a registry key or subkey.

wrapper

See *encapsulation*.

Index

A

- ACME (Authentication and Credential Management Extensions), 62
- ACME agent
 - NT, 62
 - VMS, 62
- acronyms, 173
- ActiveX, 31
- ActiveX template library. *See* ATL
- Advanced Server for OpenVMS, 58, 65
- Affinity for OpenVMS program, 50
 - waves, 51
- AllConnect Program, 85
- Alliance for Enterprise Computing, 51
- APIs, 93
 - COM, 25, 53
 - RPC, 26
 - supported on OpenVMS, 53
 - supported on Tru64 UNIX, 105
 - Win32, 53
- application
 - architecture, 3
 - compiling a COM application, 78
 - designing, 31
 - developing on Tru64 UNIX, 135
 - encapsulating an existing, 32
 - evolution of, 3
 - linking, 79
 - making COM objects from, 32

- porting a Windows application to Tru64 UNIX, 169
- registering on OpenVMS, 73
- registering on Tru64 UNIX, 112
- scaling, 16
- writing in C or C++, 138
- application registration file
 - creating, 112
- ATL support, 28
- ATL templates, 31
- authentication, 63
- Authentication and Credential Management
 - authority, 67
- authorization, 63

B

- Bristol Technology, 53

C

- C++, 9, 38, 138
 - switches for compiling COM applications on Tru64 UNIX, 140
- character cell interface, 56
- class ID (CLSID). *See* CLSID
- client
 - files needed for, 78
- client/server computing
 - application programming for, 4
- CLSID, 72, 74, 77, 126

COM

- and COM+, 29
 - APIs, 25
 - as middleware, 19
 - benefits of, 16
 - choosing, 22
 - client/server functions, 5
 - comparing across platforms, 25
 - definition of, 3
 - demo CD-ROM, 147
 - designing a new application, 31
 - displaying version of on Tru64 UNIX, 130
 - examples, 13
 - making objects from an existing application, 32
 - on Compaq servers, 16
 - server, starting securely, 61
 - software interoperability, 5
 - terms and concepts, 6
- COM demo
- building the sources, 162
 - CD-ROM description, 147
 - configuring, 156
 - error messages, 164
 - how to demonstrate, 160
 - installing, 154
 - joint effort among OpenVMS, Tru64 UNIX, Microsoft, 52
 - prerequisites, 148
 - prerequisites for Tru64 UNIX, 153
 - prerequisites for Windows NT, 153
 - prerequisites from OpenVMS, 148
 - running, 157
 - what it does, 148
- COM for OpenVMS, 47
- ACME server, 67
 - authentication, 64
 - authorization, 64
 - building an application using the MIDL compiler, 77

- compiling a COM application, 78
 - configuring, 71
 - configuring the DCOM\$RPCSS accounts, 74
 - creating a DCOM\$GUEST account, 74
 - delivery of, 60
 - developing a COM application, 76
 - events, 55
 - generating CLSIDs, 73
 - generating GUIDs, 73
 - generating unique identifiers, 77
 - history, 49
 - impersonation, 64
 - installing, 71
 - linking a COM application, 79
 - MTA threading model, 56
 - NTLM security, 56
 - OpenVMS Registry, 54
 - populating the OpenVMS Registry, 73
 - prerequisites, 71
 - registering an application, 73
 - releases for, 59
 - SCM, 57
 - security, 56, 61
 - starting a server securely, 61
 - starting the server, 73
 - stopping the server, 73
 - support of DCOM events, 55
 - vision of, 52
- COM for Tru64 UNIX, 81
- APIs, 92
 - ATL, 88
 - COM Registry, 88, 92, 111
 - configuring, 131
 - developing COM applications on, 135
 - environment variables, 132
 - events, 92
 - implementation challenges, 93
 - implementation details, 87
 - implementation of, 91
 - infrastructure, 101

- MIDL compiler, 88
- porting COM to, 93
- prerequisites, 97
- programming environment, 100
- releases, 86
- rpcss, 88, 103
- running a COM application on, 95
- SCM, 88
- security, 88
- start Win32 services, 102
- starting a server application on, 94
- stop Win32 services, 102
- system-wide Registry settings, 124
- threading models, 88
- Win32 services, 101
- COM run-time, 98
- COM+, 29
- Compaq Advanced Server for OpenVMS, 71
- Compaq AlphaServers, 16
- Compaq BridgeWorks, 44
- Compaq C compiler, 71, 77
- Compaq C++ compiler, 77
- Compaq Computer Corporation, 3, 6, 84
- Compaq Enterprise Toolkit, 31, 139
- Compaq platforms
 - COM on, 25
- Compaq TCP/IP Services for OpenVMS, 71
- comparing COM across platforms
 - ATL support, 28
 - COM APIs, 25
 - MIDL compiler, 28
 - NTLM security support, 28
 - registry support, 27
 - RPC APIs, 26
 - SCM, 26
 - threading models, 29
- component middleware, 20
- component reuse, 18
- component technology
 - benefits of, 17

