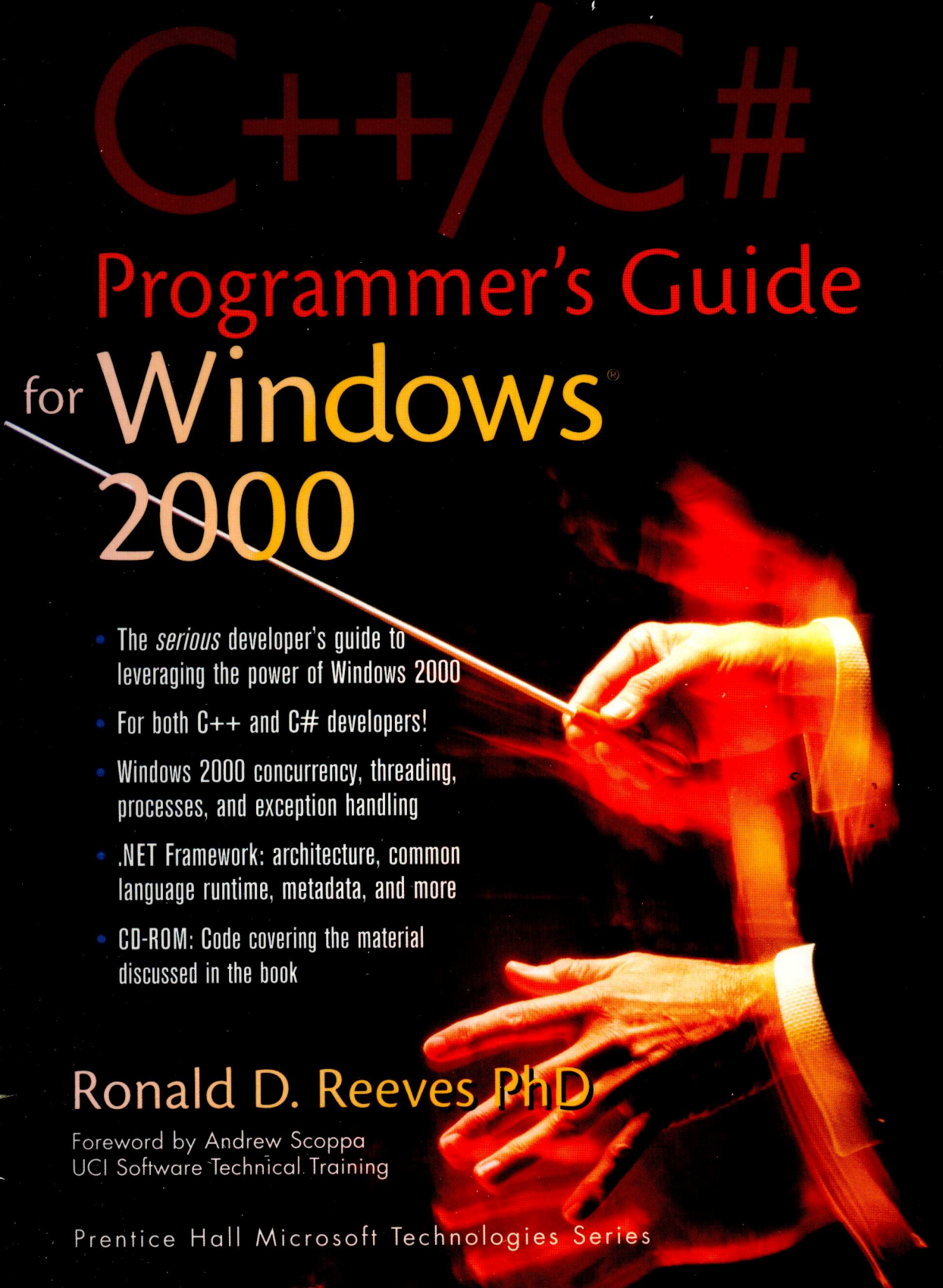
C++/C#

Programmer's Guide for Windows® 2000



- The *serious* developer's guide to leveraging the power of Windows 2000
- For both C++ and C# developers!
- Windows 2000 concurrency, threading, processes, and exception handling
- .NET Framework: architecture, common language runtime, metadata, and more
- CD-ROM: Code covering the material discussed in the book

