ENTEGRITY Solutions®

PC-DCE™

Developer's Notes

Software Version 5.0

Notices

Contents

Notices 2

Preface 5

Intended Audience 5
Documentation 5
  PC-DCE Documentation Set 5
  The Open Group Documentation 6
Obtaining Technical Support 6
Contacting Entegrity Solutions 7
Obtaining Additional Technical Information 7

Chapter 1 Overview 9

Chapter 2 Notes on Compiling and Linking 11

2.1 Linking and Compiling PC-DCE Applications 11
  2.1.1 Using Non-Microsoft Compilers 11
2.2 Exception Handling 11
  2.2.1 Global Unwinding 12
  2.2.2 DCE Exception Handler Definitions for C++ 12
  2.2.3 Runtime Library Requirement 13
  2.2.4 Mapping WIN32 Exceptions to DCE Threads Exceptions 13
2.3 Using OLE 2.0 with PC-DCE 13
2.4 Generating Stubs from the IDL Command Line 13
2.5 Using the Microsoft IDE Interface 14

Chapter 3 Migration Notes 15

3.1 Migrating UNIX DCE Applications to PC-DCE 15
  3.1.1 Retrieving Errors on DCE API Calls 16
3.2 Migrating Applications Developed With Compaq DCE for Windows NT 16
3.3 Migrating 16 bit Windows Applications To 32 bit Windows 17
3.4 Migrating Applications Developed Using Earlier Releases of PC-DCE 18
  3.4.1 Mutexes 18

Chapter 4 Miscellaneous Notes 19

4.1 Developing C++ Applications 19
  4.1.1 Typical Development Procedure 19
  4.1.2 Pthread Exceptions vs. MS Visual C++ 20
  4.1.3 Calling Conventions vs. MS Visual C++ 20
4.2 DCE Message Catalogs 21
  4.2.1 Generating Message Catalogs from .sams Files 21

4.2.2 NT_EVENTLOG Routing 21
4.3 Static Initialization of Constants 22
4.4 Microsoft RPC Interoperability 22
4.5 Name Service Interface Gateway 22
4.5.1 Supported nsid Protocols 22
4.5.2 How the nsid Works 22
4.5.3 How the PC Client Finds the nsid 23
4.5.4 APIs Supported by the nsid 23
4.6 DCECP Functionality 24
4.7 .DBG Files 24
4.8 Max Call Requests 25
4.9 External Time Providers 25
4.10 Controlling RPC Timeouts 25
4.11 Setting a Session-Wide Login Context 26
4.12 Converting an Exception to an Error Status Code 26
4.13 Starting PC-DCE Services from the Command Line 27
4.14 Developing Thread-Safe Code 28
4.15 Setting Thread Priority Under Windows 28
4.16 Using Per-Thread Login Contexts 28
4.17 Using DCE Pipes 29
4.18 Runtime Version Requirement 29

Chapter 5 Entegrity API Extensions 31

5.1 sec_login_set_default_context() 32
5.2 sec_login_setup_win32_context() 33
5.3 sec_login_set_thread_context() 34
5.4 sec_login_unset_thread_context() 35
5.5 sec_login_get_default_context() 36

Chapter 6 Samples 37

6.1 Using the Sample Applications 37
6.2 Greet Application 37
6.2.1 The Greet Client 38
6.2.2 The Greet Server 38
6.2.3 Running the Greet Sample Programs 38
6.3 Grade Application 40
6.3.1 Setting Up the Grade Server Application 40
6.3.1.1 Security Setup 41
6.3.1.2 CDS Setup 42
6.3.2 Running the Grade Server Sample Programs 42
6.3.2.1 Running the Grade Server Program 42
6.3.2.2 Running the Grade Client Program 43

Index 45

Preface

Intended Audience

This guide is for programmers who are developing distributed applications for the Entegrity® DCE implementation for Windows® called PC-DCE™. This guide assumes a general knowledge of DCE (Distributed Computing Environment).

Documentation

This section describes the documentation that Entegrity provides with PC-DCE on both the product CD and on the Entegrity web site (**www.entegrity.com**) under the Support link:

- Entegrity PC-DCE Documentation Set
- The Open Group Documentation

Documentation on other Entegrity products, such as NetCrusader/Web, is also available on the Entegrity web site.

We are always trying to improve our documentation. If you notice any inaccuracies or cannot find information, please send email to **docs@entegrity.com**. We welcome any comments or suggestions.

PC-DCE Documentation Set

The following documents are provided with PC-DCE:

- *PC-DCE Installation and Release Notes*
- *PC-DCE Overview Guide*
- *PC-DCE Administrator's Guide*
- *PC-DCE Developer's Notes* (this file)
- *PC-DCE Guide to CAS*

PC-DCE also provides online help with the following programs:

- PC-DCE Service Panel
- PC-DCE Configuration Panel
- DCE Director
- Visual DCE ACL Editor
- DCEsetup

Entegrity also provides OSF DCE Version 1.2.2 documentation on the product CD and the Entegrity Support web site.

The Open Group Documentation

The PC-DCE product CD and the Entegrity Support web site also provide The Open Group (formerly OSF) DCE Version 1.2.2 documentation, including the following guides:

- *OSF DCE Administration Guide — Core Components*
- *OSF DCE Administration Guide — Introduction*
- *OSF DCE Application Development Guide — Core Components*
- *OSF DCE Application Development Guide — Directory Services*
- *OSF DCE Application Development Guide — Introduction and Style Guide*
- *OSF DCE Application Development Reference*
- *OSF DCE Command Reference*
- *Introduction to OSF DCE*
- *OSF DCE Problem Determination Guide*
- *OSF DCE/File-Access Administration Guide and Reference*
- *OSF DCE/File-Access Users’ Guide*
- *OSF DFS Administration Guide and Reference*
- *OSF GDS Administration Guide and Reference*

Obtaining Technical Support

If you purchased PC-DCE directly from Entegrity Solutions or Gradient Technologies, you are entitled to 30 days of limited technical support beginning on the day the product is expected to arrive.

You may also purchase a support plan that entitles you to additional services. You *must* register prior to receiving this support. For details, refer to the customer support information package that accompanied your shipment or refer to **http://support.entegrity.com**. The web site also contains online forms for easy registration.

If you purchased PC-DCE from a reseller, please contact the reseller for information on obtaining technical support.

Contacting Entegrity Solutions

Contact	Address	Phone/Fax/Email
DCE Product and Sales Information	Entegrity Solutions Corporation 410 Amherst Street, Suite 150 Nashua, NH 03063 USA	Email: DCESales@entegrity.com Web: www.entegrity.com Tel: +1-603-882-1306 ext.2700 Toll Free (US): 1-800-525-4343 ext. 2700 Fax: +1-603-882-6092
All Other Product and Sales Information Requests	Entegrity Solutions Corporation 2001 Gateway Place, Suite 420W San Jose, CA 95110 USA	Email: info@entegrity.com Web: www.entegrity.com Tel: +1-408-487-8600 ext. 123 Fax: +1-408-487-8610
Technical Support	Entegrity Solutions Corporation 410 Amherst Street, Suite 150 Nashua, NH 03063 USA	Email: support@entegrity.com Web: support.entegrity.com Tel: +1-603-882-1306 ext. 2702 Toll Free (US): 1-888-368-3555 ext. 2702 Fax: +1-603-882-6092
Documentation Comments and Suggestions		Email: docs@entegrity.com
Other Inquiries	Entegrity Solutions Corporation 2001 Gateway Place, Suite 420W San Jose, CA 95110 USA	Email: info@entegrity.com Web: www.entegrity.com Tel: +1-408-487-8600 Fax: +1-408-487-8610

For a complete listing of Entegrity Solutions Corporation sales, research and development, and solutions centers worldwide, please see the Entegrity web site at http://www.entegrity.com.

