

LANDP® Family



# Programming Guide

*Version 5.0*



LANDP® Family



# Programming Guide

*Version 5.0*

#### **Note**

Before using this information and the product it supports, be sure to read the general information under Appendix A, "Notices" on page 181.

#### **First Edition (April 2000)**

This edition applies to LANDP Family Version 5 (part number 0781197 in the United States of America, program number 5639-I90 in Europe, the Middle East, and Africa) and to all subsequent releases and modifications, until otherwise indicated in new editions. Make sure you are using the correct edition for the level of product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the addresses given below.

At the back of this publication is a page titled "Sending your comments to IBM". If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories, User Technologies,  
Mail Point 095, Hursley Park, Winchester, Hampshire, England, SO21 2JN.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1992, 2000. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>About this book</b>	xiii
Who should read this book	xiii
What you need to know	xiii
How this book is organized	xiii
Conventions used in this book	xiv
Operating environments	xiv
Function, return, and event codes	xv
Bit positions	xv
DB2 Universal Database®	xv
Windows 2000	xv
Related information	xvi
Web site	xvi
 <b>Summary of Changes</b>	 xvii
 <b>Chapter 1. Clients and servers</b>	 1
Communicating between clients and servers	2
Common API used by clients	4
Common API used by servers	5
Connectivity programming request block (CPRB)	5
Passing parameters	9
Passing data	10
Naming system resources	10
Server names	10
Compiling and linking your application program	13
LANDP for OS/2 programming environments	14
LANDP for OS/2 dynamic link library	14
LANDP for OS/2 program types	15
Include file EHCDEFC.H for C and C++ language programs	16
LANDP for Windows NT programming environments	17
LANDP for Windows NT dynamic link library	17
Include file EHCDEFC.H for C and C++ language programs	17
Migration considerations	18
Moving to the latest LANDP for DOS	18
Moving LANDP for DOS and FBSS (DOS) services to LANDP for OS/2	19
Moving LANDP for DOS and FBSS (DOS) services to LANDP for Windows NT	19
Migrating FBSS (DOS) clients or user servers to the LANDP common API	19
Migrating FBSS/2 16-bit clients and user servers to 32-bit mode	21
Migrating LANDP for DOS and FBSS (DOS) clients to LANDP for OS/2	21
Migrating LANDP for DOS and FBSS (DOS) user servers to LANDP for OS/2	22
Migrating LANDP for DOS and FBSS (DOS) clients to LANDP for Windows NT	22
Migrating LANDP for DOS and FBSS (DOS) user servers to LANDP for Windows NT	23
Migrating from the shared-file server to the LANDP for OS/2 query server	24
Migrating LANDP for OS/2 clients and user servers to LANDP for Windows NT	25

<b>Chapter 2. Writing client programs</b>	27
Invoking the common API	27
CPRB fields used and set by clients	28
Call RMTREQ (Remote request)	28
Call RMTREQ using the NoWait option	29
Call GETRPLY (Get reply)	32
Hints	35
Sample application programs	36
LANDP event notification support	36
Types of event	36
Event ID	37
Receiving event notifications	38
Waiting for multiple events	38
Polling asynchronous event information	39
Event notification using graphical user interface (GUI) message posting	39
LANDP system events	40
LANDP for DOS and Windows 3.1/3.11 support	43
Running standard LANDP for DOS applications	43
Requirements for your standard LANDP for DOS applications	45
Unloading LANDP for DOS	45
Building Windows 3.1 applications that request LANDP for DOS services	45
Performance considerations	46
Restrictions	46
 <b>Chapter 3. Writing your own server programs</b>	 47
Structure of a server	47
Invoking the common API	48
CPRB fields used and set by servers	48
Call SRVINIT (server initialization)	49
Call GETREQ (get request)	51
Call RMTRPLY (remote reply)	54
Call RMTAREQ (remote asynchronous request)	55
Receiving system requests	56
End of service (ES function)	57
Server recognition (IN function)	58
Timer-generated request (TT function)	59
Workstation disconnection (** function)	61
Process disconnection (** function)	61
Workstation connection (&& function)	62
Process connection (&& function)	63
Sending asynchronous events	64
Server-to-server calls	65
Writing LANDP for DOS servers	66
Memory management considerations	66
Interrupt handling	68
Expanded memory considerations	68
Writing LANDP for OS/2 servers	69
Using multiple threads	69

Using wait multiple (WM) in a server . . . . .	70
Writing LANDP for Windows NT servers . . . . .	71
Using multiple threads . . . . .	71
Using wait multiple (WM) in a server . . . . .	72
Writing LANDP for AIX servers . . . . .	73
LANDP for AIX server structure . . . . .	73
Calling SRVINIT . . . . .	75
Reply buffer allocation . . . . .	75
Server child support . . . . .	76
Server events . . . . .	76
<b>Chapter 4. LANDP–DCE application programming interface . . . . .</b>	<b>77</b>
The LANDPDCE.IDL and LANDPDCE.ACF files . . . . .	78
Writing a LANDP–DCE client . . . . .	78
Obtaining a binding handle for LANDP services . . . . .	78
Obtaining a LANDP context . . . . .	80
Requesting LANDP services . . . . .	80
Releasing a LANDP context . . . . .	81
DCE client structure . . . . .	81
Writing a LANDP–DCE server . . . . .	82
Binding . . . . .	82
Providing a LANDP context in the server manager routines . . . . .	83
Providing services to LANDP clients in the server manager routines . . . . .	84
Releasing LANDP context in the server manager routines . . . . .	84
DCE server structure . . . . .	85
<b>Chapter 5. Object-oriented application programming . . . . .</b>	<b>87</b>
Writing application programs using C++ . . . . .	87
LandpRequest class . . . . .	87
RequestFromLandp class . . . . .	89
LandpServer class . . . . .	90
Writing application programs using Smalltalk . . . . .	92
LandpRequest class . . . . .	92
<b>Chapter 6. LANDP support for Java . . . . .</b>	<b>95</b>
Support for Version 4 classes . . . . .	95
VisualAge for Java support . . . . .	95
Java client development . . . . .	96
Support for multiple client applications within a JVM . . . . .	97
Exception handling . . . . .	100
Writing servlets to access LANDP . . . . .	100
Writing applets to access LANDP . . . . .	100
Writing LANDP servers in Java . . . . .	101
<b>Chapter 7. Writing programs using VisualAge for COBOL . . . . .</b>	<b>103</b>
Writing GUI programs under OS/2 . . . . .	103
Writing non-GUI programs . . . . .	103
VisualAge for COBOL compilation settings . . . . .	104

<b>Chapter 8. VisualAge Generator Application Programming Interface</b>	105
Overview	105
The LANDP Dynamic Link Library	105
Calling LANDP servers from VisualAge Generator application programs	106
Calling functions within the DLL	107
Return codes	112
Testing applications	112
Generating an application	112
<b>Chapter 9. LANDP for OS/2 REXX application programming interface</b>	113
<b>Chapter 10. Testing your application programs</b>	117
Defining a specific test	118
Using the keyboard	120
DCZYXSVP test program	121
Testing with Windows 3.1/3.11 or Windows NT	121
Invocation	121
Menu options	122
Parameter and data entry fields	124
Using your own SVPCPRB exit server	124
<b>Chapter 11. Sample application programs</b>	127
Sample application (C, Windows NT)	127
Sample client application LDPCMAIN.C	127
Header files	138
Sample user server LDPSMAIN.C	139
Sample application (COBOL, OS/2 AND Windows NT)	149
Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL	149
Sample server (COBOL, OS/2 and Windows NT) SAMPSESV.CBL	164
Sample client application (COBOL), DOS and OS/2	170
Sample client application program SAMPLECB.CBL	170
Building sample applications	178
Sample client and server, C, Windows NT	178
Sample client and server, COBOL, OS/2 and WINDOWS NT	179
Sample client application, COBOL, DOS and OS/2	179
Running the sample programs	180
<b>Appendix A. Notices</b>	181
Trademarks and service marks	183
<b>Glossary</b>	185
<b>Bibliography</b>	207
IBM LANDP Family	207
IBM Financial Branch System Services Licensed Programs	207
IBM Financial Branch System Integrator Licensed Programs	207
IBM Transaction Security System	207
Banking Self-Service	207



IBM workstations . . . . .	208
IBM RISC System/6000® . . . . .	208
IBM Local Area Network . . . . .	209
IBM 3270 . . . . .	209
Wide Area Communications . . . . .	209
IBM NetView . . . . .	210
IBM Financial I/O Devices . . . . .	210
Distributed Computing Environment . . . . .	211
Encryption and Decryption . . . . .	211
IBM VisualAge C++ . . . . .	211
IBM VisualAge Generator . . . . .	211
IBM VisualAge Smalltalk . . . . .	211
Java . . . . .	211
IBM Personal Communications . . . . .	212
IBM Communications Server . . . . .	212
WorkSpace On-Demand . . . . .	212
MQSeries . . . . .	212
<b>Index . . . . .</b>	<b>213</b>



---

## Figures

1.	General Client/Server Relationship . . . . .	1
2.	DCE Client Accessing LANDP Server . . . . .	77
3.	LANDP Client Accessing DCE Server with LANDP Interface . . . . .	77
4.	LANDP Client Accessing DCE Server with Non-LANDP–DCE Interface . . . . .	77
5.	VisualAge Generator SRPIBLK definition . . . . .	106
6.	Conversion table, ASCII to EBCDIC . . . . .	110
7.	Conversion table, EBCDIC to ASCII . . . . .	111
8.	System verification program - send CPRBs . . . . .	123
9.	Sample Windows application - events and function calls in client and server . . . . .	128
10.	Functions and calls in client and server . . . . .	149



---

## Tables

1.	Files needed for LANDP application program development . . . . .	13
2.	System Request Function Codes, which must be accepted by All Servers .	47
3.	SRVINIT calling syntax . . . . .	49
4.	EHC_SRVINIT_OPTS control block format . . . . .	51
5.	GETREQ syntax . . . . .	51
6.	EHC_GETREQ_OPTS structure . . . . .	53
7.	RMTRPLY syntax. . . . .	54
8.	RMTAREQ syntax . . . . .	56
9.	ES function, CPRB contents . . . . .	57
10.	IN function, CPRB contents . . . . .	58
11.	TT function, CPRB contents . . . . .	60
12.	Workstation disconnection (**) function, CPRB contents . . . . .	61
13.	Process disconnection (**) function, CPRB contents . . . . .	62
14.	Workstation connection (&&) function, CPRB contents . . . . .	63
15.	Process connection (&&) function, CPRB contents . . . . .	64
16.	SRPIBLK contents for HC function . . . . .	107
17.	SRPIBLK for CH function . . . . .	108
18.	SRPIBLK for AE and EA functions . . . . .	109



---

## About this book

This book provides information about the following IBM® LAN Distributed Platform (LANDP®) Family products:

- LANDP Family Version 5.0  
with its components:
  - LANDP for DOS
  - LANDP for OS/2®
  - LANDP for Windows NT
- IBM LANDP for AIX®, Version 2 Release 1.0 (LANDP for AIX)

This book provides guidance on how to design and develop user servers and applications for a LANDP environment.

---

## Who should read this book

This book is written for system programmers and application programmers.

---

## What you need to know

You should be familiar with the operating systems that support your LANDP environment and the programming language you are using.

Also, if you are involved in the development of application programs using wide area communications, you should be familiar with System Network Architecture (SNA) protocols and Synchronous Data Link Control (SDLC), X.25 Data Link Control, or Token-Ring Data Link Control.

---

## How this book is organized

The book contains the following chapters:

- Chapter 1, “Clients and servers” on page 1** This chapter introduces the LANDP family of programs, explains the common application programming interface (API), and tells you how to use it when writing LANDP application programs.
- Chapter 2, “Writing client programs” on page 27** This chapter provides guidance for writing client application programs, using the common API. Client programs can use IBM-supplied LANDP servers or user-written servers.
- Chapter 3, “Writing your own server programs” on page 47** This chapter gives guidance on writing your own servers that provide services that clients can request through the common API.

## About this book

- Chapter 4, “LANDP–DCE application programming interface” on page 77** This chapter describes how to write Distributed Computing Environment (DCE) clients and server applications using the LANDP-DCE application programming interface.
- Chapter 5, “Object-oriented application programming” on page 87** This chapter describes how to write object-oriented application programs for LANDP for DOS, OS/2, and Windows NT.
- Chapter 6, “LANDP support for Java” on page 95** This chapter describes support for the Java programming language in LANDP for OS/2 and Windows NT.
- Chapter 7, “Writing programs using VisualAge for COBOL” on page 103** This chapter describes how to write LANDP application programs in VisualAge® COBOL.
- Chapter 8, “VisualAge Generator Application Programming Interface” on page 105** This chapter describes how to use VisualAge Generator application programs with LANDP.
- Chapter 9, “LANDP for OS/2 REXX application programming interface” on page 113** This chapter describes how to write REXX programs to use the LANDP for OS/2 application programming interface.
- Chapter 10, “Testing your application programs” on page 117** This chapter describes how to use the LANDP system verification program to test user-written client and server application programs.
- Chapter 11, “Sample application programs” on page 127** This chapter supplies annotated listings of some of the supplied sample application programs.

A bibliography, glossary, and index are provided at the back of the book.

---

## Conventions used in this book

This book uses a conventional notation for the system environments in which LANDP server functions operate, for the values of function codes, return codes, and event codes, and for describing fields that contain blanks.

## Operating environments

Throughout the book there are tables that indicate the functions that are supported by each IBM-supplied LANDP server. Not all functions operate in all platforms, and the tables use the following convention to show the operating environment:

- 0 LANDP for DOS
- 2 LANDP for OS/2
- 6 LANDP for AIX
- N LANDP for Windows NT

For example, “-26-” denotes a function available from a LANDP server that runs in the LANDP for OS/2 or LANDP for AIX environment, but is not available on LANDP for DOS, or LANDP for Windows NT.



Also, the same information is given in words for each function request, or a global statement is made at the start of the chapter that describes the server.

## Function, return, and event codes

References made in this book to the contents of the following fields use a symbolic notation to simplify the understanding of the actual values that these fields contain:

- Function code
- Router return code
- Server return code
- Asynchronous event code

**Function codes and asynchronous event codes** are unsigned two-byte integers.

Their values have been chosen so that if you represent their numeric value in ASCII form, you get a significant string.

For example, the value X'494E' represents the function code **IN**.

Value	Symbol
X'49'	I
X'4E'	N

On **Intel Machines**, because the function code field is a two-byte integer and stored in *Intel reversed format*, the field contains X'4E' in the first position and X'49' in the second memory position.

**Router and server return codes** are unsigned four-byte integers. To get the symbol for the numeric value, apply the previous rules to the two least significant bytes. To convert a symbol into a numeric value, use the same rule as for the function code and the asynchronous event code to get the two least significant bytes. Examples are given in the next table.

Examples of LANDP Return Codes		
Value	Two least significant bytes	Symbol
X'01004C38'	4C38	L8
X'01005031'	5031	P1

## Bit positions

The highest order bit in a byte is labelled bit 7, and the lowest bit 0.

## DB2 Universal Database®

In this book, all references to DB2® or DB2/2 apply to IBM DB2 for OS/2 and to IBM DB2 Universal Database®.

## Windows 2000

In this book, all references to Windows NT apply to Microsoft Windows NT and to Microsoft Windows 2000.

---

### Related information

The LANDP family is supported by the following books. In this book, references to other LANDP books use the shortened title shown here. For the full title, order number of these publications, and a comprehensive list of LANDP-related literature, refer to “Bibliography” on page 207.

*LANDP Introduction and Planning*

This book provides a brief description of the components and features of the LANDP family, and gives information about planning a LANDP system.

*LANDP Installation and Customization*

This book provides information about installing, customizing, and distributing the LANDP family.

*LANDP Programming Reference*

This book describes the application programming interfaces that are used to develop user servers and client applications.

*LANDP Programming Guide*

This book gives guidance on writing application programs to use the interfaces described in the *LANDP Programming Reference*.