Ronald D. Reeves PhD

Foreword by Andrew Scoppa
UCI Software Technical Training

Prentice Hall Microsoft Technologies Series

PRENTICE HALL SERIES ON MICROSOFT® TECHNOLOGIES

C++/C#

PROGRAMMER'S GUIDE TO WINDOWS 2000®

Ronald D. Reeves

Prentice Hall PTR, Upper Saddle River, NJ 07458
www.phptr.com

Editorial/Production Supervision: Kathleen M. Caren
Acquisitions Editor: Mike Meehan
Development Editor: Ralph Moore
Cover Design Director: Jerry Votta
Manufacturing Manager: Maura Zaldivar
Series Design: Maureen Eide
Marketing Manager: Debby Van Dijk
Art Director: Gail Cocker-Bogusz

Prentice-Hall, Inc.
Upper Saddle River, New Jersey 07458

Prentice Hall books are widely used by corporations and government agencies for training, marketing, and resale. The publisher offers discounts on this book when ordered in bulk quantities. For more information, contact:
Corporate Sales Department,
Prentice Hall PTR
One Lake Street
Upper Saddle River, NJ 07458
Phone: 800-382-3419; FAX: 201-236-7141
E-mail (Internet): corpsales@prenhall.com

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-040947-2

Pearson Education Limited (UK)
Pearson Education Australia Pty Ltd
Prentice Hall Canada Ltd
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education Japan KK
Pearson Education China Ltd
Pearson Education Asia Pte Ltd
Prentice Hall, Upper Saddle River, New Jersey

FOREWORD

Visual Studio.NET represents a significant step forward in the continuing evolution of software application development environments. The .NET Framework represents our greatest step forward to date, in using the rich semantics of classes and object-oriented design in communication, and control of the powerful Windows 2000 Operating System. A very powerful synergy is formed between the developer, the compiler, the .NET Framework base classes, the Common Language Runtime (CLR), and the Windows 2000 Operating System. A great deal of the cognitive load is taken off the developer, by the encapsulating of the Windows 2000 Operating System Win32 APIs by the .NET Framework base classes. The intelligent collaboration between the base classes, and the CLR automates and controls many functions involved in Windows 2000 application development and execution.

Over the past 7 years, Dr. Ron Reeves has been a consultant and trainer for UCI Software Technical Training. As a trainer, Ron will consistently go the extra mile to make sure his students clearly understand the material and at the same time make the learning experience enjoyable. As an author, he demonstrates yet another remarkable talent. He is an excellent and gifted writer. As a reader of this book you will like what he has to contribute.

In this first-class book, Ron has borrowed on his 40 plus years of computer system design and implementation to discuss this revolutionary approach to creating software applications. He takes a bottom-up approach to explaining how this whole new architecture fits together. First, he covers the Windows 2000 Operating System architecture and what major components are in it. The next chapter then reviews how the Win32 APIs are used to develop an application to run under Windows 2000. Of course this is the unmanaged mode of Visual C++ and is the default for the compiler. Then he covers the architecture of the .NET Framework and the significant components involved in it. He shows the relationship of this framework to the Win32 APIs of Windows 2000. He then steps on up to the Visual C++ compiler, and how it is structured to work in the new environment. The last chapter then covers the new C# compiler and how it is structured to work in the new environment. The book also points out how we are continuing to

develop more and smarter META layers of software between the developer and the under lying hardware engine. It shows how these META layers affects the developer's cognitive understanding of the structure.