Obtaining Additional Technical Information

Contact	Address	Phone/Fax/Email
The Open Group™ Developer of DCE (Distributed Computing Architecture) software and standards.	The Open Group 29B Montvale Ave. Woburn MA 01801 U. S. A.	Tel: +1 781-376-8200 Fax: +1 781-376-935811 **http://www.opengroup.org**

CHAPTER 1

Overview

1

The PC-DCE Developer's Notes provides information about developing DCE applications for PC-DCE.

Additional information about writing DCE applications may be found in the following OSF DCE guides:

- *OSF DCE Application Development Guide — Core Components*

  This guide describes how to put DCE's core components (host services, application message service, serviceability, backing store database services) to work in your applications

- *OSF DCE Application Development Guide — Directory Services*

  This guide describes how application developers can access the DCE Directory Services including the DCE Cell Directory Service (CDS), the DCE Global Directory Service (GDS), and the X/Open Directory Service (XDS) and X/Open OSI-Abstract-Data Manipulation (XOM) programming interfaces.

- *OSF DCE Application Development Guide — Introduction and Style*

  This guide provides information about how to program the application programming interfaces (APIs) provided for each OSF DCE component.

Developer's may also use the search capabilities of the Entegrity Support web (**http://support.entegrity.com**.) for quick access to specific development information from these and other OSF DCE documents.

CHAPTER 2

Notes on Compiling and Linking

2

This chapter provides information on compiling and linking PC-DCE™ programs. It contains the following sections:

2.1 Linking and Compiling PC-DCE Applications
2.2 Exception Handling
2.3 Using OLE 2.0 with PC-DCE
2.4 Generating Stubs from the IDL Command Line
2.5 Using the Microsoft IDE Interface

2.1 Linking and Compiling PC-DCE Applications

When you compile your PC-DCE programs, be sure to use the following compile switches:

```
-DWIN32

-D_MT

-D_DLL

-DWIN32_LEAN_AND_MEAN
```

The DCE API is exported via **dce32.dll**, so you must link your applications with **dce32.lib**.

If you do not want to link with the DLL version of the C runtime library, be careful when freeing memory returned by DCE API calls and RPCs. Use **rpc_sm_client_free()** to free this memory.

2.1.1 Using Non-Microsoft Compilers

Special instructions are available if you are using the OMF-compliant link library (**dce32_omf.lib**) supplied in this release to build PC-DCE applications with the Borland C compiler. Although not yet exhaustively tested in other non-MS environments, this library should also support the Symantec build environment.

Visit our Technical Support web page for PC-DCE at **http://support.entegrity.com** for more detailed instructions on using this library.

2.2 Exception Handling

PC-DCE provides support for exception handling that conforms to OSF DCE threads specification.

NOTE: PC-DCE exception handling is not the same as exception handling provided by Visual C++ and the native WIN32 support for structured exception handling. For a discussion of how to use DCE exception handling, refer to the *OSF DCE Application Development Guide*.

2.2.1 Global Unwinding

The exception handler provided by the C++ language provides global unwinding. In global unwinding, when an exception occurs, the exception handler calls the destructors for all objects on the stack between where the exception is raised and where it is caught. If you are accustomed to C++ exception handling, you should be aware that DCE exceptions do not provide a similar feature.

2.2.2 DCE Exception Handler Definitions for C++

C++ and DCE both define certain keywords, such as TRY, for exception handling. To avoid conflict in PC-DCE programs written in C++, Entegrity® has renamed the DCE macros using the prefix DCE_ as shown in *Table 2-1*.

Table 2-1: DCE Exception Handler Macro Redefinitions

DCE Default Name	PC-DCE Name
RAISE	DCE_RAISE
TRY	DCE_TRY
CATCH	DCE_CATCH
CATCH_ALL	DCE_CATCH_ALL
RERAISE	DCE_RERAISE
FINALLY	DCE_FINALLY
ENDTRY	DCE_ENDTRY

In C programs, PC-DCE (in **exc_handling.h**) automatically redefines all instances of the DCE default names to the PC-DCE macro names. Programs written using the default names will use the DCE exception handler.

In C++, you must define the preprocessor directive STANDARD_DCE_EXCEPTIONS in order to redefine the DCE default names to the PC-DCE names. If STANDARD_DCE_EXCEPTIONS is not defined, you must use the PC-DCE names to invoke the DCE exception handler.

Be aware that the DCE and C++ exception handlers are not compatible.

2.2.3 Runtime Library Requirement

The PC-DCE exception handler implementation uses **setjmp()** and **longjmp()** calls. If your application uses PC-DCE exception handling, the runtime library that you link to your application must provide a **setjmp()** and **longjmp()** implementation compatible with the Microsoft Visual C++ runtime. This includes the size and semantics of the jump buffer allocated for these calls.

2.2.4 Mapping WIN32 Exceptions to DCE Threads Exceptions

PC-DCE exception macros use WIN32 structured handling support by default to map WIN32 exceptions such as EXCEPTION_ACCESS_VIOLATION (**winbase.h**) to equivalent DCE threads exceptions (for example, in this case, to **exc_e_illaddr**).

As a side effect of this mapping, these exceptions are no longer propagated up via WIN32 structured exception handling to be handled by an external debugger. Non-production environments and environments that do not care about this mapping can disable this feature by calling the function **__exc_w32_to_dce_map_set** with an argument of 0 (false). This function, exported from **dce32.dll** and defined in **dce/exc_handling.h**, disables the mapping for the current process. You must call this function before executing any code block that uses PC-DCE exception handling.

2.3 Using OLE 2.0 with PC-DCE

OLE 2.0 supports distributed OLE calls which internally use the Microsoft RPC to handle the distributed part of OLE. OLE and DCE both define certain types (such as HANDLE_T and UUID_T) in their header files. If a source file includes both OLE and DCE header files, compile errors occur.

If you need to use OLE and DCE in your application, make sure you do the following:

- Use the -DWIN32_LEAN_AND_MEAN compile option.
- Make sure that your source files do not include both DCE and OLE header files. Store all OLE calls inside one file and all DCE calls inside another.

2.4 Generating Stubs from the IDL Command Line

You can use the PC-DCE IDL compiler to generate stubs directly into object code from the IDL command line. For example, to run IDL and directly compile the generated stubs, use the following command:

```
prompt> idl -no_mepv -v -cc_opt "-nologo -I. -DWIN32_LEAN_AND_MEAN \
-DWIN32 -D_MT -D_DLL"
```

2.5 Using the Microsoft IDE Interface

To use the Microsoft IDE interface with PC-DCE applications:

1 Add the following preprocessor definitions:

   WIN32
   WIN32_LEAN_AND_MEAN
   _MT
   _DLL

2 Add *install_directory***include** and *install_directory***include\dce** and your current directory as Additional Include Directories.

3 Add the *install_directory***lib** as a Library Directory.

4 Add the **dce32.lib** library to your project.

5 Compile your IDL file. Be sure to include the **-keep c_source** option to the IDL precompiler.

6 Add the ***_cstub.c** or ***_sstub.c** file generated to your project.

CHAPTER 3

Migration Notes

3

This chapter contains information on how to migrate applications developed for environments other than 32 bit Windows, or with other DCE vendors. It contains the following sections:

3.1 Migrating UNIX DCE Applications to PC-DCE
3.2 Migrating Applications Developed With Compaq DCE for Windows NT
3.3 Migrating 16 bit Windows Applications To 32 bit Windows
3.4 Migrating Applications Developed Using Earlier Releases of PC-DCE

3.1 Migrating UNIX DCE Applications to PC-DCE

Keep the following in mind when porting your existing DCE applications from a UNIX system to PC-DCE:

- Time Synchronization within a Cell

  If a DTS Clerk is running on the client, it functions like its UNIX counterpart. If such a clerk isn't running in Windows 98, our **dce32init** program will automatically synchronize your clock with a DTS Server running on the Security Server. Windows 2000 and Windows NT have a similar program called **timesync.exe** that does the same thing without the clerk running.