*LANDP Problem Determination*

This book describes how to use trace tools, diagnostic programs, alerts, and return codes to debug code while developing LANDP applications and user servers, or resolve problems while using LANDP family components.

*LANDP Servers and System Management*

This book provides detailed information on the LANDP servers, and describes how to manage and administer a LANDP system.

### Web site

For more information about LANDP please visit our web site at:  
**<http://www.ibm.com/software/ts/landp/>**

---

## Summary of Changes

This manual has been updated to reflect enhancements made to LANDP in Version 5. The major changes in this version are:

- The LANDP MQSeries Link server enables LANDP applications to access the Message Queueing Interface of MQSeries®
- The LANDP TCP/IP wide area communications server enables existing SNA wide area communication networks to be replaced with TCP/IP networks without impact to LANDP applications interfacing to the LANDP SNA or PPC servers. The TCP/IP wide area communications server also supports LANDP's 3270 emulator over the TELNET protocol.
- The LANDP ODBC query server on Windows NT supports access to various relational databases through the LANDP API using industry standard ODBC drivers.
- The External Logging Replication (XLR) feature of the Shared File server, when used with the Service Availability Manager, provides improved performance and availability of replicated Shared File databases.
- The enhanced Java support enables access to LANDP services from devices not running LANDP code, for example, browser-based applications.
- Support for the IBM 9069 transaction printer has been added.
- The range of servers supported by LANDP on the Windows NT platform has been extended to be more comparable to the function available on OS/2. The additional servers available on Windows NT include Electronic Journal, Store for Forwarding/forwarding, System Manager, PPC and the 4748 DBCS printer servers.
- In addition to the new function which LANDP V5 delivers, the levels of operating systems and other system software with which LANDP operates have been updated.



## Chapter 1. Clients and servers

IBM LANDP Family (LANDP for DOS, OS/2, Windows NT, and AIX) provides the means to implement a client/server system in a distributed processing environment.

In LANDP, a *client* is an application program that needs the services of another program. A *server* is an application program that can provide these services to the client. The client and server may be running in the same machine or in another machine in the same LANDP *workgroup*. A LANDP workgroup is a set of workstations that form part of a local area network (LAN) and that use the client/server mechanism of LANDP.

When a client requests something of a server (a *server function*), it does not need to know where the server is located. Similarly, when a server receives a request, it treats the request the same way, regardless of where the client is located. However, if the server needs to know where the request came from, this information is available to it.

A server can be accessed by any number of clients and can process one or more requests at the same time depending on its characteristics.

The interaction between clients and servers is characterized by a request and reply process. Clients request services from LANDP or user-written servers by specifying the function to be performed, the server name, data, and any parameters that are required. The specified server receives the request, performs the required function, and returns the result as a return code, data, and any parameters that are defined. Figure 1 illustrates the general relationship between servers and clients.

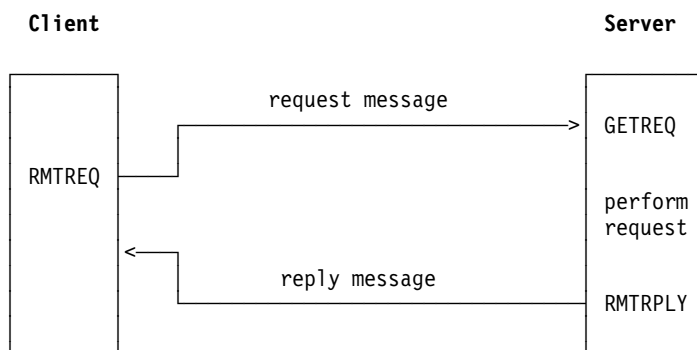


Figure 1. General Client/Server Relationship. RMTRQ, GETREQ, and RMTRPLY are some of the calls used to communicate between clients and servers.

The LANDP client/server mechanism makes the interaction between clients and servers consistent and transparent. Clients do not need to know on which LANDP workstation (DOS, OS/2, or Windows NT) or LANDP for AIX system the server is installed. Servers can be distributed within a LANDP workgroup. Clients request services from them without needing to identify the location of the server within the LANDP workgroup. The client/server mechanism automatically routes the requests to the required server

## communication between clients and servers

(identified by the server name) and the reply back to the appropriate client. Also, the client/server mechanism provides a set of *supervisor local functions*. Clients and servers can request these functions to influence their interaction and their operation.

A server can also request services from another server using the same programming interface as the clients. The LANDP electronic journal server, for example, takes advantage of this. It requests services from the LANDP shared-file server. In this relationship, the electronic journal server is *acting as a client*.

A server can also use other facilities provided by the operating system or by other programs. When LANDP for OS/2, Windows NT, and AIX servers are acting as clients, there are no limitations on what they can do. They can use any service provided by the operating system or by any other product. However, when a LANDP for DOS server is acting as a client, there are several restrictions because the server is a *terminate and stay resident* (TSR) program (see "Writing LANDP for DOS servers" on page 66 for more information about the restrictions).

A major part of the client/server mechanism is the application programming interface that is common for the entire LANDP family. This interface, called the *LANDP common API*, enables the clients to request services from the servers, and the servers to get the requests and send replies. (Where there is no ambiguity, the interface may be abbreviated to the *common API* or simply the *API*.)

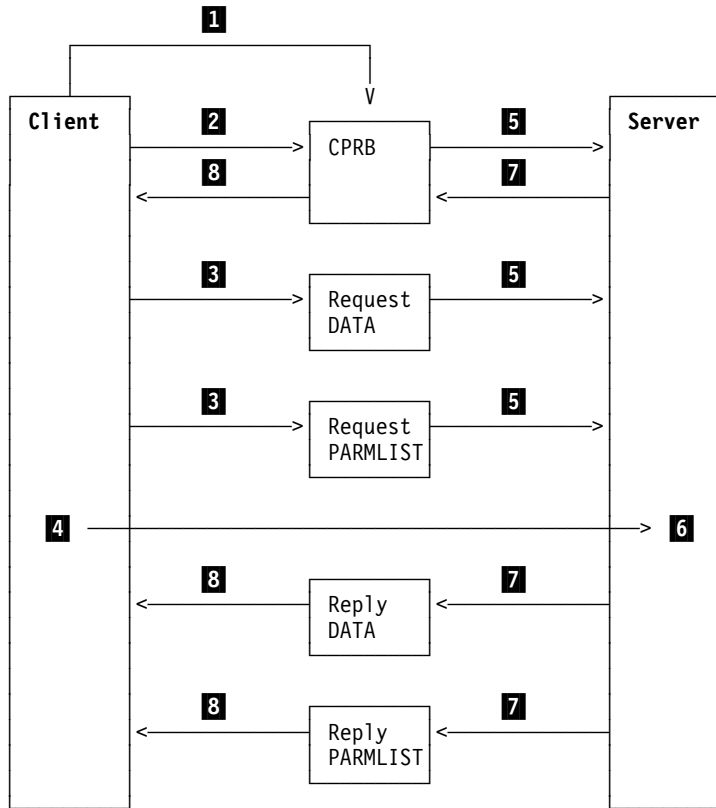
This common API is available with LANDP for DOS, OS/2, Windows NT, and AIX. It allows you to easily port LANDP application programs across DOS, OS/2, AIX, and Windows NT. However, application programs that use the *native* operating system functions (DOS, OS/2, AIX, and Windows NT) are limited in their portability. This should be considered when you design an application program.

---

## Communicating between clients and servers

Clients request a service from a server by calling a special entry point, RMTREQ, and supplying the address of the *connectivity programming request block* (CPRB). The CPRB contains data, parameters, and addresses necessary to describe the request and to identify the server. To exchange data and parameters between clients and servers, the CPRB also specifies the following complementary areas:

- |                         |  |
|-------------------------|--|
| <b>Request PARMLIST</b> | A parameter area used by the client to pass parameters to the server. It typically holds flags that specify in more detail how the requested service is to be performed, for example, that a record is to be retrieved for update. |
| <b>Reply PARMLIST</b>   | A parameter area used by the server to return parameters to the client. It typically holds flags that specify more detail about the requested service, for example, that a file was full.  |
| <b>Request DATA</b>     | A data area used by the client to transfer data to the server, for example, data intended to be printed.   |
| <b>Reply DATA</b>       | A data area used by the server to return data to the client, for example, a message received from a host computer.   |



- 1** Client defines within its own storage the areas for the CPRB, Request DATA, Request PARMLIST, Reply DATA, and Reply PARMLIST.
- 2** Client writes in the CPRB the information for the request to the server (function to be performed, Request DATA length, Reply DATA length, server ID, and so on).
- 3** Client writes Request DATA and Request PARMLIST information as required to the areas it defined in Step 1.
- 4** Client invokes the common API by calling RMTREQ.
- 5** Server receives the CPRB, Request DATA and Request PARMLIST information through the common API (calling GETREQ).
- 6** Server does the function requested.
- 7** Server completes the CPRB and provides information in the Reply DATA and Reply PARMLIST areas, as required.
- 8** Client receives the CPRB, Reply DATA, and Reply PARMLIST information through the common API.

## common API

The CPRB and the Request DATA and Reply DATA and Request PARMLIST and Reply PARMLIST areas are allocated in the storage space of the client application program. Before a client issues a request, it must initiate the required fields in the CPRB, and the associated PARMLIST and DATA areas. Request PARMLIST and Reply PARMLIST areas can be the same physical storage area. Request DATA and Reply DATA areas can be the same physical storage area.

The entire process is controlled by several routines that build up the common API. These routines are provided by LANDP in libraries that must be included in the application at link time. These routines are:

**RMTRREQ** Called by the client to issue a request and get the reply. (LANDP for DOS, OS/2, and Windows NT applications that use RMTRREQ with the "NoWait" option must use the GETRPLY routine to get the reply.)

**GETREQ** Called by the server to obtain a request issued from a client.

**RMTRPLY** Called by the server to return its reply.

**GETRPLY** Called by LANDP for DOS, OS/2, and Windows NT clients, which used RMTRREQ with the "NoWait" option, to get the reply.

**RMTRAREQ** Called by the server to notify the client about asynchronous events.

**SRVINIT** Called by the server for its initialization upon loading.

The LANDP common API provides the means to write applications that are portable at source code level among the DOS, OS/2, AIX and Windows NT environments, and that can be written in a variety of commonly available programming languages.

### Common API used by clients

For clients, LANDP provides (as the common API) the routine called RMTRREQ. To request a LANDP service, the client initializes the CPRB and defines areas for passing data and parameters as required by the requested service. Then it calls RMTRREQ, and passes the address of the CPRB as a parameter.

RMTRREQ invokes the client/server mechanism that, in turn, forwards the request to the appropriate server. Servers can be local (installed on the same workstation) or remote (installed on LAN-attached LANDP for DOS, OS/2, Windows NT, and AIX workstations). The server then processes the request and returns the reply to the requesting workstation. Processing in the requesting client is suspended until the server returns its reply (unless the NoWait option is used).

Some older LANDP for OS/2 and Windows NT applications, and LANDP for DOS Windows 3.1 applications, need to issue a RMTRREQ to LANDP servers that may have a slow response. If these applications cannot wait for the server reply, they can issue a RMTRREQ with the "NoWait" option and then get the reply issuing a call to GETRPLY later. In this way they do not cause other Windowed applications to stop.



## Common API used by servers

The common API as used by the servers is characterized by the routines SRVINIT, GETREQ, RMTRPLY, and RMTAREQ.

The typical server structure consists of an initialization part, a request processing loop, and a termination part. The first part includes all initial processing needed by the server, and the processing of the SRVINIT routine to notify LANDP that the server is being loaded. In the request processing loop, the server gets a request using GETREQ, processes the request, and returns the result using RMTRPLY. A server receives a request each time it issues a GETREQ call. The server processes the termination part when it receives an End of Service (ES) request. It then releases all the resources that were used and terminates its processing.

The RMTAREQ routine enables a server to notify a client about the occurrence of an asynchronous event. This frees the client to continue with other work until it is notified by the server that the event has occurred. The client can send specific requests for polling a status and can control time periods itself.

## Connectivity programming request block (CPRB)

The following table shows the format of the CPRB. Some of these fields contain values that apply to most requests. Therefore they do not have to be changed for each request, if the same CPRB instance is used for multiple requests.

Connectivity Programming Request Block (CPRB)			
Offset	Length	Field Name	Content/Description
X'00' (00)	4	ehc_reserved1	Reserved; see note 1 on page 7
X'04' (04)	4	ehcretcode	Router return code
X'08' (08)	1	ehcverb_type	Verb type (provided by LANDP); see note 2 on page 7
X'09' (09)	1	ehc_flags	Flags field; see notes 3 and 4 on page 7
X'0A' (10)	2	ehcfunct	Function code
X'0C' (12)	2	ehctimeout	Timeout per request; see note 5 on page 8
X'0E' (14)	2	ehcqparml	Request PARMLIST length; see note 6 on page 8
X'10' (16)	4	ehcqparmad	Request PARMLIST address; see note 7 on page 8
X'14' (20)	2	ehcqdatal	Request DATA length; see note 6 on page 8
X'16' (22)	4	ehcqdataad	Request DATA address; see note 7 on page 8

Connectivity Programming Request Block (CPRB)			
Offset	Length	Field Name	Content/Description
X'1A' (26)	2	ehcrparml	Reply PARMLIST length; see note 6 on page 8
X'1C' (28)	4	ehcrpamad	Reply PARMLIST address; see note 8 on page 8
X'20' (32)	2	ehcrdatal	Reply DATA length; see note 6 on page 8
X'22' (34)	4	ehcrdataad	Reply DATA address; see note 8 on page 8
X'26' (38)	2	ehc_event_id	LANDP for OS/2 and Windows NT only: event ID for ZN function (see the “Extended asynchronous event notification (ZN function)” section in the “Supervisor local functions” chapter of the <i>LANDP Programming Reference</i> .); otherwise, reserved (see note 1 on page 7)
X'28' (40)	4	ehcservrc	Server return code provided by the server
X'2C' (44)	2	ehcrepldplen	Replied PARMLIST length provided by the server
X'2E' (46)	2	ehcreplddlen	Replied DATA length provided by the server
X'30' (48)	2	ehcpc_id	Originator workstation ID; see note 9 on page 8
X'32' (50)	8	ehcresource_origin	Originator resource name; see note 10 on page 8
X'3A' (58)	2	ehcdest_pc_id	Destination workstation ID; see note 3 on page 7
X'3C' (60)	6	ehc_reserved5	Reserved; see note 1 on page 7
X'42' (66)	2	ehcfirst_pc_origin	First originator workstation ID of a server-to-server request; see note 11 on page 8
X'44' (68)	8	ehcfirst_res_origin	First originator resource name of a server-to-server request; see note 11 on page 8
X'4C' (76)	1	ehcfields_meaningful	Server-to-server request indicator provided by LANDP; see note 12 on page 8
X'4D' (77)	1	ehc_reserved6	Reserved; see note 1 on page 7

Connectivity Programming Request Block (CPRB)			
Offset	Length	Field Name	Content/Description
X'4E' (78)	2	ehcfirst_pid_origin	First originator process ID of a server-to-server request; see note 11 on page 8
X'50' (80)	4	ehc_reserved7	Reserved; see note 1 on page 7
X'54' (84)	2	ehcpid_origin	Originator process ID
X'56' (86)	2	ehcpid_dest	Destination process ID
X'58' (88)	6	ehc_reserved8	Reserved; see note 1 on page 7
X'5E' (94)	2	ehcservnamlen	Destination server name length. Always equal to X'0008'.
X'60' (96)	8	ehcserver	Destination server name (uppercase, left-justified, and padded with blanks); see note 13 on page 8

**Notes:**

1. The fields marked *Reserved* must be initialized with binary zero values once before issuing the first request. The fields are then used internally and must not be changed by the client or the server.
2. The client/server mechanism provides a value for the *verb type* that only servers can read. The values used are:
  - X'01' for a request
  - X'02' for a reply
  - X'03' for an asynchronous request

The value is also X'03' for process connection (&&) and for process disconnection (\*\*). These are described in Chapter 3, "Writing your own server programs" on page 47.