Andrew Scoppa
UCI software Technical Training

PREFACE

Windows 2000 is a large and important system, and it is the core of a more embracing architecture Microsoft calls Windows DNA 2000. In this context DNA stands for Distributed interNet Applications, and represents Microsoft's vision for building distributed systems. This type of architecture is focused on developing the new "digital nervous system" for enterprises. In this context, the "digital nervous system" is the corporate, digital equivalent of the human nervous system: an information system that can provide a well-integrated flow of information at the right time, to the right place in an organization. Such systems can be programmed at many levels, from the lowest level of device drivers giving access to privileged instructions, to very high levels using powerful software application development tools. This book is aimed at Windows 2000 application programming, using C++/C# and the Visual Studio.NET development environment. The C++/C# and Visual Studio.NET discussions and examples are based upon the BETA 1Win32 programming required for Windows 2000. The book should prove suitable for programmers migrating to Windows 2000 from other environments, such as UNIX and mainframes, as well as for programmers moving up from earlier versions of Windows. A large part of the book addresses issues of what components actually make up the .NET Framework and the Windows 2000 Operating System. One must realize, there are numerous constraints among all the components, and one needs to try to understand, from the beginning, how they fit into the whole .NET Framework and the Windows 2000 Operating System.

Learning such complex technology can be quite a challenge. The documentation is vast, equivalent to tens of thousands of printed pages, and it is changing all the time. You can subscribe to various Internet discussion groups, and you will receive hundreds of emails every day. There are many, many books on different parts of this technology. But, how do you grasp the whole picture? This book aims to be holistic, to provide a practical guide for the C++/C# programmer. It is not a substitute for documentation or more specialized books, including "bibles" of various sorts that help you learn different APIs. Rather, the book provides a tutorial, giving you all the basic information you need to create working Windows 2000 application systems. The book and companion CD has many example programs in both C++ and C# to aid you in gaining an understanding of how the whole environment fits together.

Chapter 1 is an introduction showing an architectural overview of Windows 2000. It shows an overall block diagram of Windows 2000 and then discusses in general some of the key components of Windows 2000. The chapter also contains a general description of the different versions of Windows 2000.

Chapter 2 covers the most essential fundamentals of Windows 2000 programming for C++/C# programmers. We start off with an architectural overview of Windows 2000. There is enough detail to enable an experienced C++ programmer new to Windows 2000 to get an understanding of Windows 2000. This chapter also covers the concepts of processes, threads, jobs, and the handling of errors and exceptional conditions. The software priority structure is also covered in this chapter. The chapter explains the use of Win32 APIs for programming without the use of the .NET Framework base classes. We will see however, that the .NET Framework base classes almost completely encapsulate these Win32 APIs for application development. As a C++ programmer you can still, in native mode, work with the Win32 APIs if you should choose to do so. You can also mix native mode and managed code mode in your application components. C# works, as we will see, in managed code mode only. There are keywords, however, to let the C# code have sections of native mode code. Visual Studio.NET, as we will see in Chapter 4, has one standard approach to handling errors and exceptions. These topics bear directly on the issues of being able to create scalable and robust applications discussed above. The recently published book, *Win32 System Services—The Heart of Windows 98 and Windows 2000*, by Ron Reeves and Marshall Brain, covers in detail the use of Win32 APIs for application development.

Chapter 3 covers the most essential features of the new .NET Framework primarily from an architectural point of view. This material is the *raison d'être* for the book. It is expected that the .NET Framework will become absolutely central to modern Windows application architecture and programming development. Hence, it is important for you to understand the basics. This chapter will give you the background for the discussions in Chapters 4 and 5 on Visual C++ and C#. COM+ will continue to play an important role in the development of multiple-tier application systems. For COM+ details, there are many other books on the subject, including Robert Oberg's book, *Understanding and Programming COM+—A Practical Guide to Windows 2000 DNA*.

Chapter 4 covers the Visual C++ 7.0 compiler and what is involved in using the compiler to create applications. The discussion is primarily based upon the use of C++ in a managed code mode, because that is the only mode that uses the Common Language Runtime (CLR) component. This mode picks up all the advantages of the new management features of the CLR. It discusses in detail the Managed Extensions for C++ that enables the programmer to take advantage of the .NET Framework architecture. The

chapter covers the considerations of using the compiler in applications, as opposed to just a language syntax discussion.