- Thread Scheduling

  Although the **pthreads** library on UNIX systems has many possible scheduling methods, the Windows environments PC-DCE supports only support the Win32 thread scheduling method. This method is a modified round robin technique that increases priority for waiting threads, decreases priority for running threads, and increases priority for threads holding resources required by higher priority threads. Any thread that receives a Win32 message also has its priority raised. Applications can use the Win32 API to change thread priority attributes.

- Integrated Login

  PC-DCE provides the integrated login function, in which logging into a Windows account also logs you into a mirror DCE account.

- Partitioning and Security

If you are using NTFS partitioning on a Windows 2000 or Windows NT system and plan to migrate existing applications from a UNIX environment, you will have comparable security to what was previously available. However, this security is tied to Windows user accounts and not to DCE accounts. If you are using the FAT file system, you will have minimal security.

- On UNIX systems, **rpc_ss_allocate()** and **rpc_ss_free()** can be used as callback arguments in DCE API calls. With PC-DCE, you may need to wrap these calls and create **__cdecl** versions as shown below so that they match argument prototypes for the API call used.

```
#if defined(WIN32)

/* wrap rpc_ss_allocate in a __cdecl call so arg pointers will work */
static idl_void_p_t rpc_ss_allocate_cdecl(idl_size_t size)
{
return rpc_ss_allocate(size);
}
#define rpc_ss_allocate rpc_ss_allocate_cdecl

/* wrap rpc_ss_free in a __cdecl call so arg pointers will work */
static void rpc_ss_free_cdecl(idl_void_p_t ptr)
{
return rpc_ss_free(ptr);
}
#define rpc_ss_free rpc_ss_free_cdecl
#endif /* WIN32 */
```

You can also use the **cdecl_rpc_ss_allocate**, **cdecl_rpc_ss_free** and **cdecl_rpc_ss_client_free** calls from the **dce32.dll** library.

3.1.1 Retrieving Errors on DCE API Calls

In PC-DCE applications, failed **pthread** API calls (and other failed DCE API calls) return an error to the application via the Win32 API call **GetLastError()** rather than by setting the **errno** variable as expected in UNIX DCE implementations.

This mechanism for PC-DCE error retrieval is necessary because there is no way to set the **errno** variable from the PC-DCE runtime that works reliably for all C compilers or for all versions of a particular compiler's C runtime library. To retrieve the **errno** value of the most recent failed DCE API call, you must use **GetLastError()**.

3.2 Migrating Applications Developed With Compaq DCE for Windows NT

Applications developed using Compaq DCE for Windows NT will have to be recompiled and relinked with PC-DCE ADK to be compatible with PC-DCE.

Developers who previously used Compaq's IDL compiler to generate stubs or header files may notice that PC-DCE's IDL compiler assigns a different calling convention to the function prototypes. PC-DCE's IDL compiler defaults the convention as being **__cdecl** whereas Compaq's IDL compiler defaults it to **__stdcall**. This may lead to some unresolved function prototype issues when applications are migrated to a PC-DCE development environment.

PC-DCE uses the **-mgr_call_seq** and **-ptr_call_seq** command line options to enable a developer to choose calling conventions. To force the call convention to be a **__stdcall** only, call the PC-DCE IDL compiler with the following parameters:

idl.exe -ptr_call_seq __stdcall -mgr_call_seq __stdcall

3.3 Migrating 16 bit Windows Applications To 32 bit Windows

Please keep the following in mind when porting your existing PC-DCE for Windows 3.1 (16-bit) applications to PC-DCE (32-bit):

- Unlike 16-bit versions of PC-DCE (PC-DCE for Windows 3.1), PC-DCE (Windows 2000, Windows NT, and Windows 98 supported) does not employ a blocking function. To execute asynchronous RPCs, you must use pthreads as demonstrated in the Greet sample application. Be aware that use of the Winsock blocking function is not supported by threads making DCE RPCs.
- To conserve resources, the default stack size in 16-bit PC-DCE for Windows 3.1 is set to 12 KB. This value, however, commonly needs to be increased to accommodate many DCE applications, especially those that issue a lot of Security calls. In PC-DCE, the default stack size is set to 32 KB. In most cases, you should no longer need to increase the stack size.
- Remove all the Entegrity-specific DCE API extension calls such as **DCEGetLastError**, **DCERegisterServerName**, **DCESetBlockingHook**, **DCETaskDestroy**, **DCETaskInit**, **DCEUnhookBlockingHook**, and **DCEYield**.
- Change all of the header files to their full names since they were abbreviated in Windows 3.1. Please refer to Appendix C of your *PC-DCE for Windows 3.1 Application Developer's Guide* for translations.
- The **sec_login_certify_identity** call is supported as long as the user is running with **dced**.
- The client and server stubs have the **_cstub** and **_sstub** extensions, respectively (instead of the **cs** and **ss** extensions). All IDL files will need to be recompiled using the new 32-bit IDL compiler.
- Remote Procedure Calls (RPCs) are not limited to 64K.
- On Windows 2000 and Windows NT, the local file system may be considered secure if you are using NTFS.
- Memory models are irrelevant.
- If you were using **pthread_yield** or **DCEYield** to maintain multitasking ability, these are no longer necessary.

- Unlike in Windows 3.1, threads in PC-DCE cannot freely look at the Windows message queue of any other thread or the main process, nor can they automatically share resources from other processes. Please see the **AttachThreadInput** Win32 API call for a workaround to this issue.

3.4 Migrating Applications Developed Using Earlier Releases of PC-DCE

- All PC-DCE programs you may have developed with pre-3.1 releases should run without any problems on PC-DCE Version 3.1 Client Runtime Kit.
- A version 3.1-compiled program may not work on previous PC-DCE releases; Entegrity does not support forward compatibility.
- Regarding DTS: If no external time provider is in use in the cell, all times will have infinite inaccuracy.
- ONC RPC Interoperability: ONC RPCs and DCE RPCs will neither affect nor interact with each other.

3.4.1 Mutexes

PC-DCE fully supports recursive (**MUTEX_RECURSIVE_NP**), nonrecursive (**MUTEX_NONRECURSIVE_NP**) and fast (**MUTEX_FAST_NP**) mutexes, with fast mutexes set as the default. As a result, any code from a previous PC-DCE release that tries to lock a mutex more than once in the same thread should be rewritten to unlock the mutex before it tries to relock it, unless the recursive mutex is explicitly selected.

The **pthread_mutexattr_getkind_np()** and **pthread_mutexattr_setkind_np()** routines are also supported, allowing you to set and retrieve the mutex kind attribute for recursive, nonrecursive, and fast mutexes.