3. Explicit destination routing is where a request is directed to a specific server in a specific workstation, rather than using the standard LANDP routing system, which is fixed at customization time. The *ehcdest\_pc\_id* field specifies the destination. One bit (bit 5) in *ehc\_flags* indicates whether such routing is to be used:

B'..0.....': do not use explicit destination routing

B'..1.....': use explicit destination routing

See also note 4. Other flags are reserved; see note 1.

4. LANDP for OS/2 and Windows NT only: three bits in *ehc\_flags* are used with the Extended Asynchronous Event Notification (ZN) supervisor local function (see the "Extended asynchronous event notification (ZN function)" section in the "Supervisor local functions" chapter of the *LANDP Programming Reference*):

B'....1...': bit 3 = 1 means that bits 1 and 4 have significant values

B'...X....': bit 4 value

B'.....X.': bit 1 value

See also note 3 on page 7. Other flags are reserved; see note 1 on page 7.

5. During customization you define a default timeout in seconds (the LAN TIMEOUT parameter) for all the requests. If you want to modify the timeout value for a specific request, you must provide the appropriate value in this field. Allowed values are in the range from X'0001' to X'7FFF' seconds. When the contents of this field are X'0000', the customized timeout is assumed. For value X'FFFF', the timer is not limited to a certain time.
6. These length specifications are set by the client. The server cannot change them.
7. These address specifications are used by the server to access information (parameter and data) provided by the client. The server cannot change them.
8. These address fields are specified by the client. The servers use these areas to write the output information, and must not change these addresses.
9. This field, set by RMTREQ, is useful for servers. They can inspect it whenever such information is needed. For more information, see Chapter 3, "Writing your own server programs" on page 47.
10. This is the name of the client (the process that creates the request) that calls the RMTREQ. A server that issues a request must initiate this field with its server name, left-justified, and padded with blanks.
11. First resource origin on server-to-server communication. If a server that just received a request needs to call another server and wants to identify the original resource initiating the client to it, it can provide the client's resource ID, workstation ID, and process ID. (A process is either a server or a client.)
12. This field shows whether the three fields at offsets X'42', X'44', and X'4E' contain significant information. If it contains the character '+', these fields tell the called server the workstation ID, the resource ID, and the process ID of the original client.

The server which was called first must fill in these fields.

13. To make requests to a server, you can use any of its registered service names (see "Server names" on page 10).

The following CPRB fields contain ASCII characters:

- Originator workstation ID (*ehcpc\_id*)
- Originator resource name (*ehcresource\_origin*)
- Destination workstation ID (*ehcdest\_pc\_id*)
- First originator workstation ID (*ehcfirst\_pc\_origin*)
- First originator resource name (*ehcfirst\_res\_origin*)
- Server-to-server request indicator (*ehcfields\_meaningful*)
- Destination server name (*ehcserver*)

The remaining fields are filled with binary information. In these binary fields, words and integers are in byte-reversed format and long integers are in word-reversed byte-reversed format.

### Including the CPRB and options control block structures

When you use the common API, you need to include the CPRB structure (and, under LANDP for OS/2 and Windows NT, the *options control block* structures—see “The RMTREQ options control block (EHC\_RMTREQ\_OPTS)” on page 31 and “The GETRPLY options control block (EHC\_GETRPLY\_OPTS)” on page 34) provided with LANDP into the application program. To do this, use the following statements:

Language	Statement
VisualAge C++ Visual C++ C Set/2 C/6000	#include <EHCDEFC.H>
MASM/2	include EHCDEFM.INC
Pascal/2 Pascal/6000	(*\$I EHCDEFP.INC *)
VisualAge COBOL COBOL/2™ COBOL/6000 Micro Focus COBOL	(In the WORKING-STORAGE SECTION) COPY “EHCDEFVA.CBL” COPY EHCDEFB.CBL

### Passing parameters

Parameters are passed between clients and servers within defined PARMLIST areas. The server specifications tell you which parameters are to be provided for each case. The CPRB fields associated with the PARMLISTS are:

*Request PARMLIST length* (ehcqparml, at offset X'0E')

The length in bytes of the PARMLIST **at request**. If a value of 0 is specified, no parameters are provided at request.

*Request PARMLIST address* (ehcqparmad, at offset X'10')

The address of the PARMLIST used **at request**.

*Reply PARMLIST length* (ehcrparml, at offset X'1A')

The length in bytes of the PARMLIST **used by the server** to return parameters.

*Reply PARMLIST address* (ehcrparmad, at offset X'1C')

The address of the PARMLIST used by the server **at reply**. It can be the same as the Request PARMLIST address field.

*Replied PARMLIST length* (ehcreplplen, at offset X'2C')

The *actual* length in bytes of the PARMLIST **returned by the server**.

## naming servers

### Passing data

Data is passed between clients and servers within defined Request DATA and Reply DATA areas. The server specifications tell you which data is to be provided for each case. The CPRB fields associated with the data transfer are as follows:

*Request DATA length* (ehcqdatal, at offset X'14')

The length in bytes of the DATA area **at request**. If a value of 0 is specified, no data is provided at request.

*Request DATA address* (ehcqdataad, at offset X'16')

The address of the DATA area used **at request**.

*Reply DATA length* (ehcrdatal, at offset X'20')

The length in bytes of the DATA area **used by the server** to return data.

*Reply DATA address* (ehcrdataad, at offset X'22')

The address of the DATA used by the server **at reply**. It can be the same as the Request DATA address field.

*Replied DATA length* (ehcreplddlen, at offset X'2E')

The *actual* length in bytes of the DATA area **used by the server** to return data.

#### Notes:

1. The length of PARMLIST and DATA areas is limited.

LANDP for DOS, OS/2, and Windows NT:

The sum of Request PARMLIST and Request DATA length must be less than or equal to 57500 bytes.

LANDP for AIX:

The sum of Request PARMLIST and Request DATA length must be less than or equal to 4160 bytes.

These values are also the maximum values for the sum of the Reply PARMLIST and Reply DATA lengths.

2. You should always specify the length of the PARMLIST fields as indicated in the text. Often, this length is 26 (decimal), even though some of it may not be required by the specific function requested.

### Naming system resources

LANDP for OS/2, Windows NT, and AIX use system resources, such as shared memory, system semaphores, queues, and pipes. To avoid conflicts, clients and servers that need to allocate their own resources must not use names that start with **EHC** for LANDP for OS/2 and LANDP for Windows NT or **DCZY** for LANDP for AIX.

### Server names

To request a LANDP service, the client must provide the name of the server required in the CPRB field *ehcserver*.

LANDP servers provide general network services to share software and hardware resources for applications running in a LANDP workgroup. To access these resources, LANDP servers can register and make available to clients, one or more *service names*. The service names are registered to LANDP using the SRVINIT call. Using SRVINIT, a server can register:

- The service name that is the name of the server executable file.
- In LANDP for OS/2, Windows NT, and AIX, service names that are used in addition to, or instead of, the server executable file name. You can also specify an alias name—see “Alias names in LANDP for OS/2, Windows NT, and AIX” on page 12.

The statements required to call SRVINIT are given in “Call SRVINIT (server initialization)” on page 49.

Any of these registered service names can be entered in the *server name* field of the CPRB, when requesting a function that is provided by a server. The service names are described in the chapters of *LANDP Programming Reference* that describe each LANDP server.

### **Mono-service servers**

A server that makes available only *one* service name is called a *mono-service server*. Usually, mono-service servers provide the services of a unique resource, although a server can also define a service name for a set of resources.

### **Multiple-service servers**

A server that makes available several service names is called a *multiple-service server*. Each service name can correspond to different resources. A server makes several service names available, in two possible ways:

- By registering service names with the SRVINIT call.

In LANDP for OS/2, Windows NT, and AIX, a SRVINIT call can be issued with or without a service names list. Clients then access a server using any of its registered service names. This name (see below) can be any valid name, from one to eight characters long. An alias name can also be specified.

In LANDP for DOS, a SRVINIT call can only be issued without a service names list. The server's executable file name is registered to LANDP.

- By using the '#' character as a place-holder for any valid characters that you can enter in the server name. A server with a name including a # character is a set of logical servers with names resulting from the substitution of this special character.

For example, if a server with the name PRINTER# is loaded, clients can specify the server name as PRINTER1, PRINTER2, PRINTERA, or PRINTERB, as specified during customization. The request is routed to the server if the LAN was configured as the server giving services to the client. If you load the PRINTER# server, it is as if several servers with names resulting from the substitution of the # character by valid characters had been loaded. A set of logical servers are loaded and provide the same services for several objects managed by the same server.

## naming servers

The names of the servers provided by LANDP are reserved. Do not use any of the names listed below, nor names beginning with the characters **EH**C or **DC**ZY to name your files.

BIWP	BMLS	BMOP	BPP	DCADLC
DTAQ	DTAU4733	EHCTCP	ELECJO##	EMU3270
EMU3287	FORWARD	LDA7	EHCMQ##	MSRE47##
EHCOB##	OPBS	OPER	PBMS	PINP47##
PPC	PR4748##	PR4770##	PR47X2##	PRTMGR
PT4721	RCMS	RDVVOLS	SDLC	SFORFORW
SFQUERY	SHFILE##	SHRDIR	SMGR	SMOP
SNA##	SPV	SP4721##	SS#####	SS#####
TRDLC	VBIWP	VFILE	X25DLC	X25DLC2
X25NAT##				

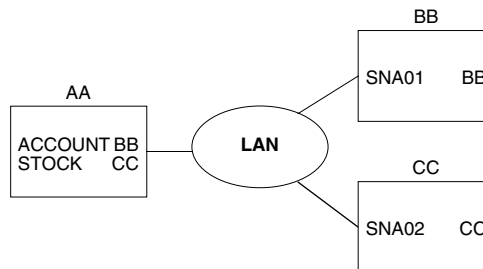
The # character can appear in any position in the server name, provided it follows the server name restriction described above. This means, for example, that a server name with eight # characters is not allowed, because this would include reserved names.

### Alias names in LANDP for OS/2, Windows NT, and AIX

With LANDP for OS/2 and Windows NT, and LANDP for AIX you can specify logical names for each resource during customization. (See the *LANDP Installation and Customization* book for more information.) These *alias names* are handled transparently by LANDP. Three examples of the use of alias names are given below:

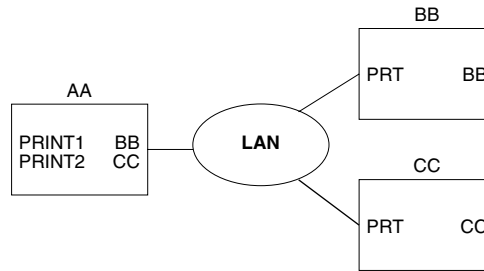
- You can apply other preferred names to LANDP servers.

In this example, AA, BB, and CC are workstation identifiers. Workstation AA has defined resource SNA01 in workstation BB, and resource SNA02 in workstation CC. Aliases are defined, with SNA01 having ACCOUNT as an alias name, and SNA02 having STOCK: CC



- You can enable one client to access replicated servers that are located in various workstations. LANDP for OS/2, Windows NT, and AIX allow one client/workstation to access the same server replicated in different workstations. Different alias names can be applied to the same resource when located in different workstations:





- You can use replicated servers to back up your servers dynamically. Specify two names in the client— one for the default server, and one for the backup server. Load the server program with its actual file name. Then, application programs running under LANDP for OS/2, Windows NT, and AIX can request a service from a server by specifying the previously defined alias name in the server name field of the CPRB, before calling RMTREQ.

## Compiling and linking your application program

The routines of the LANDP common API are included in a library file. Table 1 shows the files needed for application program development.

Table 1 (Page 1 of 2). Files needed for LANDP application program development	
File	Description
EHCDOS.LIB (LANDP for DOS)	Library needed by the application programs and user-written servers during link editing. See also “LANDP for OS/2 programming environments” on page 14 for more information.
EHCDOSXM.LIB (LANDP for DOS for use with Micro Focus COBOL programs that use the XM interface)	
EHCOS216.LIB (LANDP for OS/2) or EHCOS2.LIB (FBSS/2)	
EHCOS232.LIB (LANDP for OS/2) LIBDCZY.A (LANDP for AIX)	
EHCDEFM.INC	CPRB structure for programs written in MASM/2.
EHCDEFVA.CBL	CPRB structure for programs written in VisualAge COBOL
EHCDEFB.CBL	CPRB structure for programs written in COBOL.

Table 1 (Page 2 of 2). Files needed for LANDP application program development	
File	Description
EHCDEFC.H	CPRB structure for programs written in VisualAge C++, Visual C++, C, C/2™, C Set/2, or C/6000.
EHCDEFP.INC	CPRB structure for programs written in Pascal/2 or Pascal/6000.
EHCWINNT.COF (LANDP for Windows NT: COFF)  EHCWINNT.OMF (LANDP for Windows NT: OMF)	Libraries needed by the application programs and user-written servers during link editing with COFF (for example, for Microsoft Visual C++) or OMF (for example, for VisualAge C++) format object files and libraries. To use the libraries, copy the appropriate file to EHCWINNT.LIB.  See also “LANDP for Windows NT programming environments” on page 17 for more information.

Application programs using LANDP routines must specify the library at program link time, and also any other libraries needed.

---

### LANDP for OS/2 programming environments

LANDP for OS/2 runs on IBM OS/2 Warp V4 (or higher), which supports the 16-bit and 32-bit programming environments. This section describes how the dynamic link library (DLL), the import library, and the include file provided by LANDP for OS/2 enable you to develop application programs (clients or user servers) for the 16-bit or the 32-bit environment.

### LANDP for OS/2 dynamic link library

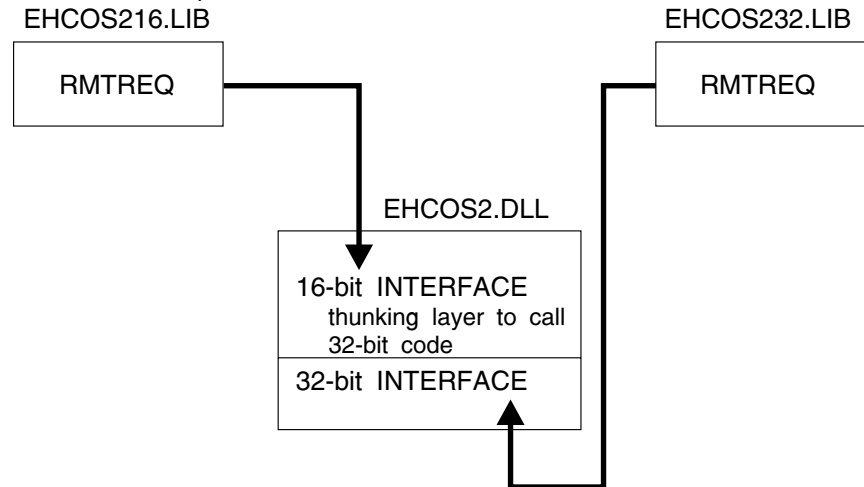
LANDP for OS/2 provides the EHCOS2.DLL dynamic link library (DLL) that contains the interface routines RMTREQ, GETRPLY, SRVINIT, GETREQ, RMTRPLY, and RMTAREQ.

EHCOS2.DLL is a 32-bit dynamic link library that provides entry points for both 16-bit and 32-bit programs. For the 16-bit entry points, the DLL provides a “thunking” layer which converts 16-bit calls to 32-bit calls. This conversion is done transparently by EHCOS2.DLL. For more information about “thunking,” see the *Application Design Guide* for your level of OS/2.

Clients or user servers that belong to the 16-bit or the 32-bit environment call the same set of interface routines, for example, RMTREQ. To provide the appropriate conversion, LANDP for OS/2 provides two import libraries to be used when linking the program: EHCOS216.LIB and EHCOS232.LIB.

**Note:** LANDP for OS/2 also provides the library EHCOS2.LIB for compatibility with FBSS/2. You can use EHCOS2.LIB or EHCOS216.LIB for your 16-bit applications.