Chapter 5 covers the C# compiler and what is involved in using the compiler to create applications. Windows 2000 and .NET Framework is expected to rely heavily on C# for enterprise level system development. Also, this new approach to distributed processing using C# does not require the System Register for any of its activity—just a language syntax discussion.

The appendices cover in detail the supporting material for the chapters. In some cases, a given appendix will be as big as a chapter. All the Win32 APIs and the .NET Framework base classes are listed in the appendices, along with software priority charts, and so on.

CONTENTS

Foreword *xi*
Preface *xiii*

▼ ONE Introduction 1

Windows 2000 Operating System Architecture *3*
Executive 3
Protected Subsystems 5
Local Procedure Call Facility 5

▼ TWO Processes, Threads, and Jobs in Windows 2000 *9*

Object Categories *11*
Processes 11
Creating and Terminating Processes *11*
Terminating a Process *16*
Process Use of Mutexes, Semaphores, and Events *17*
Process Security and Access Rights *18*
Threads *19*
Creating and Terminating Threads *19*
Terminating a Thread *22*
Suspending Thread Execution *23*
Thread Stack Size and Thread Local Storage *23*
Thread Synchonization *27*
Mutex and Semaphore Creation *28*
Acquiring Mutexes, Semaphores, and Releasing *29*
Events *30*
Critical Section Objects *31*
Thread Priorities *32*
Thread Multitasking *35*

Thread Pooling 36
Thread Security and Access Rights 37
Jobs 38
Creating, Opening, and Terminating Jobs 39
Acquiring Job Status Information 41
Managing Job's Processes 43
I/O Completion Port and Job Notification 44
I/O Completion Ports 45

▼ THREE .NET Framework 47

Introduction 47

.NET Framework Base Classes 52
Common Type System 55
Classes 56
Interfaces 56
Value Types 58
Enumerations 59
Delegates 60

Common Language Runtime 61
Managed Execution 63
Microsoft Intermediate Language (MSIL) 64
JIT Compilation 64
Assemblies 65
Assembly Concepts 66
Versioning and DLL Conflicts 66
An End to DLL Conflicts 67
Assemblies and Deploying 67
The Minimum You Need to Know About Assemblies 67
Assembly Manifest 68
Assembly Custom Attributes 70
Creating Assemblies 71
Naming an Assembly 72
Assembly Location 72
Loader Optimization 73
Shared Name 74
How to Assign and Reference a Shared Name 74
Assemblies and Security Consideration 75

Assemblies and Versioning 76
How the Runtime Locates Assemblies 77
Step 1: Initiating the Bind 77
Step 2: Version Policy in the Application Configuration 77
Step 3: Locating the Assembly Through Codebases or Probing 79
Locating the Assembly Through Codebases 79
Locating the Assembly Through Probing 79
Explicit Codebases 81
Probing URLs 81
Step 4: The Global Assembly Cache and Auto-QFE Policy 81
Step 5: Administrator Policy 82
Partially Specified References 82
How the Runtime Determines Type Identity 83
Namespaces 83
How the Runtime Uses Assembly Version Information 83
An Assembly's Informational Version 84
Specifying Version Policies in Configuration Files 85
Default Version Policy 85
Bind to a Specific Version 86
Don't Accept Implicit QFEs 86
Safe Mode 87
Application Domains 87
Unloading and Threads 89
Metadata and Self-Describing Components 89
What is Metadata? 89
Description of PE or Assembly 90
Description of Types 90
Attributes 90
The Benefits of Metadata 90

▼ FOUR Visual C++ 7.0 93
Introduction 93
C++ Programming Language 94
Managed Code and Targeting the .NET Framework 94
Programming in Native Code 95