CHAPTER 4

Miscellaneous Notes

4

This chapter contains miscellaneous information useful when developing DCE applications with PC-DCE. It contains the following sections:

4.1 Developing C++ Applications
4.2 DCE Message Catalogs
4.3 Static Initialization of Constants
4.4 Microsoft RPC Interoperability
4.5 Name Service Interface Gateway
4.6 DCECP Functionality
4.7 .DBG Files
4.8 Max Call Requests
4.9 External Time Providers
4.10 Controlling RPC Timeouts
4.11 Setting a Session-Wide Login Context
4.12 Converting an Exception to an Error Status Code
4.13 Starting PC-DCE Services from the Command Line
4.14 Developing Thread-Safe Code
4.15 Setting Thread Priority Under Windows
4.16 Using Per-Thread Login Contexts
4.17 Using DCE Pipes
4.18 Runtime Version Requirement

4.1 Developing C++ Applications

In order to use the XIDL/DCE IDL compiler for C++ development, you need to have a C++ environment available on a machine where the PC-DCE ADK is installed. Currently, Microsoft Visual C++ Version 4.1 or later is the only supported environment. For portability reasons, stubs generated in this environment and the runtime library (**libidlcxx.lib**) used in this environment do not use features such as templates, exceptions, and multiple inheritance.

4.1.1 Typical Development Procedure

The following steps represent a typical development flow while using XIDL/ DCE IDL C++ support:

1 Create an interface definition using IDL.

2 Identify operations that create remote objects in the IDL file with the **cxx_new** attribute in the attribute configuration (**.acf**) file.

3 Invoke the IDL compiler with the **-lang cxx** option to generate the C++ header and the client/server stubs.

4 For the client, develop the code that uses the class generated. The client can use the static member function **<class>::bind()** to bind to any named or exported remote object. The client can also optionally set rebinding policy on communication failures with remote objects using the **<class>::SetRebind()** static member.

5 For the server, develop the implementation of the class and initialization code that performs the RPC setup and desired object registration (using the **<class>::register_named_object()** static member function).

6 Link application C++ code and the stubs into an executable file.

XIDL/DCE IDL also supports passing of C++ objects as RPC parameters. Applications may use the represent_as parameter attribute to pass in the object or the **cxx_delegate** attribute on the interface definition to pass the object by reference.

By default, XIDL/DCE IDL with the **-lang cxx** option generates a manager class from which the manager implementation may be derived. This can be prevented with the **-no_cxxmgr** option for a manager implementation in the C language.

4.1.2 Pthread Exceptions vs. MS Visual C++

When the pthread header files are included by C++ code, the exception handling macros (**CATCH**, **TRY**, **ENDTRY**, etc.) are renamed with the prefix **DCE_** (**DCE_CATCH**, **DCE_TRY**, **DCE_ENDTRY**, etc.) by default. If your C++ code expects to use the original macro names, you must define the **STANDARD_DCE_EXCEPTIONS** macro either by using the **CL** command line option **-DSTANDARD_DCE_EXCEPTIONS** or by adding the **#define STANDARD_DCE_EXCEPTIONS** statement to the C++ source code prior to including **<pthread.h>** or **<dce/exc_handling.h>**.

Be aware, however, that the DCE and C++ exception handlers are not compatible.

The DCE default names are always used when compiling a C application.

4.1.3 Calling Conventions vs. MS Visual C++

Since there are a number of different function-calling conventions (call sequences) available in C and C++, there are some issues when interfacing between user-written code and IDL-generated code. In particular, functions that are called through the server manager interface are declared with the **IDL_MGR_CALL_SEQ** macro which is defined as **__cdecl** by default. This is the compiler's default for functions written in C.

This macro may be redefined on the CL command line in situations where **__cdecl** is inappropriate or unusable. For example, to specify the standard call convention, you can add the compiler option **-DIDL_MGR_CALL_SEQ=__stdcall**. To use the default C++ calling convention (also known as "thiscall"), define the macro as **-DIDL_MGR_CALL_SEQ=""** (i.e., nothing). Such a routine will not be callable from outside C++.

When generating new code, Entegrity recommends that you use the **IDL_MGR_CALL_SEQ** macro wherever classes are derived from IDL-generated manager classes.

4.2 DCE Message Catalogs

DCE message catalogs are located in:

*install_directory***opt\dcelocal\nls\msg\en_US.ISO8859-1**

4.2.1 Generating Message Catalogs from .sams Files

Message catalogs (**.cat**) generated for application-defined **.sams** (symbol and message strings) files should be compatible with PC-DCE. Currently, the PC-DCE **sams** utility does not support the **-c** command line option, which generates the message catalogs.

In order to generate the necessary message catalogs:

1. Use MSVC 2.2 or later.
2. Generate the message file (**.msg**) from the **.sams** file using **sams** with the **-m** command line option:

   ```
   sams -m app.sams
   ```

3. Use the PC-DCE **strtbl.exe** to generate a resource (**.rc**) file with definitions for the message strings:

   ```
   strtbl app_svc.rc dceapp.msg
   ```

4. Compile the generated resource file using the resource compiler.
5. Build a DLL that includes the compiled resources with a dummy entry point and name it **dceapp.cat**.

4.2.2 NT_EVENTLOG Routing

On Windows NT systems, applications may direct serviceability messages to the NT event log by specifying the NT_EVENTLOG route. Select this route by placing the keyword NT_EVENTLOG in the routing specification. For example:

```
FATAL:NT_EVENTLOG
```

This example directs all fatal DCE serviceability messages to the NT event log.

See *The Open Group DCE 1.1 Application Development Guide* for more information on routing and DCE serviceability.

4.3 Static Initialization of Constants

The PC-DCE runtime DLL exports several DCE constants which cannot be included in static initialization. For example, if an application wants to import the **rpc_x_comm_failure** exception into an array, the following is invalid:

```
static EXCEPTION *pExcList[]={&rpc_x_comm_failure,...};
```

The same can be initialized as:

```
pExcList[0]=&rpc_x_com_failure ;
```

4.4 Microsoft RPC Interoperability

Since Microsoft uses a "DCE-compliant" Remote Procedure Call (RPC) layer, PC-DCE can use Microsoft RPC DCE clients and servers as long as no security or CDS namespace operations are also being used. However, stub files are not compatible. You must use the MIDL compiler to generate a Microsoft RPC program and the PC-DCE IDL compiler to generate a DCE RPC program.

If you want Microsoft RPC Clients to read entries from the CDS namespace, set up an NSID in your cell. This does not let Microsoft RPC servers write to the CDS namespace.

4.5 Name Service Interface Gateway

The Name Service Interface Gateway (**nsid**), allows remote systems that only have RPC services to use the DCE CDS name service. The **nsid** runs on one or more DCE systems in the cell and acts on behalf of the remote system to execute the RPC name service API calls. Through a hidden level of indirection, the **nsid** allows the PC to appear as if it is directly involved in the broader cell namespace.

4.5.1 Supported nsid Protocols

The **nsid** is currently designed to listen on TCP/IP and DECnet. It checks to ensure that each protocol sequence is supported before using it. If neither TCP/IP or DECnet are supported, then the **nsid** will exit with an error message to that effect.

4.5.2 How the nsid Works

An application on a PC running Microsoft Windows makes a call to familiar name-service procedures, such as **rpc_ns_binding_export** (or, in the Microsoft native format, **RpcNsBindingExport**). Within these procedures, the parameters are passed using RPC to the **nsid**.

The **nsid**:

1 Receives the parameters from the PC

2 Converts them to native DCE format

3 Makes a call to the native rpc_ns_binding_export procedure that corresponds to the procedure called on the PC

The CDS server:

1 Receives the parameters

2 Performs the requested operation

3 Returns the results to the **nsid**

The **nsid**:

1 Converts the results back into a format the PC caller can understand

2 Returns them to the PC using RPC

The PC client now has the results of the call and can take appropriate action.

The system where the **nsid** is running can be a different system from where the CDS server is running, because any operation defined by the NSI can be called from any member of the cell.