For example, the routine `RMTRREQ` exists with the same name as an entry point in both libraries. The relationship between the import libraries and the DLL for 16-bit and 32-bit program environments is shown in the following figure. This relationship is the same for the other routines provided with the LANDP common API:



## LANDP for OS/2 program types

LANDP for OS/2 considers three different types of programs according to the programming environment to which they belong:

- Pure 16-bit programs
- Pure 32-bit programs
- Mixed 16-bit and 32-bit programs

### Pure 16-bit programs

These programs are developed using 16-bit code objects that call the LANDP common API routines, for example, `RMTRREQ`. These 16-bit programs could be existing programs developed in a 16-bit environment for IBM Financial Branch System Services or 16-bit LANDP for OS/2 programs developed on OS/2 V2.0. See the *IBM OS/2 Application Design Guide*.

These 16-bit programs:

- Use a 16-bit compiler
- Use the 16-bit OS/2 toolkit (if needed)
- Include the appropriate FBSS or LANDP for OS/2 include file
- Use a 16-bit linker with the import library: `EHCOS216.LIB` (or library `EHCOS2.LIB` for FBSS/2)

These programs have a 16-bit EXE-format, and must run on OS/2 Warp Version 4.0 (or later) with LANDP for OS/2.

### Pure 32-bit programs

These programs are developed using 32-bit code objects that call the LANDP common API routines. These 32-bit programs must be developed in a 32-bit environment with, for example, the IBM VisualAge for C++ compiler.

These 32-bit programs:

- Use a 32-bit (C or C++) compiler
- Use the 32-bit OS/2 Warp V4.0 (or later) Toolkit (if needed)
- Include the LANDP for OS/2 file: EHCDEFC.H
- Use a 32-bit linker with the import library: EHCOS232.LIB

These programs have a 32-bit EXE-format and must run on OS/2 Warp V4.0 (or later) with LANDP for OS/2.

### Mixed 32-bit programs

These programs are developed using two different types of objects:

- 16-bit code objects
- 32-bit code objects

These programs consist of 16-bit objects, developed in a 16-bit programming environment, and 32-bit objects. These programs need to follow the rules for mixed 32-bit programs as explained in the *Application Design Guide* for your level of OS/2. These programs:

- Use a 32-bit (C or C++) compiler and the corresponding toolkit (if needed)
- Use a 16-bit compiler and the corresponding toolkit (if needed)
- Include the file: EHCDEFC.H
- Use a 32-bit link program with the import library: EHCOS232.LIB

These programs have a 32-bit EXE-format, and must run on OS/2 Warp V4.0 (or later) with LANDP for OS/2.

### Include file EHCDEFC.H for C and C++ language programs

Clients and user servers developed in the C or C++ language need to include the EHCDEFC.H file. This file contains the definitions of the LANDP common API for the different environments.

For 32-bit programs, the include file uses the `__32BIT__` macro. This declares the LANDP common API using the 32-bit `_System` linkage convention.

For C++ programs, EHCDEFC.H must be included as an external C header file. This can be done as follows:

```
extern "C"
{
#include <ehcdefc.h>
}
```

## LANDP for Windows NT programming environments

LANDP for Windows NT runs on Microsoft Windows NT Version 3.51 or higher, which supports 32-bit programming environments. This section describes how the dynamic link library (DLL), the import library, and the include file provided by LANDP for Windows NT enable you to develop application programs (clients or user servers) for 32-bit environments.

Examples of these environments are:

- Microsoft Visual C++

This development environment processes object files in the Common Object File Format (COFF). Link them with the COFF type library, which is in EHCWINNT.COF. Copy this file to EHCWINNT.LIB.

- VisualAge C++

This development environment (provided by, for example, Borland and Symantec compilers) processes object files in the Intel Object Module Format (OMF) file format. Link them with the OMF type library, which is in EHCWINNT.OMF. Copy this file to EHCWINNT.LIB.

### Size Mismatch for Type bool in Visual C++

A size mismatch for type bool can occur in Visual C++ 4.2 Programs that are built with Visual C++ 6.0.

In Visual C++4.2, the Standard C++ header files contained a typedef that equated bool with int. In Visual C++ 6.0, bool is implemented as a built-in type with a size of 1 byte. This means that for Visual C++ 4.2, a call of sizeof(bool) yields 4, while in Visual C++ 6.0, the same call yields 1. This can cause memory corruption problems if you have defined structure members of type bool in Visual C++ 4.2 and are mixing object files (OBJ) or DLLs built with the 4.2 and 6.0 compilers.

## LANDP for Windows NT dynamic link library

LANDP for Windows NT provides the EHCWINNT.DLL dynamic link library (DLL) that contains the interface routines RMTREQ, GETRPLY, SRVINIT, GETREQ, RMTRPLY, and RMTAREQ.

EHCWINNT.DLL is a 32-bit dynamic link library that provides entry points for 32-bit programs.

EHCWINNT.COF and EHCWINNT.OMF (copied, as appropriate, to EHCWINNT.LIB) are provided as the import libraries when linking programs.

## Include file EHCDEFC.H for C and C++ language programs

Clients and user servers developed in the C++ language need to include the EHCDEFC.H file.

## migration considerations

For C++ programs, EHCDEF.C must be included as an external C header file. This can be done as follows:

```
extern "C"
{
#include <ehcdefc.h>
}
```

The calling convention is CDECL (not Pascal, as on DOS and OS/2).

---

## Migration considerations

This section describes changes that you may need to make to your application programs (clients and user servers), and also some compatibility considerations if you are using any of the supplied LANDP servers.

Considerations that apply to specific servers are described later, in the sections that describe those servers.

### Moving to the latest LANDP for DOS

The initialize (IN) and disconnect (EJ) functions are now mandatory in LANDP for DOS application programs. The connect (CN) function is also required by the SNA communications server. If your programs do not contain these requests, then add them. (IN and EJ are described in the “Extended asynchronous event notification (ZN function)” chapter in the “Supervisor local functions” chapter of the *LANDP Programming Reference*. CN is described in the “SNA communication server” chapter in the *LANDP Programming Reference*.)

Two supplied applications issue the supervisor IN and the SNA server CN functions for user applications that do not contain these function requests. They are EHCIN and EHCCONN respectively. They must be executed before starting the application.

EHCIN has no parameters.

The syntax of the EHCCONN program call is:

```
EHCCONN /SNA01/SNA02/.../SNA $n$ 
```

where:

```
SNA01, ... SNA $n$ 
```

are the SNA sessions required by the application.

There is also a corresponding program, EHCREL, that issues a release (RL) function request to the SNA server. Its syntax is:

```
EHCREL /SNA01/SNA02/.../SNA $n$ 
```

## Moving LANDP for DOS and FBSS (DOS) services to LANDP for OS/2

You can continue running your applications on DOS, but go over to use the servers provided by LANDP for OS/2. You need to install LANDP for DOS in all DOS workstations and LANDP for OS/2 in all OS/2 workstations, and during customization specify which LANDP for OS/2 servers you want to use.

## Moving LANDP for DOS and FBSS (DOS) services to LANDP for Windows NT

You can continue running your applications on DOS, but go over to use the servers provided by LANDP for Windows NT. You need to install LANDP for DOS in all DOS workstations and LANDP for Windows NT in all Windows NT workstations, and during customization specify which LANDP for Windows NT servers you want to use.

## Migrating FBSS (DOS) clients or user servers to the LANDP common API

The LANDP and FBSS program families provide a common API that enables you to write applications and servers that can be ported at the source code level from one current LANDP or FBSS program to another. The following sections describe how to migrate existing FBSS (DOS) applications or servers to this interface.

### Migrating existing clients

If the application uses the CPRB/SEND\_REQUEST interface, make the following changes:

1. Change every occurrence of the verb `SEND_REQUEST` to `RMTREQ`.
2. Change the CPRB type to **EHC\_CPRB**.

The LANDP family provides a definition of a CPRB structure that can be copied into the program. To include it in the program, use the following statements:

Language	Statement
VisualAge C++ Visual C++ C/2 C Set/2	<b>#include &lt;EHCDEF.C.H&gt;</b> instead of <code>#include "UUCPCRB.H"</code>
VisualAge COBOL COBOL/2	(in the WORKING-STORAGE SECTION) <b>COPY "EHCDEFVA.CBL"</b> <b>COPY EHCDEF.CB.CBL</b> instead of <code>COPY UUCBCPRB.CBL</code>
Pascal/2	<b>(*I EHCDEF.INC*)</b> instead of <code>*\$include "UUPCPRB.INC"</code>
MASM/2	<b>include EHCDEFM.INC</b> instead of <code>include UUMCPRB.INC</code>

3. Change the names of the CPRB fields (by changing the first three characters from **UER** to **EHC**). For example *uerqparmsl* (request parameter length) becomes *ehcqparmsl*.

## migration considerations

Depending on the programming language used, it may be necessary to make one more change in the definition of the server name. Formerly, using C or COBOL, the *uerver* field of the CPRB contained a pointer to the address of a string containing the server name. Now, the *ehcserver* field is 8 bytes long and must contain the server name itself.

For example, using VisualAge C++, where the current application has:

```
cprb.uerver = "SNA03  ";
```

you must change this to:

```
memcpy(cprb.ehserver,"SNA03  ",8);
```

Using Pascal/2 this change is not necessary since it already provided the server name, not the address.

4. Change the length of the variable to store the return code from RMTREQ. For example, in C or C++, the variable type has to be changed from long integer to short integer. (This modification has been introduced to conform with OS/2 remote calls programming rules. System routines in OS/2 return an integer in the AX register.)
5. In FBSS V2.1.1, the return code is the router return code, which could have been used directly without checking the CPRB. Now, the return code is nonzero if any of the CPRB return codes (router or server) is nonzero. The application must check the return codes in the CPRB if it did not already do so.
6. A new parameter has to be added at the end of every call. This parameter (EHC\_RESERVED) is language independent. For further information, refer to the specific call description, for example "Call RMTREQ (Remote request)" on page 28.

If the application uses the COMMAREA/DATAREA format, make the following change:

- Map the COMMAREA fields into a CPRB.

**Note:** This is the same change needed to migrate the API from FBSS V2.0 to FBSS V2.1. For a detailed explanation refer to the *IBM FBSS Version 2 Programmer's Reference Manual*.

### Migrating existing user servers

The existing user servers for FBSS (DOS) run under LANDP for DOS without modification even if they are used in a mixed LAN. However, if they have to run in an OS/2 workstation they have to be modified to use the common API.

- You must call server initialization (SRVINIT).
- You must issue a GETREQ (get request) call to receive requests and a RMTRPLY (remote reply) at the end of the request processing. The request processing itself can be kept unchanged.
- You must change type Dn (device type) asynchronous events to use the Z5 or ZN (under LANDP for OS/2) function code.



The WM function does not support type Dn asynchronous events (with function code Z2). Use Z5 or ZN instead.

The server can use the Z5 or ZN function to send asynchronous messages or signals to any client. The server must create a CPRB with function code Z5 or ZN and queue this CPRB to the FBSS client/server mechanism queue using the RMTAREQ routine.

For detailed information see “Call RMTAREQ (remote asynchronous request)” on page 55, and the “Extended asynchronous event notification (Z5 function)” and “Extended asynchronous event notification (ZN function)” sections in the “Supervisor local functions” chapter of the *LANDP Programming Reference*.

## Migrating FBSS/2 16-bit clients and user servers to 32-bit mode

The following steps are required to migrate 16-bit mode programs to 32-bit mode:

1. Migrate the programs to 32-bit mode following the programming requirements of OS/2 Warp V4.0 or higher. For a detailed description of these programming requirements, refer to your *OS/2 Application Design Guide*.
2. At the FBSS/2 source code level, the migration is carried out without any change in the code by:
  - Using EHCDEFC.H
  - Linking your program with EHCOS232.LIB
  - Running your program with EHCOS2.DLL

## Migrating LANDP for DOS and FBSS (DOS) clients to LANDP for OS/2

If you have a LANDP for DOS or FBSS (DOS) application using the common API and you want it to run under LANDP for OS/2, you must follow these rules:

1. Change functions that are unique to DOS.
 

Ensure that the application is portable from the operating system point of view. Use the subset of OS/2 system functions that are compatible with DOS.
2. Add the IN and EJ supervisor local functions, if they are not already included in the existing application. (See the “Supervisor local functions” chapter of the *LANDP Programming Reference*.)

It is mandatory that the first service requested by a LANDP application is the initialization (IN) function. Also, the program must issue the EJ function before ending.

3. Change the EX, SA, and RA supervisor local functions.

Use the WM (wait for asynchronous events) function instead of the EX (exit to idle state) function.

The WM function allows the application to remain in idle state while it is waiting for a *specific* asynchronous event or events specified in a parameter list. These events can be sent by the keyboard, the host processor, the I/O devices, user servers, user timers, or a timeout of a special WM timer. The client/server

## migration considerations

mechanism checks for these entries in the order specified by the WM function. While this function is running the application remains idle.

If no parameter list is specified for the WM function, the default provides the same function as the EX function.

Since the WM function allows waiting for a specific event, it is no longer necessary to use the enabling and disabling asynchronous events (SA and RA) functions.

LANDP for OS/2 does not support the asynchronous event type Dn in the WM function.

### Migrating LANDP for DOS and FBSS (DOS) user servers to LANDP for OS/2

If you want to migrate a LANDP for DOS or FBSS (DOS) user server that uses the common API to LANDP for OS/2, then:

1. Change functions that are unique to DOS.

Ensure that the server is portable from the operating system point of view. Use the subset of OS/2 system functions that are compatible with DOS.

2. The server cannot use the supervisor local function Z2.

If the server must be portable at the same source code level to a LANDP for DOS or FBSS (DOS) workstation, take into account that:

- The last statement in the initialization procedure must be the statement calling the SRVINIT routine. As control is not returned to the server in the LANDP for DOS or FBSS (DOS) environment, this routine allocates the server as a resident program in memory. Therefore the server at its initialization procedure must calculate the program size before calling the SRVINIT routine providing this size as a parameter.  
**Note:** In the LANDP for DOS or FBSS (DOS) environment, the routine SRVINIT does not return the control to the caller. Therefore statements following the SRVINIT call can be processed only in a LANDP for OS/2 environment.
- An entry point to the server process associated with every request received by the server must exist in the server code. The LANDP for DOS or FBSS (DOS) client/server mechanism calls a server as a routine, so the system must know this server entry point. The client/server mechanism calls the server at the entry point address specified in a parameter of the SRVINIT function.
- RMTRPLY in LANDP for DOS and FBSS (DOS) ends the routine, returning control to the client/server mechanism. Therefore anything following the call to RMTRPLY is not processed under DOS. However, in LANDP for OS/2 the RMTRPLY routine returns control to the caller, requiring a loop statement to the GETREQ to wait for the next request to be processed.

### Migrating LANDP for DOS and FBSS (DOS) clients to LANDP for Windows NT

If you have a LANDP for DOS or FBSS (DOS) application using the common API and you want it to run under LANDP for Windows NT, you must follow these rules:

1. Change functions that are unique to DOS.

Ensure that the application is portable from the operating system point of view. Use the subset of Windows NT system functions that are compatible with DOS.

2. Add the IN and EJ supervisor local functions, if they are not already included in the existing application. (See the “Supervisor local functions” chapter of the *LANDP Programming Reference*.)

It is mandatory that the first service requested by a LANDP application is the initialization (IN) function. Also, the program must issue the EJ function before ending.

3. Change the EX, SA, and RA supervisor local functions.

Use the WM (wait for asynchronous events) function instead of the EX (exit to idle state) function.

The WM function allows the application to remain in idle state while it is waiting for a *specific* asynchronous event or events specified in a parameter list. These events can be sent by the keyboard, the host processor, the I/O devices, user servers, user timers, or a timeout of a special WM timer. The client/server mechanism checks for these entries in the order specified by the WM function. While this function is running the application remains idle.

If no parameter list is specified for the WM function, the default provides the same function as the EX function.

Since the WM function allows waiting for a specific event, it is no longer necessary to use the enabling and disabling asynchronous events (SA and RA) functions.

LANDP for Windows NT does not support the asynchronous event type Dn in the WM function.