Attributed Programming 95
ATL Server 95
New Intergrated Debugger 96
Event Handling in Visual C++ 96
Visual C++ Editions 96
What's Included in the Visual C++ Standard Edtion 97
What's Included in the Visual C++ Professional Edition 98
Programming Features 98
Internet 98
ActiveX Controls 98
Project Features 98
Optimizations 98
Database Support 99
What's Included in the Visual C++ Enterprise Edition 99
Programming with Managed Extensions for C++ 99
When to Use Managed Extensions for C++ 100
Introduction to Managed Extension for C++ 101
Managed Types 101
Garbage-Collected Classes 103
Destroying a Garbage-Collected Class 103
Value Classes 104
Value Classes and Boxing 105
Managed Interfaces 107
Implementation of Ambiguous Base Interface Methods 108
Default Implementation of a Method 108
Managed Arrays 109
Automatic Array Initialization 110
Multidimensional Arrays 110
Array Covariance 111
Delegates in Managed Extensions for C++ 112
Single-Cast Delegates 112
Multicast Delegates 112
Properties of Managed Objects 114
Property Types in Managed Extensions 115
Scalar Properties 115
Indexed Properties 115

Adding Support for Managed Extensions for C++ to an Existing Application 117
Modifying the Existing Project Settings 117
Employing New Managed Extensions Functionality in Existing Applications 117
Handling Exceptions Using Managed Extensions for C++ 118
Basic Concepts in Using Managed Extentions 119
Throwing Exception Using Managed Extentions 119
Try/Catch Blocks Using Managed Extensions 119
Order of Unwinding for C++ Objects 121
Catching Unmanaged C++ Types 121
Managed Extensions and The_Finally Keyword 121
C++ Exceptions Examples 122

▼ FIVE C# 127

Introduction 127
Comparison Between C++ and C# 128
General Structure of a C# Program 128
C# Version of Hello World 132
Developing a Simple Windows Forms Control 134
Class versus Component versus Control 134

▼ APPENDIX A APIs 139

▼ APPENDIX B Base Priority 143

▼ APPENDIX C Object Categories 147

▼ APPENDIX D Functions in Alphabetic Order (1939 APIs) 149

▼ APPENDIX E Win32 API Functions by Category (95) 173

▼ APPENDIX F Win32 Data Types 259

▼ APPENDIX G .NET Framework Namespaces 265

▼ APPENDIX H Attributes 277

▼ APPENDIX I Debugging Visual C++ 309

▼ APPENDIX J Event Handling in Visual C++ 321

▼ APPENDIX K Managed Extensions for C++ Reference 333

▼ APPENDIX L /CLR (Common Language Runtime Compilation) 335

▼ APPENDIX M C# Compiler Options 339

▼ APPENDIX N C# Keywords 345

▼ APPENDIX O C# Libraries Tutorial 349

About the Author 357

About UCI 359

Index 361

ONE

Introduction

In the next few sections we will establish the main Windows 2000 Operating System capabilities for the fundamentals involved in programming for Windows 2000. There are obviously many more sections that could be covered, but we deemed these to be most important for understanding the programming of Windows 2000. This fundamental knowledge, along with the other topics covered in this book, will enable you to program the many Windows 2000 services. Included in the Windows 2000 services are base services, component services, data access services, graphics and multimedia services, management services, messaging and collaboration services, networking services, security, tools and languages, user interface services, and Web services. The topics covered in the overall book should put into perspective these Windows 2000 services, and provide a kind of roadmap for how to program these capabilities. With this understanding, we will go into the Visual Studio.NET software development environment and show you how to implement applications under this new development environment. We will show you how this seamless new development environment lets you control and harness the power of the Windows 2000 engine.