4.5.3 How the PC Client Finds the nsid

The PC client queries information contained in a database called the registry. On Windows 98, the registry is in a file in the PC root directory, called RPCREG.DAT. In RPCREG.DAT, the first four parameters completely define how the **nsid** is to be contacted, and consist of:

- the protocol sequence (DECnet or TCP/IP, represented as ncacn_dnet_nsp or ncacn_ip_tcp, respectively),
- network address (TCP/IP host name or DECnet node name)
- endpoint (TCP/IP port number or DECnet object name -should be specified as null for TCP/IP. For DECnet, use rpccp to determine the dynamic endpoint used, and specify this endpoint)
- default name syntax (should be 0)

For example, within the procedure RpcNsBindingLookupBegin RPCREG.DAT is read and a remote procedure call is made to the binding represented by that set of parameters.

On Windows NT and Windows 2000, the Registry is an integral part of the operating system's services.

For details on how to set up the registry for Windows NT and Windows 2000, see Chapter 5 of the *PC-DCE Adminstrator's Guide*

4.5.4 APIs Supported by the nsid

All of the NSI is supported by the **nsid**, but not on all platforms. For instance, because Windows provides only client RPC services, server-only APIs are not supported.

The following list of APIs is in the DCE style. The native Microsoft style is slightly different. For example, **rpc_ns_group_delete** appears as **RpcNsGroupDelete** in the Microsoft native style. A dceport.h INCLUDE file is provided to enhance application portability on Microsoft platforms.

rpc_ns_binding_import_begin
rpc_ns_binding_import_next
rpc_ns_binding_import_done
rpc_ns_binding_lookup_begin
rpc_ns_binding_lookup_next
rpc_ns_binding_lookup_done
rpc_ns_binding_export
rpc_ns_binding_unexport
rpc_ns_group_mbr_add
rpc_ns_group_mbr_remove
rpc_ns_group_delete
rpc_ns_mgmt_inq_exp_age
rpc_ns_mgmt_set_exp_age
rpc_ns_profile_elt_add
rpc_ns_profile_elt_remove
rpc_ns_profile_delete
rpc_ns_mgmt_entry_create
rpc_ns_mgmt_entry_delete
rpc_ns_mgmt_entry_inq_if_ids
rpc_ns_mgmt_binding_unexport
rpc_ns_entry_expand_name
rpc_ns_group_mbr_inq_begin
rpc_ns_group_mbr_inq_next
rpc_ns_group_mbr_inq_done
rpc_ns_profile_elt_inq_begin
rpc_ns_profile_elt_inq_next
rpc_ns_profile_elt_inq_done
rpc_ns_entry_object_inq_begin
rpc_ns_entry_object_inq_next
rpc_ns_entry_object_inq_done

4.6 DCECP Functionality

The DCE Control program (which is a **tcl** interpreter extended with DCE-specific commands) may be embedded in applications that are themselves **tcl** interpreters. When embedding **dcecp** functionality into an application, you must include the **tcl.h** and **dcecp.h** header files. These files are shipped as part of the ADK.

In addition, **dcecp** and **tcl** are shipped as DLLs rather than static libraries in order to save paging space. **dcecp.exe** is a thin veneer above **dcecp.tcl** (lib), which simply initializes the interpreter and starts the **tcl** command loop.

4.7 .DBG Files

Symbol files corresponding to all DCE executables (including those in both the runtime and server kits) are installed under the *install_directory*/**bin/symbols** directory. In order to debug PC-DCE (optimized) executables, these symbol files need to be copied into the default symbol directory (for example, **windows_directory/symbols**) thus preserving the directory structure. This step is not necessary if **windbg** is being used for debugging and the **-y** command line option is used to specify the symbol path; **msdev** has no such option and requires the **.dbg** files in the default location.

After the debug files are installed, you can set breakpoints inside any function within PC-DCE executables and DLLs by editing the breakpoints and specifying the name of the function to break in. For functions that use the **__stdcall** convention (as most of the functions exported from **dce32.dll** do), function calls need to be suffixed by *@size of arguments*. For example, if you are setting a breakpoint in the function **dce_msg_define_msg_table** which takes a pointer, unsigned32 and a pointer as arguments, you should specify the function to break in as **_dce_msg_define_msg_table@12**.

Once the debugger stops at the breakpoint of interest, you can now examine registers, call stacks and variables.

4.8 Max Call Requests

OSF DCE v1.2.2 had a limitation where if you specified a max call request argument with certain RPC functions, the argument was ignored and the default max call request located in the header file was used instead. This limitation affected the following functions:

- rpc_server_use_all_protseqs
- rpc_server_use_all_protseqs_if
- rpc_server_use_protseq
- rpc_server_use_protseq_ep
- rpc_server_use_protseq_if

PC-DCE lets you specify a max call request with these functions that is greater than the default, enabling more endpoints upon which the server can listen for client requests.This can help reduce the possibility of a server rejecting a client call due to the socket queue becoming full.

4.9 External Time Providers

Entegrity does not provide a DTS provider implementation. However, Entegrity does support the DTS provider interface so that you can add your own time provider into DTS. The **dtsprovider.idl** file is shipped with the PC-DCE Application Developer’s Kit (ADK). For information on implementing time providers, refer to the *OSF DCE Application Development Guide*.

4.10 Controlling RPC Timeouts

In certain situations, you may want to exercise more control over the timeout value of RPCs that regularly fail or generally do a better job of handling unexpected RPC failures. The timeout value for connection-oriented services (such as TCP) cannot be controlled at the API level because that attribute value is determined by the underlying transport layer. However, the datagram service (UDP) works differently. Default timeouts are 2 minutes for TCP and 45 seconds for UDP.

You can use the DCE API **rpc_mgmt_set_com_timeout()** to define the timeout for the datagram protocol. While TCP is often the protocol of choice (because it offers guaranteed delivery), some of your specific RPCs or your test environment might want to use UDP to take advantage of the control it offers in this area. With PC-DCE for Windows 3.1 and PC-DCE, you can tune the timeout from a few seconds to 30 seconds, to several minutes and longer.

After your client application gets a binding, make your call. For example:

rpc_mgmt_set_com_timeout(binding_h, 0, &status);

where a value of 0 represents the minimum timeout and a value of 10 attempts to communicate forever. Although the values in between do not represent actual seconds, they are instead arbitrarily assigned and presented in ascending order; each is longer than the previous value.

You can have your client code selectively secure a UDP binding or use the **RPC_SUPPORTED_PROTSEQS** environment variable to restrict the runtime to one protocol or the other, for example: **RPC_SUPPORT_PROTSEQS=ncadg_ip_udp**.

4.11 Setting a Session-Wide Login Context

If your application uses the standard DCE **sec_login** functions, it will create a login context that applies only to that specific instance of the application. If you run such an application, open a command prompt window and issue the **klist** command, it will not list the application's login context.

The PC-DCE **dce_login** program sets a session-wide login context. If you open a command prompt, run **dce_login**, close the command prompt, and then open a second command prompt and run **klist**, it will list the credentials created by **dce_login**. You can create such a session-wide login context for your application by using the following functions:

- **sec_login_set_default_context()**
- **sec_login_setup_win32_context()**

For more information on these functions, refer to *Chapter 5 on page 31*.

4.12 Converting an Exception to an Error Status Code

Some programs may cause an unhandled exception to be raised by the DCE runtime. Symptoms include:

- Application produces an abnormal program termination, *Runtime Error!* with a dialog box titled *Microsoft Visual C++ Runtime Library*
- Access violation
- Application hangs

You can trap the exception that is being raised and convert it to an error message. At that point you can change the application or at least handle similar failures better. Here's an example of how to CATCH an exception for the RPC made in the Greet sample.

```
TRY
{
    for (i = 0; i < reps; i++)
    {
        greet (binding_h, greeting, reply);
        printf ("The server says: %s\n", (LPSTR) reply);
        pthread_yield ();
    }
}

CATCH_ALL
{
exc_get_status (THIS_CATCH, &status);
ERROR_CHECK (status, "This was caught during the RPC");
}
```

4.13 Starting PC-DCE Services from the Command Line

PC-DCE integrates all DCE services directly into the Windows 2000 and Windows NT services layer. Typically, you manage DCE services from the PC-DCE Service Panel. For debugging purposes only, you can start the DCE services manually from the command line. Use the commands in the following list to start DCE manually. This list is complete and is presented in the order that the services would start if activated automatically.