## Migrating LANDP for DOS and FBSS (DOS) user servers to LANDP for Windows NT

If you want to migrate a LANDP for DOS or FBSS (DOS) user server that uses the common API to LANDP for Windows NT, then:

1. Change functions that are unique to DOS.

Ensure that the server is portable from the operating system point of view. Use the subset of Windows NT system functions that are compatible with DOS.

2. The server cannot use the supervisor local function Z2.

If the server must be portable at the same source code level to a LANDP for DOS or FBSS (DOS) workstation, take into account that:

- The last statement in the initialization procedure must be the statement calling the SRVINIT routine. As control is not returned to the server in the LANDP for DOS or FBSS (DOS) environment, this routine allocates the server as a resident program in memory. Therefore the server at its initialization procedure must calculate the program size before calling the SRVINIT routine providing this size as a parameter.

**Note:** In the LANDP for DOS or FBSS (DOS) environment, the routine SRVINIT does not return the control to the caller. Therefore statements following the

## migration considerations

SRVINIT call can be processed only in a LANDP for Windows NT environment.

- An entry point to the server process associated with every request received by the server must exist in the server code. The LANDP for DOS or FBSS (DOS) client/server mechanism calls a server as a routine, so the system must know this server entry point. The client/server mechanism calls the server at the entry point address specified in a parameter of the SRVINIT function.
- RMTRPLY in LANDP for DOS and FBSS (DOS) ends the routine, returning control to the client/server mechanism. Therefore anything following the call to RMTRPLY is not processed under DOS. However, in LANDP for Windows NT, the RMTRPLY routine returns control to the caller, requiring a loop statement to the GETREQ to wait for the next request to be processed.

### Migrating from the shared-file server to the LANDP for OS/2 query server

Migrating existing applications that use the LANDP for DOS or the FBSS (DOS) shared-file server to work with the query server involves three steps:

#### 1. Migrating the application.

- Change the resource name in the requester CPRB from SHFILE## to EHCSQL##.
- Remove shared-file server functions that are not supported in the LANDP for OS/2 environment:
  - Open batch (OB), close batch (CB)
  - Read header (RH) (information not available in SQL database)

You must redesign your applications that use the batch mode of the shared-file server to work with the query server. Online mode must be used instead of batch mode. The EX function can be used in online mode to lock a table.

- There are some differences between the shared-file server and the database manager that should be noted:
  - Depending on the use of functions, accesses to the DB2 Universal Database have different performance characteristics. LANDP for OS/2 provides some control at server load time that allows you to select different processing options with different performance characteristics.
  - A non-ASCII data type for indexed fields can result in a different collating sequence of the data.
  - Because of multitasking, OS/2 may access databases in a different sequence. With the shared-file server all function calls are sequenced, while the query server functions are processed in parallel if possible. The maximum number of simultaneous threads is defined during query server customization.
  - There are some differences in the base locking mechanism between the shared-file server and the database manager. The SQL database manager performs physical record locking where the shared-file server performs logical record locking. This means that:

- Functions that give return code zero with the shared-file server can now return RL.
- Get functions do not return a record that is currently held by another workstation (RL status returned).

The meaning of "being held by another workstation" depends on the ISOLATION LEVEL parameter that is used when the server is bound to RDBMS. Although this is done transparently by the server, a manual rebind of the file EHCSQLRQ.BND is possible to change the isolation level. Refer to the DB2 UDB documentation for a description of the binding process parameters.

2. Migrating shared-file server data structures (DBD and PCB) to SQL tables using the utilities provided by LANDP for OS/2.

### EHCMGR1

This program reads the .PCB and .DBD definition files and optionally the record definition file, RDF.CFG, and creates an ASCII file with the correct SQL sentences to create the SQL tables.

### EHCS2Q or EHCDSQ

EHCS2Q (for LANDP for OS/2 workstations) or EHCDSQ (for LANDP for DOS or FBSS (DOS) workstations) read the ASCII files created by EHCMGR1 and use the query server to create the SQL tables.

3. Migrate existing shared-file server data files to SQL files using the utility program provided by LANDP for OS/2.

### EHCMGR2

This program runs on LANDP for DOS and reads the shared-file server data files and the output files from the data structure migration program (EHCMGR1). It uses the query server to move data to the SQL files.

### EHCMGROS

This program runs on LANDP for OS/2 and is functionally equivalent to EHCMGR2.

## Migrating LANDP for OS/2 clients and user servers to LANDP for Windows NT

If you have a LANDP for OS/2 client or user server using the common API and you want it to run under LANDP for Windows NT, change functions that are unique to OS/2. Ensure that the application is portable from the operating system point of view.

It is mandatory that the first service requested by a LANDP client application is the initialization (IN) function. Also, a client must issue the EJ function before ending.



## Chapter 2. Writing client programs

The LANDP common application programming interface (API) consists of several routines that client and server programs call when they interact with each other. To request a LANDP service, the client (or server acting as client) program calls the synchronous routine RMTREQ and supplies the address of the *connectivity programming request block* (CPRB) as a parameter. The CPRB is the control block used for specifying the request, and for the server program to supply the answer.

If the client and the server programs are not in the same workstation of the LANDP LAN, the client/server mechanism passes the request to the respective workstation. The client program does not need to know where the server program is located.

Calling the routine RMTREQ is similar to a main routine in a program calling a subroutine. The local or remote server program acts as such a subroutine. It processes the request and returns the results to the application program. The client program calling the RMTREQ is suspended until the reply arrives or the request timeout expires. The example below shows how the call is made using the C language:

```
retcode = RMTREQ(cprb_addr, EHC_RESERVED);
```

The statements required to call RMTREQ are given in “Call RMTREQ (Remote request)” on page 28.

---

### Invoking the common API

The common API contains a set of routines. These routines are provided by LANDP in an import library. Depending on the LANDP licensed program and the environment (16-bit, or 32-bit with OS/2 Warp V4 or higher or Windows NT), you need to link your application program with one of the following libraries:

- **EHCDOS.LIB** for LANDP for DOS (or **EHCDOSEX.LIB**, see note)
- **EHCOS216.LIB** for LANDP for OS/2 (or **EHCOS2.LIB** for FBSS/2)
- **EHCOS232.LIB** for LANDP for OS/2
- **EHCWINNT.COF** for LANDP for Windows NT (for Microsoft compilers, for example)
- **EHCWINNT.OMF** for LANDP for Windows NT (for VisualAge compilers, for example)
- **LIBDCZY.A** for LANDP for AIX

Copy **EHCWINNT.COF** or **EHCWINNT.OMF** to **EHCWINNT.LIB** as appropriate.

See “LANDP for OS/2 programming environments” on page 14 and “LANDP for Windows NT programming environments” on page 17 for more information.

**Note:** The EHCDOSEX.LIB library allows Micro Focus COBOL applications that use the XM interface and run in protected mode to work with extended memory and use LANDP services.

### CPRB fields used and set by clients

These entries are the fields the client program must supply when requesting a service.

CPRB Fields Used by Clients			
Offset	Length	Field Name	Content/Description
X'0A'	2	ehcfunct	Function code
X'0E'	2	ehcqparml	Request PARMLIST length
X'10'	4	ehcqparmad	Request PARMLIST address
X'14'	2	ehcqdatal	Request DATA length
X'16'	4	ehcqdataad	Request DATA address
X'1A'	2	ehcrparml	Reply PARMLIST length
X'1C'	4	ehcrparmad	Reply PARMLIST address
X'20'	2	ehcrdatal	Reply DATA length
X'22'	4	ehcrdataad	Reply DATA address
X'5E'	2	ehcservnamlen	Destination server name length always equal to X'0008'
X'60'	8	ehcserver	Destination server name

### Call RMTREQ (Remote request)

The common API routine that client programs use to request services from a server program is RMTREQ. To call RMTREQ, the following statements are used:

Language	Statement
VisualAge C++, Visual C++, C, C Set/2, C/6000	retcode = RMTREQ (cprb_addr, EHC_RESERVED);
MASM/2	@RMTREQ cprb_addr EHC_RESERVED
Pascal/2 Pascal/6000	retcode := RMTREQ (cprb_addr, EHC_RESERVED);
VisualAge COBOL	(In the PROCEDURE DIVISION) CALL "RMTREQ" USING BY REFERENCE EHC_CPRB BY VALUE EHC_RESERVED END-CALL
COBOL/2 COBOL/6000 Micro Focus COBOL using XM	CALL __RMTREQ USING EHC_RESERVED, cprb_addr CALL "__RMTREQ" USING EHC_RESERVED, cprb_addr
REXX	call RMTREQ cprb_addr



where:

- **retcode** is the return code
- **cprb\_addr** is the address of the CPRB structure
- **EHC\_RESERVED** is a reserved value, an unsigned long integer set to zero. It must be specified exactly as shown. (It is not required in REXX.)

**Examples:** To see the calls in the context of a sample program, refer to the following pages for C and COBOL.

C page 133

```
retcode = RMTREQ (&mycprb, EHC_RESERVED);
```

COBOL page 155

```
CALL "RMTREQ" USING BY REFERENCE EHC-CPRB
                    BY VALUE      EHC-REQUIRED
END-CALL.
```

## Call RMTREQ using the NoWait option

**LANDP for OS/2, LANDP for Windows NT, and LANDP for DOS** (Microsoft Windows 3.1/3.11 or non-Windows) clients can issue a RMTREQ without being suspended until the reply arrives, by using the RMTREQ with the NoWait option. Here, control is returned immediately to the client. The client can issue further RMTREQ calls without having to wait for the reply to the RMTREQ call with the NoWait option.

To use the NoWait option, the client must call the RMTREQ routine and supply the address of the *RMTREQ options control block* (see "The RMTREQ options control block (EHC\_RMTREQ\_OPTS)" on page 31).

**Note:** RMTREQ with the NoWait option cannot be used to call supervisor local functions.

The statements required to call RMTREQ using the NoWait option, are:

Language	Statement
VisualAge C++ Visual C++ C or C Set/2	retcode = RMTREQ (cprb_addr, ehc_rmtreq_opts);
MASM/2	@RMTREQ cprb_addr, ehc_rmtreq_opts
Pascal/2	retcode := RMTREQ (cprb_addr, ehc_rmtreq_opts);

## RMTREQ NoWait call

Language	Statement
COBOL/2	(In the PROCEDURE DIVISION) CALL __RMTREQ USING ehc_rmtreq_opts, cprb_addr
VisualAge COBOL	CALL "RMTREQ" USING BY REFERENCE EHC_CPRB BY REFERENCE EHC_RMTREQ_OPTS END-CALL
REXX	call RMTREQ cprb_addr, ehc_rmtreq_opts

where:

- **retcode** is the return code
- **cprb\_addr** is the address of the CPRB structure
- **ehc\_rmtreq\_opts** is the address of the options control block, described in “The RMTREQ options control block (EHC\_RMTREQ\_OPTS)” on page 31

The client that issued the RMTREQ using the NoWait option obtains the reply by calling the synchronous routine GETRPLY and supplying the address of the *GETRPLY options control block*. The sample below shows how the call is made using the C language:

```
retcode = GETRPLY(cprb_addr, &ehc_getrply_opts);
```

The statements required to call GETRPLY are given in “Call GETRPLY (Get reply)” on page 32.

### Application flow using RMTREQ NoWait

The typical flow of a client program that uses the RMTREQ call with the NoWait option is given below.

1. The client issues the RMTREQ NoWait call, and RMTREQ then returns a *reply handle*. Clients written as OS/2, Windows NT, or Windows 3.1/3.11 applications can show in RMTREQ NoWait a window handle, and a message identity to be posted when the reply arrives.
2. The client resumes its operation. The client can issue other requests or, in LANDP for OS/2, it can wait for the reply by requesting the WM function. Clients written as OS/2, Windows NT, or Windows 3.1/3.11 applications can return to OS/2, Windows NT, or Windows 3.1/3.11 and wait for the posted message.
3. In LANDP for OS/2 and Windows NT, to wait for the reply using the WM function, the client must use as event ID for the WM function the concatenation of *reply handle* + *SPV*.
4. Clients in LANDP for OS/2 and Windows NT that use the WM function are notified that the reply is ready with the event ID described in the previous step. Clients written as OS/2, Windows NT, or Windows 3.1/3.11 applications that have defined

a window handle and message identity using RMTREQ NoWait, receive this message in the window message queue.

5. The client obtains the reply using the GETRPLY routine, passing as a parameter the *reply handle*.

The client may choose to use GETRPLY without using the WM, or waiting for the posted message. Here the client is suspended until the reply arrives, or until the timeout in the GETRPLY CPRB expires.

If the GETRPLY timeout expires, RMTREQ NoWait is cancelled, unless the Keep\_Flag was set to 1 in the GETRPLY. Here the client must issue another GETRPLY to obtain the reply, or cancel the RMTREQ NoWait.

## The RMTREQ options control block (EHC\_RMTREQ\_OPTS)

LANDP for DOS, OS/2, and Windows NT clients that issue a RMTREQ call with the NoWait option must supply the address of the EHC\_RMTREQ\_OPTS control block. The NoWait indicator is specified in this control block.

Clients written as OS/2, Windows NT, or Windows 3.1/3.11 applications can also specify in this control block:

- A *window handle*
- A *message identity* to be posted when the server reply arrives

The use of the EHC\_RMTREQ\_OPTS control block is supported only in LANDP for DOS, OS/2, and Windows NT. Clients that are installed on other LANDP systems must set this option to *EHC\_RESERVED*, which is an unsigned long integer and set to zero.

You can copy the definition of this control block into your program as described in “Including the CPRB and options control block structures” on page 9.

EHC_RMTREQ_OPTS Format			
Field	Type	I/O	Content
struct_size	2 bytes unsigned short	Input	Length of EHC_RMTREQ_OPTS, including this field (12).
reserved	6 bytes	Reserved	Must be set to binary zeros.
nowait_parmad	address (4 bytes)	Input	Address of the NoWait parameter structure (EHC_NOWAIT_PARM). If set to zero, the RMTREQ is then Wait.

## GETRPLY call

EHC_NOWAIT_PARM Format Used With RMTREQ			
Field	Type	I/O	Content
struct_size	2 bytes unsigned short	Input	Length of EHC_NOWAIT_PARM, including this field (16).
reply_handle	2 bytes	Output	reply_handle.
nowait_flags	4 bytes unsigned long	Not used	Not used. Must be set to zeros.
window_handle	handle (4 bytes)	Input	Window that is to receive a message. If set to zero, no automatic posting of the reply occurs and the value in the message identity field is ignored.
message_id	4 bytes unsigned long	Input	Message identity to be posted.

**Example:** To see the call in the context of a sample COBOL program, refer to page 159.

```
SET NOWAIT-PARMAD OF EHC-RMTREQ-OPTS
    TO ADDRESS OF
    EHC-NOWAIT-PARM

CALL "RMTREQ" USING BY REFERENCE EHC-CPRB
    BY REFERENCE EHC-RMTREQ-OPTS

END-CALL.
```

### Call GETRPLY (Get reply)

LANDP for DOS, OS/2, and Windows NT clients (or servers acting as clients) that have issued a RMTREQ call using the NoWait option must later issue a GETRPLY to obtain the reply. The statements required to call GETRPLY, are:

Language	Statement
VisualAge C++, Visual C++ C or C Set/2	retcode = GETRPLY (cprb_addr, ehc_getreply_opts);
MASM/2	@GETRPLY cprb_addr, ehc_getreply_opts
Pascal/2	retcode := GETRPLY (cprb_addr, ehc_getreply_opts);

Language	Statement
COBOL/2	(In the PROCEDURE DIVISION) CALL __GETRPLY USING ehc_getrply_opts, cprb_addr
VisualAge COBOL	CALL "GETRPLY" USING BY REFERENCE EHC_CPRB BY REFERENCE EHC_GETRPLY_OPTS END-CALL
REXX	call GETRPLY cprb_addr, ehc_getrply_opts

where:

- **retcode** is the return code
- **cprb\_addr** is the address of the CPRB structure
- **ehc\_getrply\_opts** is the address of the options control block, described in “The GETRPLY options control block (EHC\_GETRPLY\_OPTS)” on page 34

### CPRB fields required for GETRPLY

LANDP for DOS, OS/2, and Windows NT clients that issue GETRPLY to obtain the reply to a previous RMTREQ issued using the NoWait option must set the CPRB fields shown below. For the CPRB fields not listed below, you can either use the same values that were entered during the original RMTREQ, or you can set these fields to zero.