One of the key architectural features of a sophisticated operating system, which enables it to handle many activities concurrently, is the software priority scheme. Usually, this is one of the first things I try to figure out when I want to start developing application programs using the Operating System (OS). The OS also uses a hardware priority scheme, but we will not cover that in this book. The next thing that I like to understand is the interface communication capabilities APIs (Application Programming Interface) that are available to me for controlling and communicating with the OS. Of course, rolled into this scheme of things is how the OS partitions work and allocate resources to the various application software activities, especially

memory allocation. The software synchronizing mechanism, available to control application and system resource allocations, is closely related to the software priority scheme. In a pseudo real-time OS like Windows 2000 it is very important to be able to synchronize the various events relative to thread priorities. We will cover these aspects in detail as we proceed through the various sections. These fundamentals work the same across all Windows 2000 platforms. The Windows 2000 platform consists of four products as follows:

- Windows 2000 Professional
- Windows 2000 Server
- Windows 2000 Advanced Server
- Windows 2000 Data Center

Microsoft has done an excellent job, in that the Win32 API family of Windows 2000 programming interfaces is the standard programming interface for all Windows 2000 platforms. The Win32 APIs are the interface communication capabilities (APIs) that I mentioned earlier, which allow the programmer to control and communicate with the various capabilities of Windows 2000. Microsoft has also provided a rich set of libraries, the Windows Foundation Classes (WFC). The .NET Framework wraps the Windows 2000 and provides a very rich semantic interface for our use of the Windows 2000 Operating System capabilities. The Visual C++ compiler, and the C# compiler and associated tools of the Integrated Development Environment (IDE), allows program development, execution, and debug within this IDE. This environment gives the programmer a simpler but powerful tool, for developing application software using the Windows 2000 APIs and associated services. If one is programming using either the WinForm or the Web Services APIs that wrap the Win32 APIs, they significantly reduce the software footprint the application programmer needs to worry about to create their applications. The Microsoft Foundation Class (MFC) library and the Active Template Library (ATL) are also available to the application developer. These libraries simplify the creation of COM+ components, Graphical User Interfaces (GUI), database interfaces, and other aspects of application development. Interestingly enough, ATL is integrated into MFC and allows the best of both worlds for GUI and small efficient software component development.

As we proceed, we will use both the C++ compiler and the C# compiler, and use them with the associated tools of the new Visual Studio.Net Integrated Development Environment (Visual Studio.Net is the next release after Visual Studio 6.0). We are using Visual Studio.Net BETA 1 so there will be some changes by the time of the final release. However, the software development environment does not affect the fundamentals of the Windows 2000 OS, which is what we will cover in the first few chapters of the book. We are covering the fundamentals from the standpoint of API calls and using them for your application. Visual Studio.Net does introduce another frame-

work that wraps the Windows 2000 OS; this is the .NET Framework. We will cover this new framework after we cover Windows 2000. This framework is middleware that sets between your application and the Windows 2000 OS.

Windows 2000 Operating System Architecture

Figure 1–1 shows the overall view of the major blocks in the Windows 2000 Operating System. As the block diagram shows, applications are kept separate from the operating system itself. The operating-system code runs in a privileged processor mode known as the *kernel* and has access to system data and hardware. Applications run in a nonprivileged processor mode known as *user mode* and have limited access to system data and hardware through a set of tightly controlled APIs. One of the design goals of the Windows 2000 operating system was to keep the base operating system as small and efficient as possible. This was accomplished by allowing only those functions that could not reasonably be performed elsewhere to remain in the base operating system. The functionality that was pushed out of the kernel ended up in a set of nonprivileged servers know as the *protected subsystems*. The protected subsystems provide the traditional operating system support to applications through a feature-rich set of APIs.

Executive

The kernel-mode portion of the Windows 2000 operating system is called the *Executive* and, except for a user interface, is a complete operating system unto itself. The Executive is never modified or recompiled by the system administrator. The Executive is actually a family of software components that provide basic operating-system services to the protected subsystems and to each other. The Executive components, as shown in Figure 1–1, include:

- I/O Manager
- Object Manager
- Security Reference Monitor
- Process Manager
- Local Procedure Call Facility
- Virtual Memory Manager
- Window Manager
- Graphics Device Interface
- Graphics Device Drivers

The Executive components are completely independent of one another and communicate through carefully controlled interfaces. This module design allows existing Executive components to be removed and replaced with ones