For the lightweight client, there are no daemons to start with the possible exception of **rpcd** if the client is so configured.

For a full client:

```
prompt> rpcd -d -f
prompt> sec_clientd -d -v
prompt> cdsadv -v
```

For a server:

```
prompt> rpcd -d -f
prompt> sec_clientd -d -v
prompt> secd -v
prompt> cdsd -v
prompt> cdsadv -v
```

4.14 Developing Thread-Safe Code

Symptoms that you may see if your server manager code is not thread-safe include:

- With shared variables, it is possible for each thread to use the variable and set it to the correct value for that thread. If the application is not thread-safe, the value meant for one thread could be passed to another thread, causing incorrect results.
- When you are debugging your server code, check the values of shared variables. If you are not seeing the expected value, the server manager code may not be thread safe.

Sometimes the RPC server manager code includes operations such as file I/O and database update routines that are not thread-safe. To use DCE **pthread mutex lock** to make sure the routines are not executed by more than one server thread:

- In the server initialization code, call **pthread_mutex_init()** before calling **rpc_server_listen()**.
- In your RPC manager code, wrap the appropriate routines with **pthread_mutex_lock()** and **pthread_mutex_unlock()** to mutex-protect them.

  This ensures that only one thread at a time performs the thread-unsafe routines.

4.15 Setting Thread Priority Under Windows

Currently, **pthread_attr_setpri()** takes priority values from 1-15 where levels 1-9 map to Above Normal priority and levels 10-15 map to Highest priority. Note that the Windows 2000 and Windows NT environment makes no distinction between values within each range. For example, a one has the same priority as nine (Above Normal). At this time, Entegrity does not allow any priority to go below normal.

To set the base priority of a thread below normal, you can call the Win32 API **SetThreadPriority** function. The following priority levels are available using this function: Lowest, Below Normal, Normal, Above Normal, Highest.

4.16 Using Per-Thread Login Contexts

Standard DCE allows you to have as many login contexts as you wish — you supply the login context handle when annotating an RPC binding handle with security or when negotiating a GSS-API session. However, standard DCE lets you set only one process-wide default login context (using the **sec_login_set_context()** call).

PC-DCE enhances standard DCE to allow default login contexts on a per-thread basis. Your application calls **sec_login_set_thread_context()** to set up the per-thread context. Then, calling **sec_login_get_current_context()**

from that thread returns the per-thread context rather than the per-process context. To break the association between a thread and a login context you call **sec_login_unset_thread_context()**.

One example of the usefulness of per-thread login contexts is when you can not pass around a DCE login context handle within your application. This happens many times when you are calling across a standardized interface, such as ODBC. ODBC has no place in the interface to pass down a DCE login context handle to an underlying DCE-enabled ODBC driver, so these drivers simply call **sec_login_get_current_context()** to get the per-process login context. This works fine in a 2-tier environment, but breaks down in a 3-tier delegated environment.

In the 3-tier environment, the intermediate server has one login context for each client that is accessing it, reflecting the fact that the server is acting on behalf of the client. Without per-thread login contexts, you would need to mutex-protect the per-process login context and single-thread each user through the ODBC layer. With per-thread login contexts, you associate the delegated login context with the thread, make your ODBC calls, and then break the association.

For details on the Entegrity extensions, refer to *Chapter 5 on page 31*.

4.17 Using DCE Pipes

Consider using DCE pipes in situations where:

- You are not sure how much data you have to handle.
- You want to deal with the data (display, process, etc.) without having to wait for the full transfer to complete.
- You expect large data transfers (an approximate minimum of half a megabyte).
- You need to perform large data transfers of a common type.

The *Guide to Writing DCE Applications* by Shirley, Wu, and Magid (O'Reilly & Associates) discusses this topic in detail. Entegrity has ported the Transfer_data sample to PC-DCE. You can download this sample from the following location: **ftp://ftp.entegrity.com/pub/dce/pcdce/samples/ xferdata.zip.**

4.18 Runtime Version Requirement

If you build applications using the current PC-DCE Application Developer's Kit, you must run them with the PC-DCE runtime at the current revision or later.

CHAPTER 5

Entegrity API Extensions

5

This chapter provides reference information on the following Entegrity OSF DCE version 1.2.2 API:

- 5.1 sec_login_set_default_context()
- 5.2 sec_login_setup_win32_context()
- 5.3 sec_login_set_thread_context()
- 5.4 sec_login_unset_thread_context()
- 5.5 sec_login_get_default_context()

5.1 sec_login_set_default_context()

Purpose

Sets session-wide credentials for a login context.

Definition

```
#include <dce/sec_login.h>

boolean32 sec_login_set_default_context (
     sec_login_handle_t     login_context,
     error_status_t         *status
)
```

Input Parameters

login_context — An opaque handle to login context data.

Output Parameters

status — A pointer to the completion status.

Description

Sets the credentials as the default credentials for this Windows login session. This is equivalent to how **dce_login.exe** or the integrated login sets session-wide credentials. This is useful when developing a login program or application that has to set session-wide credentials.

This context must have been previously validated.

Use care when using this routine. Applications setting session-wide credentials can impact other running applications.

If this function fails it returns *false*. For more specific error information refer to the contents of the **status** parameter.

Related Functions

sec_login_setup_win32_context()

5.2 sec_login_setup_win32_context()

Purpose

Sets session-wide credentials within a network provider or GINA DLL.

Definition

```
#include <dce/sec_login.h>

boolean32 sec_login_setup_win32_context (
     sec_login_handle_t     login_context,
     idl_char               *username,
     error_status_t         *status
)
```

Input Parameters

login_context — An opaque handle to login context data.

username — Character string containing the WIN32 username of the user. If the user is a member of a Windows domain this must be in the format *domain\\username*.

Output Parameters

status — A pointer to the completion status.

Description

Sets the credentials as the default credentials for the login session being established. This routine must be used by network providers or GINA implementations.

This routine should not be used by a traditional DCE login program. This routine will properly set up the credentials pointer in the Windows registry so that the security runtime can access the credentials.

If this function fails it returns *false*. For more specific error information refer to the contents of the **status** parameter.

Related Functions

sec_login_set_default_context()

5.3 sec_login_set_thread_context()

Purpose

Makes the specified login context the per-thread login context for the calling thread.

Definition

```
#include <dce/sec_login.h>

void __stdcall sec_login_set_thread_context (

    sec_login_handle_t      login_context,
    unsigned32              flags,
    error_status_t          *st
)
```

Input Parameters

login_context — An opaque handle to login context data.

flags — Flags for this call. Defined flags currently include:

(1) sec_login_thread_purgeonexit — When the thread exits, the login context is released.

Output Parameters

st — A pointer to the completion status.

Description

PC-DCE enhances standard DCE to allow default login contexts on a per-thread basis. Your application calls **sec_login_set_thread_context()** to set up the per-thread context. To break the association between a thread and a login context you call **sec_login_unset_thread_context()**.

If this function fails it returns *false*. For more specific error information refer to the contents of the **status** parameter.

For a discussion of per-thread login contexts, refer to *Section 4.16 on page 28*.

Related Functions

sec_login_unset_thread_context()
sec_login_get_default_context()

5.4 sec_login_unset_thread_context()

Purpose

Unsets the current per-thread login context.