CPRB fields used by GETRPLY			
Offset	Length	Field Name	Content/Description
X'0C'	2	ehctimeout	Timeout to wait for the reply. Valid values are:  X'0000' (use default LAN timeout) X'0001' to X'7FFF' (timeout in seconds) X'FFFE' (timeout=0) X'FFFF' (no timeout—wait until the reply arrives)
X'1A'	2	ehcrparml	Reply PARMLIST length
X'1C'	4	ehcrparmad	Reply PARMLIST address
X'20'	2	ehcrdatal	Reply DATA length
X'22'	4	ehcrdataad	Reply DATA address

On reply to the GETRPLY, LANDP copies the CPRB that was used with the original RMTREQ, except for the five fields above.

## GETRPLY call

If the length for Reply DATA or Reply PARMLIST is too short to contain the replies from the server, the available length is copied and a nonzero router return code results.

LANDP also provides the router return code and copies the server return code.

### The GETRPLY options control block (EHC\_GETRPLY\_OPTS)

LANDP for DOS, OS/2, and Windows NT clients that issue a RMTREQ call with the NoWait option must use the GETRPLY call to obtain the server reply. The client must supply the address of the options control block *EHC\_GETRPLY\_OPTS*. This control block supplies the *reply handle* of the reply.

You can copy the definition of this control block into your program as described in “Including the CPRB and options control block structures” on page 9.

EHC_GETRPLY_OPTS Format			
Field	Type	I/O	Content
struct_size	2 bytes unsigned short	Input	Length of EHC_GETRPLY_OPTS, including this field (12)
reserved	6 bytes	Reserved	Must be set to binary zeros
nowait_parmad	address (4 bytes)	Input	Address of the NoWait parameter area. Must be the address of a valid EHC_NOWAIT_PARM structure

EHC_NOWAIT_PARM Format Used With GETRPLY			
Field	Type	I/O	Content
struct_size	2 bytes unsigned short	Input	Length of EHC_NOWAIT_PARM, including this field (16)
reply_handle	2 bytes	Input	Handle returned by RMTREQ NoWait
nowait_flags	4 bytes unsigned long	Input	Flags Keep_Flag 1 = Do not cancel RMTREQ NoWait, if the GETRPLY timeout expires
window_handle	handle (4 bytes)	Not used	Not used
message_id	4 bytes unsigned long	Not used	Not used

**Example:** To see the call in the context of a sample COBOL program, refer to page 159.

```

      SET NOWAIT-PARMAD OF EHC-GETRPLY_OPTS
                                TO ADDRESS OF
                                EHC-NOWAIT-PARM

      CALL "GETRPLY" USING BY REFERENCE EHC-CPRB
                        BY REFERENCE EHC-GETRPLY-OPTS

      END-CALL.

```

## Hints

This section gives tips on economical and efficient coding of LANDP client applications.

### Requesting services

After you have initialized the CPRB, following these suggestions should simplify requesting services:

- Use the same area for Request PARMLIST and Reply PARMLIST.
- Use the same area for Request DATA and Reply DATA. Here, the information on request can be overlapped by the information on reply.

For each request you only need to specify the following fields of the CPRB:

- Function code
- Request PARMLIST length
- Request DATA length
- Reply PARMLIST length
- Reply DATA length
- Server name

On return from a request, you need to inspect the following fields:

- Replied PARMLIST length
- Replied DATA length
- Router return code
- Server return code

### Return codes

It is strongly recommended that you always test first the *router return code* and then the *server return code* after requesting a service. If both are zero (X'00000000') the operation was performed successfully.

If the *router return code* is different from X'00000000', the *server return code* is meaningless.

The X'00000000' *router return code* shows successful routing of the RMTREQ function. Any other value shows an error. Here you should see the appropriate section of *LANDP Problem Determination*, to identify the cause of this return code and the possible actions to take.

The X'00000000' *server return code* shows successful completion of the RMTREQ function. Any other value shows an error. Here you should see the appropriate section of the *LANDP Problem Determination* book to identify the cause of this return code and the possible actions to take. Return codes in the *LANDP Problem Determination* book are listed according to the server's operating environment.

---

### Sample application programs

Sample application programs are supplied with LANDP. After LANDP has been installed, the sample programs can be found in the samples sub-directories of the platform directories. If the installation directory is defined as EHC:

DOS samples are in EHC\EHCD500\SAMPLES

OS/2 samples are in EHC\EHCO500\SAMPLES

Windows NT samples are in EHC\EHCN500\SAMPLES

Each samples sub-directory contains an introductory readme.txt file.

**Note:** Five of the sample programs are analysed in Chapter 11, "Sample application programs" on page 127.

For LANDP for AIX, a description file with the name `landpsamples.doc` is provided which describes the LANDP for AIX sample programs, profiles, and specific features. Sample programs and profiles are listed according to the server to which they apply. The purpose of each sample program and details of how each sample program can best be compiled and carried out, are also explained. This file is located in the sub-directory `/usr/lpp/landp`.

---

### LANDP event notification support

A LANDP client application program can request services from servers and be informed when specific events occur in the system. The LANDP event notification support allows servers to notify clients about events and provide a dispatching mechanism to clients. This support includes:

- Clients to get notified of external events
- Servers to process the requests asynchronously to the main process
- Transactions to be processed in parallel

### Types of event

The LANDP event notification support allows clients to be notified of:

- System-generated events (see page 40). These events are generated by the client/server mechanism when, for example, a key is pressed on the keyboard or a timer has elapsed.
- Server-generated events. Servers generate events in a way similar to clients requesting services from a server. Instead of calling RMTREQ, the server calls the RMTAREQ routine with function Z5 (or ZN under LANDP for OS/2 and Windows NT)



to generate the event. After that, the client/server mechanism receives the events generated by servers and passes the information to the destination client.

Event ID

The *event ID* is the identification of one event. It is used by clients to request notification of one specific event, and by the client/server mechanism to notify clients when this event has occurred.

The event ID is a 10-byte record that consists of two fields:

- A 2-byte value, called the *Event Code*
- An 8-byte character string that identifies the originator of the event

Event ID		
Offset	Length	Content
0	2	X'nnnn'    Event code (integer)
2	8	SERVNAME Name of the resource that originates the event. (Eight characters, left-aligned and padded with blanks)  The server name field may contain a <i>resource name</i> , a <i>server name</i> , or an <i>alias name</i> (see "Server names" on page 10)

In the remainder of this book, event IDs are shown as character strings, where the first two characters are the corresponding ASCII codes of the 2-byte event code, and the remaining characters represent the resource name of the originator of the event.

For example, the event ID "KBSPV" consists of an event code X'4B42' (KB) and an originator name "SPV".

On Intel machines, the event code is in byte-reversed form. To assist with the portability of programs across different environments, you should define a structure for an event ID as follows:

```
typedef struct
{
    unsigned short event_code;
    char resource_name[8];
}
EVENT_ID;
```

With this structure, you could assign values in a portable way:

```
#define KB_CODE 0x4b42
...
EVENT_ID one_event;
...
one_event.event_code = KB_CODE;
memcpy (one_event.resource_name, "SPV",8);
...
```

with similar programming for testing a received event notification.

### Receiving event notifications

The client/server mechanism keeps and maintains event information. If application programs specify events for which they want to get notified, the client/server mechanism notifies them about these events. Application programs receive the event notification by requesting the appropriate supervisor local function, all of which are described in more detail in the “Supervisor local functions” chapter of the *LANDP Programming Reference*:

- WM (Wait Multiple)
- AA (Ask for Asynchronous Events)
- SP (Start Posting Events)
- QE (Query Events)
- TP (Stop Posting Events)

WM allows clients to wait for specific events to occur. When a client requests WM control passes to the client/server mechanism. This checks the event information it maintains, and passes back control to the client as soon as a specific event occurs that the client is waiting for.

With the AA function, clients can poll the asynchronous event information maintained by the client/server mechanism for a list of events. The client/server mechanism returns either nothing or the first event that is pending to be notified. WM and AA return the same information in the form of a 10-byte event ID in the Reply PARMLIST.

The functions SP, QE, and TP are used by OS/2 PM, Windows NT, Windows 3.1/3.11, and (with LANDP for AIX) X-Windows application programs. SP requests the client/server mechanism to post an OS/2 PM, Windows NT, Windows 3.1/3.11, or X-Windows message to a window procedure when any of the specified events take place. When the message is received in the window procedure, the application interrogates the client/server mechanism by issuing a QE function. With the TP function, the application can reset any event specified through an earlier SP function.

LANDP for OS/2, Windows NT, and AIX servers can also receive event notifications using their GETREQ routine. For more information, refer to “Additional GETREQ options” on page 52.

### Waiting for multiple events

Clients mainly use the supervisor local function WM (Wait Multiple) to receive event notifications. A client can be in the active or idle state. When a client requests WM, it enters the idle state and gives control to the client/server mechanism.

When a client requests WM it usually specifies a list of events it is waiting for. The client/server mechanism checks the specified event list, and scans its event queue in the order the client has specified them in its event list. If the event queue contains one of the events in the list, the client/server mechanism returns the corresponding event ID. If the event queue does not contain any of the events the client has specified, WM waits until an event occurs that matches one of the events the client has specified.

WM then returns the corresponding event ID to the client. If the client does not specify a list of events, the client/server mechanism scans its event queue in a default order.

WM returns only one event ID every time it is requested. After an event ID is returned to the client, the client/server mechanism removes the corresponding event from its event queue. Exceptions are defined for events generated by communication servers. Here the client must issue a request to the server to read the data. This prompts the server to request the local supervisor to remove the event from the queue.

## Polling asynchronous event information

The AA supervisor local function provides a polling mechanism for clients. This mechanism allows clients to ask for specific events to be notified, without waiting or removing the event from the event queue.

**Note:** You should be wary of using polling mechanisms in a client application because it can be very expensive on system resources.

## Event notification using graphical user interface (GUI) message posting

A GUI application program for LANDP can be developed for the following environments:

- OS/2 Presentation Manager® (PM)
- Windows NT
- Windows 3.1 or 3.11
- X-Windows

These applications have their main process in a window procedure. It is an event-driven process, and the events received by the window procedure are serialized by OS/2 PM, Windows NT, Windows 3.1/3.11, or X-Windows in the application queue. LANDP provides asynchronous notifications to the application program as posted graphical user interface (GUI) messages. This conforms to the conventions of these environments, allowing the use of the environments' standard programming tools.

The message is posted to the window procedure message queue that the application defines using the SP function. The application chooses which window procedure receives and processes the message. The application also defines the message identity posted. In LANDP for AIX this identity is an X-Windows atom, and the event is a ClientMessage event. The *message\_type* contains the *message\_identity*. The *data* field contains the 2-byte *event\_handle* and the 10-byte *event ID*.

The application can use its current parsing of the posted windows messages also for parsing the LANDP events. The *window\_handle* and *message\_identity* pair constitutes the notification posting identifier, because it uniquely identifies both the place to notify and the identity to be notified. For each *window\_handle* and *message\_identity* pair, LANDP maintains a list of events. The application program uses the SP function to add or the TP function to remove events from this list.

LANDP accepts, for OS/2 and Windows NT applications, a maximum of 100 different pairs of *window\_handle* and *message\_identity* per application. For OS/2 and Windows NT applications, LANDP maintains a maximum of 102 events in the event list for each

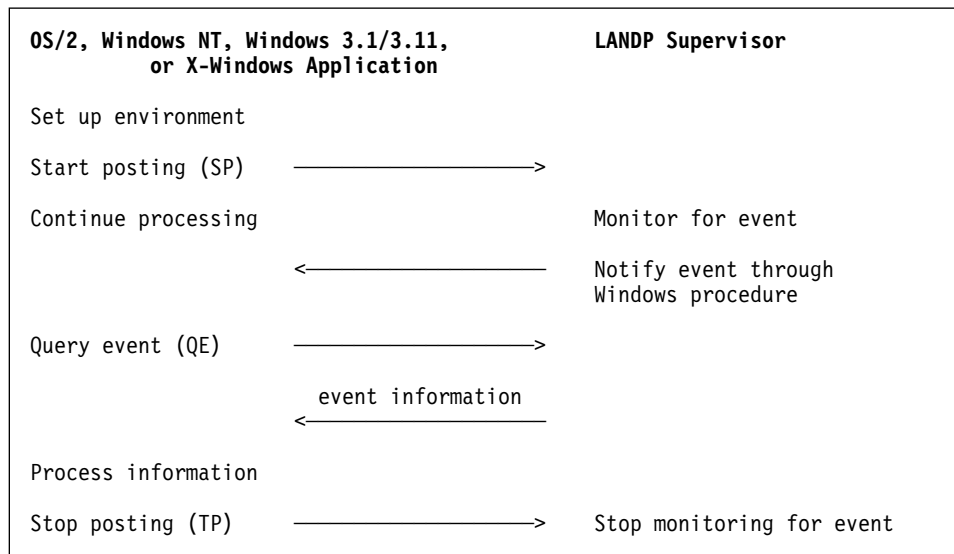
## system events

*window\_handle* and *message\_identity* pair. For Windows 3.1/3.11 applications, LANDP maintains a maximum of 23 events in the event list for each *window\_handle* and *message\_identity* pair.

When an event matching one of the events in the list occurs, LANDP generates an OS/2 or Windows NT message for the window message queue, with the message identity associated in the SP function. LANDP also generates an event handle in the low word of the *message parameter 1*. This event handle can later be used with the QE (query event) function to obtain the event ID of the posted event. LANDP for AIX applications using X-Windows can alternatively use the event ID provided with the ClientMessage event.

Three supervisor local functions are provided to notify events by posting a windows message:

- SP (Start posting events)
- QE (Query event)
- TP (Stop posting events)



## LANDP system events

LANDP system events are events generated by the LANDP client/server mechanism. They provide notifications for:

- User input through the keyboard
- LANDP for OS/2 and Windows NT: user input through the mouse
- Application timers
- OS/2 and Windows NT semaphore events
- Connection and disconnection of LANDP processes and workstations

## Keyboard events

LANDP provides a system event that notifies the client of any key pressed on the keyboard. The event ID is KBSPV. This event is generated every time the supervisor local functions WM or AA are requested and the keyboard buffer contains data to be read.

After the client receives the event ID, usually when the WM ends and the client regains control from the client/server mechanism, the client must read the input from the keyboard using the standard functions of the programming language used. If the application program does not read the input from the keyboard, the input (keystroke) remains in the keyboard buffer. The next request of WM or AA repeats the event.

OS/2 PM or Windows 3.1 application programs cannot use the WM or SP functions to wait for keyboard events. They receive keyboard events through the standard OS/2 event notification.

X-Windows application programs receive events through the standard X-Windows event notification system that uses X-Windows messages. X-Windows applications cannot use the WM function to wait for keyboard events.

## Mouse events under LANDP for OS/2 and Windows NT

Mouse events are notified in the same way as keyboard events. LANDP for OS/2 and Windows NT provide the event ID MOSPV that notifies of a mouse movement or a clicked mouse button (including button press, button release, and double clicks).

After the client receives the event ID, usually when it regains control from the client/server mechanism and the WM ends, the client must use standard operating system functions to determine whether the mouse was moved or a button was clicked.

To receive a mouse event notification, the client must specify the event ID MOSPV in the event list when requesting WM or AA. If the event ID is not specified or WM or AA is requested without passing an event list, mouse events are ignored and therefore not notified to the application program.

OS/2 PM application programs cannot use the WM function to wait for or receive a mouse event. They receive mouse events through the standard OS/2 event notification using OS/2 messages as WM\_MOUSEMOVE or WM\_LBUTTONDOWN. The SP (Start Posting Events) does not support the MOSPV event, because SP only works with OS/2 PM application programs.

## Timer events

Clients use timer events, for example, to control transaction timeouts or to submit jobs at predetermined times. Clients start one of the LANDP system timers, to expire at a given time. When the timer expires the client/server mechanism generates a system event with event ID TxSPV, where *x* identifies the timer and has a value from 1 to 8. The timer event IDs T1SPV through T8SPV are notified to the client following the same procedure (using WM or AA) as for keyboard events.