- components, 8
 - communication between, 17
 - creating classes, 8
 - current thinking on technologies, 22
 - distributing, 10
 - guidelines for choosing a technology, 23
 - in three or more tiers, 13
 - in two tiers, 12
 - in-proc, 9
 - interfaces, 8
 - methods and parameters, 6
 - out-of-proc, 9
 - out-proc, 9
 - registering, 9
- CORBA
 - choosing, 23
- CORBA implementation, 20

D

- data type macros, 169
- DCE\$RPCD, 26
- DCOM, 6
- DCOM\$CNFG, 72, 74
- DCOM\$GUEST account, 74
- DCOM\$PRCSS
 - configuring, 74
- DCOM\$REGSVR32, 72, 76
- DCOM\$SETUP, 72
- DCOMCNFG, 123
- dcomconfig, 131
- dcomsetup, 99, 111
- dcomver, 99, 130
- df2t, 99, 129
- Digital Equipment Corporation, 3, 6, 50, 53, 84
- Distributed Computing Environment (DCE), 57
- DLL (dynamic link library file), 9

E

encapsulation
 advantages and disadvantages of, 42
 database, 36
 database manipulation routines, 37
 definition of, 33
 examples, 37
 executable files, 34
 functionality, 35
 techniques, 34
 tools, 44
 user interface, 36

endpoint map
 definition of, 27

endpoint mapper, 26, 54, 103, 104

Enterprise JavaBeans (EJB), 22

EXE (executable file), 9

EXTAUTH, 68

F

functionality
 encapsulating, 39

G

globally unique identifier (GUID). *See*
 GUID

GUID, 9, 10, 27, 72, 74, 76, 77, 94, 95, 112,
 113, 171

H

Hewlett-Packard Company, 84

HOSTMAP database, 68

I

IDispatch, 8

impersonation, 64

inet_addr, 172

in-process component
 definition of, 9

in-process server
 files needed for, 78
 on Tru64 UNIX, 94

interfaces
 encapsulating, 37
 IDispatch, 8
 IUnknown, 8
 on Tru64 UNIX, 93

internationalization, 109

interoperability, 85

IUnknown, 8

J

Java, 21

Java/EJB
 choosing, 22

L

libmutant, 98

M

makedef, 137

marshalling, 177

message-oriented middleware, 20

Microsoft, 3, 6, 84

middleware
 component, 20
 CORBA, 21
 definition of, 19
 Java, 21
 message-oriented middleware, 20
 object request broker, 20
 RPCs, 19
 transaction processing, 20

MIDL, 8, 88, 136

MIDL compiler, 28
 input and output, 77

- on OpenVMS, 53
- on Tru64 UNIX, 100, 136
- modular deployment, 18
- module definition file, 137
- monikers, 88
- MTA threads, 29

N

- NetOLE, 6
- network protocols, 104
- NT LanManager (NTLM). *See* NTLM
- ntd, 98
- ntd daemon, 102
- NTLM, 28, 29, 56, 57, 59, 65, 67, 74, 89, 94, 117, 121, 122, 149
- NTLM pass-through, 28, 94, 121, 122
- NTLM security, 56

O

- Object Linking and Embedding (OLE), 6
- Object Management Group, 21
- object model
 - definition of, 5
- object request broker, 20
- OLE, 6
- OLE automation, 88
- olecnfg, 99, 123
 - application-specific registry settings, 126
- Open Group, The, 26, 84
- OpenVMS, 16
 - Affinity for OpenVMS program, 50
 - authentication, 64
 - authorization, 64
 - collbaoration with Tru64 UNIX engineers, 52
 - COM stack on, 59
 - impersonation, 64
 - implementation challenges, 53
 - infrastructure, 49

- infrastructure architecture, 57
- infrastructure projects, 51
- integration with Windows NT, 50
- MIDL compiler on, 53
- OpenVMS Registry, 54
- rpss process on, 54
- security, 28, 56, 58, 59, 61, 68
- starting a COM server securely on, 61
- using OpenVMS and Windows NT
 - security together, 68
- Windows environment on, 53
- Windows NT authentication on, 67
- OpenVMS credentials, 65
- OpenVMS infrastructure, 49
- OpenVMS Registry, 27, 58
 - reading and writing to, 55
 - server management, 55
- out-of-process component
 - definition of, 9
- out-of-process server
 - files needed for, 79
- out-of-process servers
 - on Tru64 UNIX, 94
- OXID key size, 170

P

- password protection, 119
- paulad, 98, 118
- paulas, 98, 118
- persona extension services, 66
- pointers in embedded structures, 170
- primary domain controller (PDC), 118
- proxy DLL
 - files needed for, 78