Definition

```
#include <dce/sec_login.h>

void __stdcall sec_login_set_thread_context ()
```

Description

PC-DCE enhances standard DCE to allow default login contexts on a per-thread basis. Your application calls **sec_login_set_thread_context()** to set up the per-thread context. To break the association between a thread and a login context you call **sec_login_unset_thread_context()**. This call removes the per-thread context information and releases a reference to the current thread login context. This has no effect on externally-visible credentials.

If this function fails it returns *false*.

For a more detailed discussion of per-thread login contexts, refer to *Section 4.16 on page 28*.

Related Functions

sec_login_set_thread_context()
sec_login_get_default_context()

5.5 sec_login_get_default_context()

Purpose

Gets the default login context. This function allows the caller to specify whether to get the per-thread or per-process login context.

Definition

```
#include <dce/sec_login.h>

void __stdcall sec_login_get_default_context (
     unsigned32             type,
     sec_login_handle_t     *login_context,
     error_status_t         *st
)
```

Input Parameters

type — Identifies the type of login context to get. Defined types currently include:

(0) sec_login_default_context_proc — Gets the per-process login context.

(1) sec_login_default_context_thr — Gets the per-thread login context.

Output Parameters

login_context — A pointer to the requested login context data.

status — A pointer to the completion status.

Description

PC-DCE enhances standard DCE to allow default login contexts on a per-thread basis. Your application calls **sec_login_set_thread_context()** to set up the per-thread context. Once the per-thread context is set up, you can call **sec_login_get_default_context()** to obtain either the process login context or thread login context.

You can also use **sec_login get_current_context()**, which is modified from standard DCE to get the thread login context (if available). Otherwise get the process-wide login context.

If this function fails it returns *false*. For more specific error information refer to the **status** parameter.

For a discussion of per-thread login contexts, refer to *Section 4.16 on page 28*.

Related Functions

sec_login_set_thread_context()
sec_login_unset_thread_context()

CHAPTER 6

Samples

6

6.1 Using the Sample Applications

PC-DCE provides the following sample programs to introduce you to DCE programming and to test the integrity of your PC-DCE installation and configuration:

- Greet Application (*Section 6.2.1*)
- Grade Application (*Section 6.2.2*)

6.2 Greet Application

The Greet sample application consists of the following programs:

- **client.exe** — Greet Client program
- **server.exe** — Greet Server program

The Greet application is implemented as a 32-bit Windows application. The executable files for the Greet application (**client.exe** and **server.exe**) are located in *dce_install_directory***samples****greet** and appear as icons (Greet Client and Greet Server) in the PC-DCE program group. You can run these programs after installation and configuration, or at any time to verify that PC-DCE is operating properly.

The Greet application provides a simple way for you to demonstrate how **pthreads**, CDS, and RPC can be used to accomplish the following tasks:

- Advertise the existence of a remote server to a client
- Locate a remote server
- Establish a connection between a client and server
- Exchange a handshake (or greeting message) between a client and server

You can run the Greet client and server programs on the same system, or run one of the programs on your local PC and the other program on a remote system.

6.2.1 The Greet Client

The Greet client program supports three different methods of locating the Greet server:

- Use string binding

  Locates a Greet server by specifying the exact network location of the target server using a DNS name or an IP address.

- Import bindings from CDS

  Locates the server(s) registered in the CDS namespace via the cdsclerk process in the DCE DLL.

- Use local server

  Locates the server by communicating with **dced** and restricting the search to the local system.

Entegrity suggests that you try running the Greet client using the string binding first, then local server, and finally by looking up server locations in the CDS namespace.

6.2.2 The Greet Server

The Greet server program advertises its existence to **dced** only, or to both **dced** and CDS. Registration with **dced** only allows the Greet client to establish a connection by using string bindings or a local server. However, registration with CDS enables the Greet client to locate the server without knowing the location of the system where the Greet server is running. Again, Entegrity suggests that you try the simple case first by disabling registration of the Greet server's bindings in the CDS namespace.

6.2.3 Running the Greet Sample Programs

1. For the Greet application to work properly, dced and the CDS Advertiser (cdsadv) must be running on your system. If in doubt, select the PC-DCE icon in the Windows Control Panel to activate the PC-DCE Service Panel and verify that the Remote Procedure Call daemon and the CDS Advertiser are running.
2. If not already logged in, DCE_login to your cell, using a valid principal and password.

   To log in, open an MS-DOS window and type: `dce_login`. Then enter your username and password.
3. Select the Greet Server icon from the PC-DCE program group to start the Greet server program. The Greet Server dialog appears. The Server name field displays the predefined CDS name for your Greet server (/.:/hosts/*yourhostname*/greet-server) that was created as part of the PC-DCE configuration on your system.

4. In the Threads dialog box, specify how many threads you want the Greet server to use when listening for Greet client communications.
5. By default, the checkbox Export Server Binding into CDS is marked with an X to allow Greet clients to establish a connection with the server by performing a name lookup in the CDS namespace. If you want Greet clients to connect with their local Greet server, or to establish a connection with this server only by using a string binding, deselect the checkbox Export Server Binding into CDS.
6. When satisfied with your settings, click Start. The Greet server responds by registering itself with **dced** and with CDS (if you accepted the default in the checkbox Export Server Bindings into CDS in step 5).

   The RPC registration process may take a few moments. When the server is ready to receive client calls, it prints `Listening...` in the Output: box.
7. Select the Greet Client icon from the PC-DCE program group to start the Greet client program. The Greet Client Sample Program dialog appears.
8. Click the appropriate radio button to select the method by which the Greet Client will locate a Greet Server:
   - Select Use Local Server to restrict a search for a Greet server by dced to your local system.
   - Select Import Bindings from CDS to instruct the client to look up the location of the Greet server in the CDS namespace.
   - Select Use String Binding to Server to locate the server by the information provided in the string binding.
9. In the Message To Send field, enter the message that you want to send to the Greet server. The message `Hello Server` is specified by default.
10. In the Repetitions field, enter the number of times you want the Greet client to send the message that you specified. By default, the client repeats the message 5 times.
11. When satisfied with your settings, click Send. The client imports the server's binding handle (if CDS is selected) and exchanges messages with the server. The server prints the client's greeting message and the client prints the server's default reply `Hi client!`.

NOTE: If you directed the Greet client to look up the location of a Greet server in the CDS namespace and if there are multiple Greet servers (of the same name) active in your cell, you will be unable to control with which servers the Greet client actually connects. This is because, based on identical names, the Greet client binds randomly to a Greet server via CDS. You can alter this behavior by using RPC profiles or by specifying unique server names.

To cancel either program, click the Cancel button. When you cancel the Greet server, it automatically unregisters its endpoint.

6.3 Grade Application

The Grade sample application consists of the following programs:

- **client.exe** — Grade Server client program
- **server.exe** — Grade Server server program

The Grade Server application is implemented as a Windows Console application to demonstrate how you can create standard UNIX-style client and server programs in the Win32® environment and avoid the overhead of creating a Windows GUI. The executable files for the Grade Server application (**client.exe** and **server.exe**) are located in *dce_install_directory* **\samples\grade**.

The Grade Server programs shipped with PC-DCE are implemented as Windows Console applications. They are based on the UNIX Grade Server application that appears in the second edition of the *Guide to Writing DCE Applications* (Shirley, Hu, and Magid) published by O'Reilly & Associates, Inc. Entegrity's adaptations of these programs demonstrate how easy it is to modify an existing DCE application to run under Windows. (Setup instructions for this application were derived, in part, from materials provided by O'Reilly & Associates, Inc.)