### **Semaphore events under LANDP for OS/2 and Windows NT**

LANDP for OS/2 and Windows NT applications can use the multiple-tasking facilities that OS/2 and Windows NT provide. Semaphores are used to signal between or synchronize concurrent threads or processes running on OS/2 or Windows NT. A thread or process waits on a semaphore to be cleared by another thread or process.

LANDP for OS/2 and Windows NT provide for the notification of semaphore system events, combining semaphore waiting or signalling with other LANDP events in the same WM function.

The semaphore event notifications can also be received using the functions AA and SP. Using the WM function clients can make most use of the semaphore event notifications.

LANDP for OS/2 and Windows NT support waiting for semaphores, but provide no functions to handle them. The semaphores need to be handled by the client program, which is responsible for creating, opening, clearing, or setting the semaphores. The client can specify semaphore events on the event list that is passed to the client/server mechanism when it requests the WM function. LANDP for OS/2 and Windows NT only accept and wait for 32-bit event semaphores, not for Mutex or MuxWait semaphores.

LANDP for OS/2 and Windows NT associate semaphores, identified by semaphore handles, with Semaphore events. In this way, LANDP for OS/2 and Windows NT allow clients to wait for the required semaphore, together with other events specified on the event list for the WM function. They also allow clients to associate the notified event ID with a semaphore handle. This association is similar to the one used to set up or to clear the LANDP for OS/2 and Windows NT timers. For this association, LANDP for OS/2 and Windows NT provide two supervisor local functions:

- AS (Associate Semaphore with Event)
- DS (Disassociate Semaphore from Event)

### **Process connection and disconnection events**

The LANDP client/server mechanism supervises the availability of processes in the LANDP workgroup. For example, a LANDP client application program receives notification of events related with the availability of servers that the client has accessed. LANDP generates two system events:

- Process connection event (with event code X'2626' or '&&')
- Process disconnection event (with event code X'2A2A' or '\*\*')

The event IDs consist of the event code and the name of the server (left-justified and padded with blanks) that has been connected or disconnected. If the server name corresponds to an alias name, LANDP for OS/2, Windows NT, and AIX notify with the alias name, and generate as many process connection or process disconnection events as there are resource names defined as aliases for the server.

Every time a server successfully completes its initialization and responds to the initialization request, LANDP makes it available to clients and notifies them about the connection of that server. The client/server mechanism generates in turn the process connection event with the name of the server. The process connection and

disconnection events are received by the clients and servers located in workstations, with access to the server being connected or disconnected.

Similarly, when a server is unloaded, or the workstation containing that server disconnects, LANDP releases the server, and each client/server mechanism (in each workstation that has access to that server) generates the process disconnection event with the name of the server. When the unload or disconnection affects a complete workstation, the client/server mechanism generates a process disconnection event for each server that was accessible in that workstation.

---

### LANDP for DOS and Windows 3.1/3.11 support

Up to eight concurrent applications (both Windows 3.1/3.11 and non-Windows) can request LANDP services, each standard LANDP for DOS application running in its own virtual DOS machine, and the Windows 3.1 applications running in the system virtual machine.

Application programs for LANDP for DOS can also be developed as Windows 3.1 applications. Such Windows 3.1 programs use the LANDP common API by calling the RMTREQ routine.

For Windows 3.1 applications, asynchronous events, such as communications events or the connection or disconnection of servers, can be handled following the standard Windows 3.1 protocol, by queueing Windows 3.1 messages to the application queue. Clients running under Windows 3.1 request the SP, QE, and TP local functions in the same way that an OS/2 client does (see “Event notification using graphical user interface (GUI) message posting” on page 39). This standard Windows 3.1 protocol “replaces” the use of the WM function, which is not recommended because it can cause other Windows 3.1 applications to be stopped and unpredictable results might occur. However, if you choose to use WM instead of SP, QE, and TP, you should set a timeout value of zero. In this way, control immediately returns, allowing you to poll for any desired event. Better performance results from handling timer and keyboard events through the standard Windows 3.1 interface.

### Running standard LANDP for DOS applications

The LANDP for DOS kernel (supervisor and servers) runs alone under Microsoft Windows 3.1 in a virtual DOS machine (VDM), and has access to the entire VDM machine memory space.

Standard LANDP for DOS applications run in other VDMs, and can therefore use more memory than if they were running in a pure DOS environment. Also, multiple LANDP Windows 3.1 applications are supported.

When working with Windows 3.1, you can have the following running at the same time:

- LANDP for DOS kernel in VDM1
- LANDP non-Windows applications in VDM1, VDM2, ... VDMn
- Non-LANDP non-Windows applications in other virtual machines
- Windows applications (LANDP or non-LANDP) in the system virtual machine

## DOS and Windows 3.1/3.11 support

To communicate from a LANDP application to the LANDP for DOS kernel, you may require four modules supplied with LANDP for DOS in the LANDP subdirectory of the workstation where LANDP is installed. They are:

### **EHCVMDS.386**

This is a Windows 3.1 virtual device driver (also called a virtual machines services device) that must be included in the Windows 3.1 SYSTEM.INI file in the [386Enh] section as follows:

```
device=drive:\path\ehcvmsd.386
```

This module is always required.

### **EHCWGMDI.EXE**

This interface (called a global memory data interface) must be loaded before Windows 3.1, and reserves a buffer in global memory to contain the data areas of the requests from LANDP applications. This buffer supports multiple concurrent requests.

To load EHCWGMDI, enter:

```
EHCWGMDI /L:xx
```

where xx is a value in kilobytes that is the maximum length of the Request DATA, Reply DATA, and PARMLIST fields in the CPRB used by all the installed applications. It can take any value from 1 through 56. The default is 4. Calculate the maximum length of DATA plus PARMLIST fields and add 156 bytes. This module is always required.

### **EHCWVDMI.EXE**

This interface (the virtual DOS machine interface) must be loaded before the LANDP application in the corresponding VDM. It transfers the request from the application to the LANDP for DOS kernel VDM. This interface serves either the common API, or the LANDP 3270 emulator API and the 3270 high-level language API. This module must be included in the WINSTART.BAT file, so that it can service requests from LANDP Windows 3.1 applications. Resident modules included in this file remain accessible to all the Windows 3.1 applications. This module is always required.

### **EHCWIN.DLL**

This Windows 3.1 DLL exports the LANDP RMTREQ protected mode call from Windows 3.1 applications to a real interrupt call using DOS protected mode interface (DPMI) services provided by Windows 3.1. The call is then processed by the EHCWVDMI module, just like any other call from a standard DOS application in a virtual DOS machine. This module is required if you are running LANDP Windows 3.1 applications.

Other files, such as program information files (PIFs) may be required. Some are generated during LANDP customization, and can include the four files named above. For example, the SYSTEM.ADD file contains the device sentence for ehcvmsd.386. The EHCPREV.BAT file (which must be run before Windows 3.1) contains the call to ehcwgmdi and appropriate LOADER calls. EHCDO SVM.BAT contains calls to ehcwvdm and AUTOUSER.BAT.

See the *LANDP Installation and Customization* book for more information.



### Requirements for your standard LANDP for DOS applications

LANDP clients and user servers may require a few changes to run under Microsoft Windows 3.1. Those changes are:

- If the client or user server uses the CPRB/SEND\_REQUEST or the COMMAREA/DATAREA interface, it must be migrated to the LANDP common API. The common API allows you to write programs portable at the source code level between LANDP for DOS, OS/2, and Windows NT.
- The Wait for Asynchronous Events (WM) supervisor function must be used instead of the Exit to Idle Status (EX) supervisor function. Because the WM function allows waiting for a specific event, it is no longer necessary to use the Activate Dispatcher Entries (SA) and Deactivate Dispatcher Entry (RA) supervisor functions.
- The Initialize (IN) and Disconnect an application program (EJ) functions are mandatory. If you are using the SNA server the Connect (CN) function is also mandatory.
- When starting both LANDP for DOS and the client automatically through Microsoft Windows 3.1, the client can start before LANDP for DOS has ended its loading procedure. Here the client should check for LANDP availability.

### Unloading LANDP for DOS

To unload LANDP for DOS, the FREE program must be run from the application's VDM after EHCWVDMI. Some LANDP modules are removed from memory, if the application performs an Unload LANDP for DOS (ES) supervisor function. The FREE program removes only the LANDP modules loaded after Microsoft Windows 3.1.

### Building Windows 3.1 applications that request LANDP for DOS services

If you want to write Windows 3.1 applications that request LANDP for DOS services, you must follow these guidelines:

- Link your applications with the Dynamic Link Library EHCWIN.DLL, which is included on the LANDP CD-ROM or diskette images.
- Your applications must use the common RMTREQ routine to call the LANDP interface.
- The Initialize (IN) function is mandatory.
- Asynchronous LANDP events are notified to your application through the standard Windows 3.1 protocol. Windows 3.1 messages are queued in the application queue.
- You must not use the Wait for Asynchronous Events (WM) function call because it may conflict with the Windows 3.1 dispatching mechanism, unless the timeout interval is zero.
- To deal with asynchronous events, your Windows 3.1 applications must use the following supervisor function calls:
  - SP (Start posting events)
  - QE (Query event)
  - TP (Stop posting events)

## DOS and Windows 3.1/3.11 support

- Your Windows 3.1 application must deal with system events (keyboard, mouse, timers, and so on) through the standard Windows 3.1 protocol. These events are not available to LANDP.
- Load EHCWGMDI before loading Microsoft Windows 3.1 and the LANDP for DOS kernel in a VDM.
- Build your WINSTART.BAT file to include the EHCWVDMI module.
- Windows 3.1 applications may use the RMTREQ call with the NoWait option (see “Call RMTREQ using the NoWait option” on page 29) and GETRPLY (see “Call GETRPLY (Get reply)” on page 32) to access servers whose response is slow. In this way, other Windows 3.1 applications are not stopped.

## Performance considerations

You should investigate the following Windows 3.1 settings to achieve optimum system performance, depending on your specific environment:

- MinTimeSlice
- WinTimeSlice
- KeyIdleDelay
- KeyBoostTime

Also, the Detect\_Idle\_Time setting of both the LANDP kernel and the LANDP application program information files (PIFs) should be set OFF in most cases to get better performance.

The EHCWKDE program should be executed after the LANDP supervisor in the kernel VDM to enhance Windows 3.1 dispatching.

## Restrictions

The implementation of LANDP for DOS with Microsoft Windows 3.1 involves some restrictions:

- The IBM 3270 emulator LAN management program is not supported. This means that alerts generated by LAN adapters are not collected by the system manager server.
- The IBM X.25 Interface Co-Processor/2 adapter used by the X.25 data link control (X25DLC2) server is not compatible with Microsoft Windows 3.1.
- The IBM 4717 magnetic stripe reader and the IBM 4718 PIN pad devices are not supported. (The IBM 4777 and 4778 are supported, attached to a serial port.)

## Chapter 3. Writing your own server programs

A LANDP server is a program that uses LANDP routines to receive, perform, and reply to requests coming from a LANDP application program (client).

A server application program can also act as a client. If you write such programs, read Chapter 2, "Writing client programs" on page 27.

All server programs must be prepared to process the requests that are listed in Table 2. The client/server mechanism of LANDP directs these requests to a server.

*Table 2. System Request Function Codes, which must be accepted by All Servers. The first and second columns give the function code and the name of the system request. The third column shows the operating environment of the server, as explained in "Operating environments" on page xiv. "026N" means that it applies to LANDP for DOS, OS/2, Windows NT, and AIX servers. The last column refers to the page where you can find the function described.*

Function code	Description	Env.	Page
<b>&amp;&amp;</b>	Process connection	026N	63
<b>&amp;&amp;</b>	Workstation connection	026N	62
<b>**</b>	Process disconnection	026N	61
<b>**</b>	Workstation disconnection	026N	61
<b>ES</b>	End of service	026N	57
<b>IN</b>	Server recognition	026N	58
<b>TT</b>	Timer-generated request	026N	59

### Structure of a server

The structure of a LANDP server is determined by the sequence in which it calls the common API routines:

1. After the server is loaded and has received control, it calls the SRVINIT routine to register itself to LANDP. This notifies the client/server mechanism that the server is loaded.

LANDP for DOS servers are automatically converted into terminate and stay resident (TSR) programs that remain in memory until LANDP is unloaded.

2. The server calls the GETREQ routine to obtain a pending request. The server provides an address to which the GETREQ routine passes the CPRB information sent by the client (or server acting as client). Then the server takes the request and does the processing required. The first request that a server receives through the GETREQ routine is the IN function. This shows that the SRVINIT routine (which operates asynchronously) has completed and that the client/server mechanism recognizes the server as being loaded. See "Server recognition (IN function)" on page 58.

## structure of a server

3. After the server has received a request through the GETREQ routine, it may call operating system functions to satisfy this request (for example, opening, reading, writing, and closing files).

The server can also use RMTREQ to send requests to other servers, or RMTAREQ to issue asynchronous notifications (functions Z4, Z5, and, under LANDP for OS/2 and Windows NT, ZN) In both cases, it should use a separate CPRB so that the initial one is not modified. You should save the Process ID and Workstation ID fields of the incoming CPRB, so that you can specify them as the destination for any Z5 or ZN function that you subsequently use.

4. The server must be prepared to receive, and reply to, system requests (see “Receiving system requests” on page 56). In all requests, the originator resource name is “SPV”, except for process connection (&&) and process disconnection (\*\*) where the originator resource name is the name of the process. The server can perform any required processing when receiving a system request. The default action associated with a system request is to simply send a reply to it.
5. Every request that the server receives through the GETREQ routine is answered using the RMTRPLY routine. The server supplies to the RMTRPLY routine the address of the CPRB used with the GETREQ routine. Before the server calls RMTRPLY, it updates the required fields in the CPRB, and the Reply PARMLIST and Reply DATA areas.
6. The server continues to call GETREQ and to process any requests until it receives an End of Service (ES) request. On receiving the ES request, the server releases its resources and terminates processing. This includes restoring any interrupt vectors that were chained during the IN request.

---

## Invoking the common API

The link routines (see “Compiling and linking your application program” on page 13 and “Including the CPRB and options control block structures” on page 9) should be included in any user server program.

The main LANDP calls used in servers are: SRVINIT, GETREQ, RMTRPLY, and RMTAREQ. They are described later in this chapter.

If a server is acting as a client, it may need to call RMTREQ (see “Call RMTREQ (Remote request)” on page 28) or GETRPLY (see “Call GETRPLY (Get reply)” on page 32).

## CPRB fields used and set by servers

The following table shows the CPRB fields that must be returned by server programs when replying to a service request.

CPRB Fields Used by LANDP			
Offset	Length	Field name	Content/Description
X'28'	4	ehcsevrvc	Server return code
X'2C'	2	ehcrepldplen	Replied PARMLIST length

CPRB Fields Used by LANDP			
Offset	Length	Field name	Content/Description
X'2E'	2	ehcrepldden	Replied DATA length

## Call SRVINIT (server initialization)

SRVINIT is the first routine that a server calls. It initializes the server and notifies the client/server mechanism that the server has been loaded. If the server tries to call GETREQ or RMTRPLY before SRVINIT has completed, it receives an error message.

Under LANDP for OS/2, Windows NT, and AIX, when a SRVINIT call is issued, a list of service names can be supplied containing the services that the server wishes to register and therefore make available to clients. See "Additional SRVINIT options" on page 50 for further information. Alternatively, under LANDP for OS/2, Windows NT, and AIX, there may be no list of service names supplied. Here, the server's executable file name is registered to LANDP.

Under LANDP for DOS, a SRVINIT call can *only* be issued without a list of service names, and the server executable file name is registered to LANDP.

Table 3 shows the calling syntax for SRVINIT. The SRVINIT return code and parameter fields are explained after the table.