R

- reengineering
 - vs wrapping, 42
- registry

- COM Registry on Tru64 UNIX, 27
- OpenVMS Registry, 27, 54
- Windows NT registry, 27, 54
- registry support, 27
- regsvr, 99, 115
- remote object activation, 119
- remote procedure call
 - defined, 19
- remote procedure call (RPC). *See* RPC
- reuse
 - component, 18
- RPC, 5, 19, 26, 27, 44, 52, 53, 54, 56, 57, 59, 61, 62, 63, 67, 68, 119, 120, 121, 122, 124
 - DCE RPC, 26
 - MS RPC, 26, 88, 170
 - on OpenVMS, 59
 - RTL, 120
- rpcss, 98
- running objects table (ROT), 26

S

- SAM database, 64
- SCM, 26
 - on OpenVMS, 57
 - on Tru64 UNIX, 98
 - Windows NT, 56
- security, 61
 - on OpenVMS, 28, 56, 58, 59, 61, 68
 - on Tru64 UNIX, 117
- Security Support Provider Interface (SSPI). *See* SSPI
- sermon, 99, 114
- service control manager (SCM). *See* SCM
- shared libraries, 139
 - registration, 115
 - required and optional, 141
- Siemens Nixdorf Information Systeme AG, 84
- SSPI, 28, 58, 59, 62, 65, 67, 89, 117

- STA threads, 29
- stgview, 99, 129
- SYSUAF.DAT, 68
- SYSUAF.DAT file, 64

T

- testing, 17
- threading
 - COM for Tru64 UNIX, 88
- threading models, 29
- TP (transaction processing) systems, 5
- transation processing middleware, 20
- Tru64 UNIX, 16
 - authentication levels with olecnfg, 123
 - COM registry server on, 111
 - compiler and linker switches and flags, 142
 - compiling COM executables and shared libraries on, 139
 - historical context of, 84
 - levels of COM security on, 117
 - NTLM pass-through, 122
 - overview of COM for, 83
 - porting a Windows application to, 169
 - remote object activation, 119
 - security on, 117
 - security subsystem, 118
 - strategic context of, 85
 - utilities, 129
- type libraries, 137

U

- Unicode, 109, 171
- Unicode support, 89
- uniform data transfer (UDT), 88
- UNIX permission mask, disabling, 171
- unmarshalling, 178
- user security profile, 67
- utilities

- on OpenVMS, 71
- OpenVMS Registry server, 54
- Tru64 UNIX, 129
- uuidgen, 112

V

- Visual Basic, 8, 114
- Visual Studio, 32

W

- wchar_t data, 171
- wide character encoding, 171
- Win32 subsystem, 98
- Windows, 49

- COM on, 25
- threading models, 56
- Windows GUI, 56
- Windows NT
 - COM stack on, 58
 - events, 55
 - OpenVMS integration with, 50
 - Tru64 UNIX interoperability, 85
- Windows NT authentication on OpenVMS, 67
- Windows NT credentials, 65
- wrapping, 33
 - advantages and disadvantages of, 42
 - tools, 44
 - vs reengineering, 42

About the CD

Compaq DCOM Demo

This companion CD features the Compaq DCOM Demo, a package of client and server software that demonstrates the operation of COM applications across a mixed NT, Tru64 UNIX, and OpenVMS environment. The CD-ROM contains a pre-built version and a source version of the Compaq DCOM Demo software.

The pre-built version of the DCOM Demo software consists of the executables and required files that demonstrate Distributed COM on Windows NT clients and OpenVMS and Tru64 UNIX Alpha servers. See the Readme file contained on the disc for instructions to install the executables and files and how you run the DCOM Demo software.

The source version of the software consists of the source files for the DCOM Demo. You can use these sources as templates for your own Distributed COM applications or to edit and customize the demo. The source files are the same for both OpenVMS and Tru64 UNIX. If you understand COM, enjoy reading code, or both, feel free to work with the sources. The Readme describes how you build the sources.

System Prerequisites

Windows NT Client Prerequisites

- Microsoft Windows NT Version 4.0 Intel system with Service Pack 3 or later.
- Microsoft Excel to run the Visual Basic statistics application.

You can connect the NT machine on a network with one or more Compaq Tru64 UNIX Version 4.0D (or later) Alpha systems or Compaq OpenVMS Version 7.2-1 (or later) Alpha systems, or both. Minimum: One Alpha server. The network connection requires that TCP/IP be configured on both the client and server platforms.

COM for Tru64 UNIX Server Prerequisites

- Compaq COM for Tru64 UNIX software installed on a Version 4.0D or later Tru64 UNIX operating system and initialize COM services (daemons running).

COM for OpenVMS Server Prerequisites (for Alpha systems only)

- OpenVMS Alpha Version 7.2-1 installed and configured.
- Advanced Server Version 7.2 installed and configured.
- Compaq COM for OpenVMS run time installed and configured.

Technical Support Information

Beyond providing replacements for defective discs, Butterworth-Heinemann does not provide technical support for the contents of this CD-ROM. Send any requests for replacement of a defective disc to Butterworth-Heinemann, Digital Press Customer Service Dept., 225 Wildwood Avenue, Woburn, MA 01801-2041 or e-mail techsupport@bhusa.com. Be sure to reference item number CD-82265-PC.