The server stores a fixed database of student names and their grade point averages (GPAs). The client queries the database for GPA information and the Grade server responds to the client.

The only interface defined for the server is the GPA interface that is called by the client. This remote procedure returns the GPA associated with individual students.

The Grade Server application illustrates the use of authenticated RPC and DCE Security services, including the following tasks:

- Annotating a binding for authenticated RPC.
- Determining a server's principal name.
- Setting up a server's login context.
- Creating a thread to manage a server's secret key.
- Setting up a reference monitor that uses name and group membership for authorization.

6.3.1 Setting Up the Grade Server Application

The implementation of the Grade Server assumes that the principal identities and groups and the keytab file have been created and that the keytab file can be accessed by the Grade Server process. It also assumes that the server can export to CDS. You perform these operations using the management utilities supplied with PC-DCE. You should perform the Security setup first to create the principal identities required for the CDS operation.

6.3.1.1 Security Setup

The implementation assumes that the Grade server process executes with the local identity of grade_server_1. The server process needs a local identity so that it can access its keytab file.

Before you can run the Grade Server application, you need to use **dcecp** to perform the following tasks:

- Create groups for both the Grade servers and teachers. (Note that the grade_server group is not used by the server; it is used by clients to authenticate the server.)
- Create principals for the Grade server, students, and teachers.
- Create accounts for each of the three principals.
- Create a keytab file for the server.

1. Invoke the DCE Login utility and enter the following command to log in as the cell administrator:

   ```
   dce_login cell_admin cell_admin_password
   ```

2. Invoke **dcecp** and enter the following commands to create the grade_server, student, and teacher groups:

   ```
   dcecp> group create grade_server

   dcecp> group create teacher
   ```

3. Create the grade_server_1, student_1, and teacher_1 principals.

   ```
   dcecp> principal create grade_server_1

   dcecp> principal create student_1

   dcecp> principal create teacher_1
   ```

4. Add the grade_server_1, student_1, and teacher_1 principals to their appropriate groups:

   ```
   dcecp> group add grade_server -m grade_server_1

   dcecp> group add none -m student_1

   dcecp> group add teacher -m teacher_1
   ```

5. Add the grade_server_1, student_1, and teacher_1 principals to their appropriate organizations:

   ```
   dcecp> org add none -m grade_server_1

   dcecp> org add none -m student_1

   dcecp> org add none -m teacher_1
   ```

6. Create accounts for the three principals. Make grade_server_1 a member of the grade_server group, student_1 a member of the student group, and teacher_1 a member of the teacher group.

   ```
   dcecp> account create grade_server_1 -group grade_server -o none -pa
   -dce- -my cell_admin_password
   ```

```
dcecp> account create student_1 -group none -o none -pa -dce- -my
cell_admin_password
```

```
dcecp> account create teacher_1 -group teacher -o none -pa -dce- -my
cell_admin_password
```

7 Create a keytab file for the grade_server_1 principal. The following command must be executed on the system where you intend to run the Grade Server program. In addition, replace *server_host_name* below with the name of your host server. Note that you must also create the **/tmp** directory if it doesn't already exist.

```
dcecp> keytab create \
/.:/hosts/server_host_name/config/keytab/grade_server_1 -storage \
"/tmp/grade_server_tab" -data {grade_server_1 plain 1 -dce-}
```

6.3.1.2 CDS Setup

Perform the following steps to create a CDS object (**/.:/grade**) and modify its ACL:

1 Enter the following command to create the object in CDS. This is the entry to which the Grade Server exports its bindings.

```
dcecp> obj create /.:/grade
```

2 Modify the ACL to grant read and write access to the grade_server_1 principal.

```
dcecp> acl modify -e /.:/grade -add {user grade_server_1 rw}
```

3 Issue the following command to verify that the grade_server_1 principal is granted the correct permissions.

```
dcecp> acl show -e /.:/grade -entry
```

```
{unauthenticated:r--t-}
{user:cell_admin:rwdtc}
{user:grade_server_1:rw---}
{group:subsys/dce/cds-admin:rwdtc}
{group:subsys/dce/cds-server:rwdtc}
{any_other:r--t-}
```

```
dcecp> exit
```

6.3.2 Running the Grade Server Sample Programs

At this point, you are ready to run the application. Begin by starting up the Grade Server. Because the server will establish its own DCE identity, the process that starts the Grade Server does not need to log into DCE.

6.3.2.1 Running the Grade Server Program

To start the Grade Server, change directories to *dce_install_directory* **\samples\grade** and enter the **server** command.

6.3.2.2 Running the Grade Client Program

1 Invoke the DCElogin utility and DCElogin as the student_1 principal. If you’ve been following the instructions presented above, keep in mind that the assigned password is **-dce-**.

2 Change directories to *dce_install_directory***samples****grade**. To start the Grade Client and display the GPA for student_1, enter **`client student_1`**. The GPA for student_1 is displayed.

3 Now enter **`client Peter`** to attempt to retrieve the GPA for the student Peter. Because you are logged in as student_1, and you are not a teacher, your query is not permitted and the server returns a GPA of -1.00000.

4 Repeat step 1, but this time, login as the teacher_1 principal. Then repeat step 2 to prove that the teacher_1 principal is permitted to retrieve the GPA for student_1.

5 Repeat step 3. This time, because you are logged in as teacher_1, you are entitled to see all GPAs in the database and the Grade Server displays Peter's GPA, 4.00000.

Index

A

API extensions 31
Application message service 9
Application programming interfaces 9
Applications
- development procedures 19
- embedding dcecp functionality 24
- login contexts 26
- running with PC-DCE runtime 29
- sample 37

B

Backing store database 9

C

C++ environmental requirements 19
Cell Directory Service 9
Clock synchronization 15
Compaq DCE for Windows NT
- migrating applications 16

Compiling 11
- switches 11

Constants
- initialization 22

Contacting Entegrity Solutions 7

D

DCE
- API 11
- pipes 29
- runtime version requirement 29
- The Open Group documentation 6

DCE control program (dcecp) 24
Debug
- files 24
- using DCE services 27

Development procedures 19
Documentation 5, 6, 7
DTS 25
- clerk 15

E

Error calls 16
Event log
- routing messages to 21

Exception handling 11, 12, 20
- converting to error messages 26

F

FAT file system 16
Freeing memory 11
Function-calling conventions 20

G

GetLastError() 16
Global Directory Service 9
Global unwinding 12
Grade sample application 40
Greet sample application 37

H

Host services 9

I

IDE interface 14
IDL 13, 19
Integrated login 15

L

Linking 11
Login contexts
- session-wide 26

M

Macros
- exception handling 12, 13

Max Call Requests 25
Message catalogs
- generating 21

Migrating applications 15
Mutexes 18

N

Name Service Interface Gateway (nsid) 22
NTFS partitioning 16

O

OLE 13
OSF DCE documentation 9
OSF documentation 6
Overview 9

P

Partitioning 15
PC-DCE
- additional documentation 5
- migrating pre-3.0 applications 18
- Service Panel 27

pthreads
- failed calls 16
- library 15

R

Routing messages to the event log 21
RPC
- functions 25
- interoperability 22
- timeouts 25

RPC services 22

S

Sample applications 37
Serviceability 9
Static initialization 22
Stub generation 13
- C++ environment 19

Support 6, 7

T

Technical support 6, 7
Threads
- per-thread login contexts 28
- prioritization 28
- scheduling 15

Thread-safe code 28
Time services 25
timesync.exe 15

U

UNIX
- migrating applications 15

W

Windows 2000 and Windows NT security 15
Windows 3.1 application
- porting 17 to 18

Windows NT event log
- routing messages to 21

X

X/Open Directory Service 9
X/Open OSI-Abstract-Data Manipulation 9