Table 3. SRVINIT calling syntax	
Language	Statement
VisualAge C++ Visual C++ C, C/2, and C/6000	retcode = SRVINIT (size, init_error, process_request, ehc_srvinit_opts);
Macro Assembler/2™	@SRVINIT size, init_error, process_request, ehc_srvinit_opts
Pascal/2 and Pascal/6000	retcode := SRVINIT (size, init_error, process_request, ehc_srvinit_opts);
COBOL/2 and COBOL/6000	call __SRVINIT using ehc_srvinit_opts, process_request, init_error, size
VisualAge COBOL	CALL "SRVINIT" USING BY VALUE SIZE BY VALUE INIT_ERROR BY VALUE PROCESS_REQUEST BY REFERENCE EHC_SRVINIT_OPTS END-CALL
REXX	call SRVINIT size, init_error, process_request, ehc_srvinit_opts

In Table 3 on page 49, the SRVINIT call parameters are as follows:

- `retcode` is an unsigned short integer. It holds any return code on completion of the routine.
- `size` is an unsigned short integer. For LANDP for DOS servers, it represents the size of the server in paragraphs (one paragraph equals 16 bytes). When SRVINIT is successfully completed the server is a TSR with that size.

Using LANDP for OS/2, Windows NT, and AIX, this parameter is ignored. It is supported for compatibility reasons.

- `init_error` is an unsigned short integer. It shows if an error was found during the initialization phase of the server. If `init_error` is zero, no errors were found. If `init_error` is not zero, an error procedure is processed and the server is terminated. The value of `init_error` is used as the return code of the loader statement in AUTOFBSS and can be checked on the command line by using `ERRORLEVEL`. Values of `init_error` for user-written servers should be in the range X'E0' to X'EF'. For more information, see *LANDP Problem Determination*.
- `process_request` is the far address of a procedure (in a 16-bit environment). For LANDP for DOS servers, this procedure receives and replies to requests from clients.

Using LANDP for OS/2, Windows NT, and AIX, this parameter is ignored. It is supported for compatibility reasons.

- `ehc_srvinit_opts` contains additional parameters for the routine. If not used, it should be set to `EHC_RESERVED`. `ehc_srvinit_opts` is the pointer to a control block called `EHC_SRVINIT_OPTS`. LANDP for DOS does not support this parameter and it should therefore be set to `EHC_RESERVED`. `EHC_RESERVED` is an unsigned long integer. It is a reserved value and should be set to zero. `EHC_RESERVED` is a predefined constant in LANDP, so the calling process must use the constant as defined here.

### Additional SRVINIT options

LANDP for OS/2, Windows NT, and AIX servers that make several service names available must register these service names to LANDP using the SRVINIT options control block **`EHC_SRVINIT_OPTS`**. Table 4 on page 51 shows the structure of `EHC_SRVINIT_OPTS`.

Table 4. EHC_SRVINIT_OPTS control block format			
EHC_SRVINIT_OPTS Format			
Field	Type	I/O	Content
struc_size	4 bytes unsigned long	Input	Length of EHC_SRVINIT_OPTS, including this field (8)
service_name_list	address (4 bytes)	Input	Address of a null-terminated (ASCIIIZ) string containing the service names that the server wishes to be registered
<b>service_name_list:</b> Each service name is separated by a semicolon (;). The maximum length of a service name is 8 bytes. The format of the <i>service_name_list</i> is: Service_Name[;Service_Name2;...;Service_Namen]\0			

**Examples** To see the calls in the context of a sample program, refer to the following pages.

C++ page 143

```
init_error = 0;
optionsptr = EHC_RESERVED;
retcode=SRVINIT( size, init_error,routine, optionsptr);
```

COBOL page 167

```
CALL "SRVINIT" USING BY VALUE TWO-BYTES
                     BY VALUE INIT-ERROR
                     BY VALUE FOUR-BYTES
                     BY VALUE EHC-RESERVED
```

## Call GETREQ (get request)

The server calls the GETREQ routine to obtain pending requests, stored in a first-in first-out (FIFO) way.

Table 5 shows the calling syntax for GETREQ. The GETREQ return code and parameter fields are explained after the table.

Table 5 (Page 1 of 2). GETREQ syntax	
Language	Statement
VisualAge C++ Visual C++ C, C/2, and C/6000	retcode = GETREQ (cprb_addr, ehc_getreq_opts);
Macro Assembler/2	@GETREQ cprb_addr, ehc_getreq_opts
Pascal/2 and Pascal/6000	retcode := GETREQ (cprb_addr, ehc_getreq_opts);

## GETREQ call

Table 5 (Page 2 of 2). GETREQ syntax	
Language	Statement
COBOL/2 and COBOL/6000	call __GETREQ using ehc_getreq_opts, cprb_addr.
VisualAge COBOL	CALL "GETREQ" USING BY REFERENCE EHC_CPRB BY REFERENCE EHC_GETREQ_OPTS END-CALL
REXX	call GETREQ cprb_addr, ehc_getreq_opts

In Table 5 on page 51, the GETREQ return code and call parameters are as follows:

- retcode is the return code from the routine.
- cprb\_addr is the address where the server has defined the CPRB structure.
- ehc\_getreq\_opts is the address of a control block **EHC\_GETREQ\_OPTS** used to specify additional parameters to the routine. If not used, it should be set to **EHC\_RESERVED** or, in the case of REXX, omitted. LANDP for DOS servers do not support this parameter, so it has to be set to **EHC\_RESERVED**. For more information, see “Additional GETREQ options.”

The server calls GETREQ to check the contents of its entry queue. When an entry is found, GETREQ passes the CPRB information defined by the client to the server.

### Additional GETREQ options

For LANDP for OS/2, Windows NT, and AIX servers, GETREQ extracts from the server queue the oldest request matching the specified conditions. These conditions are specified by the **ehc\_getreq\_opts** parameter of GETREQ. This parameter represents the address of a control block with the structure shown in Table 6 on page 53.



Table 6. EHC\_GETREQ\_OPTS structure

EHC_GETREQ_OPTS Format			
Field	Type	I/O	Content
struct_size	2 bytes unsigned short	Input	Length of EHC_GETREQ_OPTS, including this field (14)
reserved	6 bytes	Reserved	Must be set to binary zeros
eventlist_size	2 bytes unsigned short	Input	Length of the list pointed to by the following field
eventlist	address (4 bytes)	Input	Address of a list with the same format as the list that specifies the events to wait for in the WM (wait multiple) function
<b>Note:</b> For a detailed description of the list used with the WM function, see <i>LANDP Programming Reference</i> , chapter entitled “Supervisor local functions”.			

The following cases are possible:

- EHC\_GETREQ\_OPTS is set to *EHC\_RESERVED*. No control block is specified. GETREQ receives requests with *ehcverb\_type* = 1 (normal synchronous requests). Also, two special client/server mechanism requests with *ehcverb\_type* = 3 can also be received:
  - && (Process Connection)
  - \*\* (Process Disconnection)
- EHC\_GETREQ\_OPTS points to a control block where *eventlist* and *eventlist\_size* are both zero. A control block is specified, but the relevant fields are zero. GETREQ behaves as explained in the previous item.
- EHC\_GETREQ\_OPTS points to a control block where *eventlist* = 1 but *eventlist\_size* = 0. The list can be considered as the null list. GETREQ gets entries in the queue with *ehcverb\_type* = 1 only. These are normal requests from clients sent by the RMTREQ routine and the client/server mechanism requests LAN Connection Established (&&), LAN Connection Lost (\*\*) and the function Timer Generated Request (TT).
- EHC\_GETREQ\_OPTS points to a control block where *eventlist* and *eventlist\_size* are both nonzero. The pointer points to a valid list. GETREQ receives all entries with *ehcverb\_type* = 1, together with those entries with *ehcverb\_type* = 3 which are specified in the list. For more information about the verb type, see note 2 on page 7.

**Examples** To see the calls in the context of a sample program, refer to the following pages.

C++ page 143

```
retcode = GETREQ(&mycprb, EHC_RESERVED);
```

## RMTRPLY call

COBOL page 167

```
CALL "GETREQ" USING BY REFERENCE EHC-CPRB
                      BY VALUE      EHC-RESERVED
END-CALL
```

### Call RMTRPLY (remote reply)

RMTRPLY returns the result of the processing of the request received through GETREQ to the client (or server acting as client), after having updated the fields specified in “CPRB fields used and set by servers” on page 48. Table 7 shows the calling syntax for RMTRPLY. The GETREQ return code and parameter fields are explained after the table.

Table 7. RMTRPLY syntax.	
Language	Statement
VisualAge C++ Visual C++ C, C/2, and C/6000	retcode = RMTRPLY (cprb_addr, EHC_RESERVED);
Macro Assembler/2	@RMTRPLY cprb_addr, EHC_RESERVED
Pascal/2 and Pascal/6000	retcode := RMTRPLY (cprb_addr, EHC_RESERVED);
COBOL/2 and COBOL/6000	call __RMTRPLY using EHC_RESERVED, cprb_addr.
VisualAge COBOL	CALL "RMTRPLY" USING BY REFERENCE EHC_CPRB BY VALUE EHC_RESERVED END-CALL
REXX	call RMTRPLY cprb_addr

In Table 7, the SRVINIT call parameters are as follows:

- retcode is the return code from the routine.
- cprb\_addr is the address of the CPRB which was received through a GETREQ.
- EHC\_RESERVED is reserved for future use. It must be specified exactly as explained or, in the case of REXX, omitted.

The CPRB that is pointed to by *cprb\_addr* must be the same as the one received by a previous GETREQ.

For LANDP for OS/2, Windows NT, and AIX servers, the thread issuing the RMTRPLY does not have to be the same as the one that issued the GETREQ. The server has to call RMTRPLY to reply to each request received using GETREQ. But the server can first receive some requests and *then* reply to each using RMTRPLY. This means that LANDP

for OS/2, Windows NT, and AIX servers can *collect* requests and process them in batches.

LANDP for DOS servers must reply to the requests in the same order as they received the requests. The client/server mechanism sends the reply to the corresponding client. In contrast to LANDP for OS/2, Windows NT, and AIX, servers under LANDP for DOS can have only one direct line of processing. The procedure calling RMTRPLY is always the same as the one calling GETREQ. GETREQ can only be called if the previously received request has been performed and replied to.

A CPRB received from GETREQ cannot be modified before calling RMTRPLY, except for the *ehc\_servrc*, *ehcrepldplen*, and *ehcreplddlen* fields. This means that a CPRB received from GETREQ cannot be reused (for other GETREQs or RMTREQs, for example) until RMTRPLY completes successfully.

### Event notification and cancellation with RMTRPLY

By setting the *ehc\_flags* and *ehc\_event\_id* fields in the CPRB, you can send or cancel asynchronous event notifications in a reply, using RMTRPLY. Here you cannot send event data with the reply. For more information, see the “Extended asynchronous event notification (ZN function)” section in the “Supervisor local functions” chapter of the *LANDP Programming Reference*.

**Examples** To see the calls in the context of a sample program, refer to the following pages.

C++ page 145

```
retcode = RMTRPLY(&mycprb, EHC_RESERVED);
```

COBOL page 167

```
CALL "RMTRPLY" USING BY REFERENCE EHC-CPRB
                     BY VALUE      EHC-RESERVED
END-CALL
```

## Call RMTAREQ (remote asynchronous request)

The RMTAREQ routine is used to:

- Start, stop, and control TT requests using the asynchronous function Z4. Z4 is explained in *LANDP Programming Reference*, chapter entitled “Supervisor local functions”, and TT in “Timer-generated request (TT function)” on page 59.
- Signal asynchronous events to clients. Use the Z5 or ZN (under LANDP for OS/2 and Windows NT) asynchronous request, which are explained in “Sending asynchronous events” on page 64.

Table 8 on page 56 shows the calling syntax for RMTRPLY. The GETREQ return code and parameter fields are explained after the table.

## RMTAREQ call

Table 8. RMTAREQ syntax	
Language	Statement
VisualAge C++ Visual C++ C, C/2, and C/6000	retcode = RMTAREQ (cprb_addr, EHC_RESERVED);
Macro Assembler/2	@RMTAREQ cprb_addr, EHC_RESERVED
Pascal/2 and Pascal/6000	retcode := RMTAREQ (cprb_addr, EHC_RESERVED);
COBOL/2 and COBOL/6000	call __RMTAREQ using EHC_RESERVED, cprb_addr.
VisualAge COBOL	CALL "RMTAREQ" USING BY REFERENCE EHC_CPRB BY VALUE EHC_RESERVED END-CALL
REXX	call RMTAREQ cprb_addr

In Table 8, the RMTAREQ return code and parameters are as follows:

- retcode is the return code.
- cprb\_addr is the address of the CPRB passed to the routine.
- EHC\_RESERVED is reserved. It must be specified exactly as previously explained or, in the case of REXX, omitted.

**Example** To see the call in the context of a sample program, refer to the following page.

C++ page 145

```
retcode = RMTAREQ (&myacprb, EHC_RESERVED);
```

---

## Receiving system requests

Besides the requests from client application programs, servers also receive requests and notifications from the client/server mechanism. There are several requests and notifications that apply to servers running in LANDP for DOS, OS/2, Windows NT, and AIX:

- LAN session services that have lost the connection with another workstation in the LAN (for example, because a workstation has been turned off). The client/server mechanism can thereby prevent servers from sending requests to a workstation that has lost communication with the LANDP workgroup.
- A server that has been recognized by, and synchronized with, the client/server mechanism.
- A new workstation or LANDP process that has been connected to the LAN.

These requests are queued in the server entry queue in the same way that client requests are queued in the client/server mechanism entry queue. Servers must be able to manage these requests issued by the client/server mechanism. As for any other type of LANDP requests, the server reads the request by calling GETREQ and sends a response by calling the RMTRPLY routine. No other routines or procedures need to be invoked.

Servers can request functions from the client/server mechanism, for example, to start, cancel, or modify the frequency of periodic timer requests (TT) sent by the client/server mechanism to the server. Clients and (for LANDP for OS/2, Windows NT, and AIX) servers acting as clients, can request functions from the client/server mechanism.

The following sections describe these requests in more detail.

## End of service (ES function)

The client/server mechanism issues this request to the resident servers because it received an ES from a client or an operator. This function is the counterpart of the IN function. Servers should undo whatever they did in the IN function. LANDP for DOS servers should also restore interrupt vectors that were chained during the IN function.

After having received this request, the server should send the reply and then terminate processing.

Note the difference between a client calling RMTREQ to issue an ES function and a server receiving an ES function through the GETREQ routine. In the first case, the client tells the *client/server mechanism* that it wants to unload LANDP in the machine where the client resides. In the second case, the client/server mechanism generates the request to tell the *server* to unload itself. The server must then reply to the request by calling the RMTRPLY routine providing the same CPRB. The client/server mechanism releases the server the next time the server calls GETREQ.

Table 9 shows the contents of the CPRB for an ES function request.

<i>Table 9. ES function, CPRB contents</i>	
<b>CPRB Field</b>	<b>Content/Description</b>
Verb type	X'01'
Function code	ES
Request DATA length	0
Request PARMLIST length	0
Reply DATA length	0
Reply PARMLIST length	0
Replied DATA length	0
Replied PARMLIST length	0
Originator resource name	SPV
Server name	SERVNAME

## server recognition (IN function)

### Server recognition (IN function)

As its first GETREQ call, the server receives an IN request from the client/server mechanism. This tells the server it has been recognized by, and synchronized with, the client/server mechanism.

When it receives the IN request, the server should initialize its own environment. LANDP for DOS servers must also chain into any interrupt required during the LANDP session.

The client/server mechanism makes the server available for clients when the server replies with a successful return code. Any other return code causes the client/server mechanism to terminate the server. For LANDP for DOS, when a server returns a nonzero return code, the least significant word of the return code is displayed and LANDP for DOS is unloaded (including all the other servers in the same machine).

Note the difference between a client calling RMTREQ to issue an IN function and a server receiving an IN request through GETREQ from the client/server mechanism. In the first case, the client informs the *client/server mechanism* that it wants to start using a server. In the second case the client/server mechanism generates this request to inform the *server* that it has been recognized and can receive requests when the server returns a successful reply to this client/server mechanism request. IN also specifies in the CPRB the destination workstation ID where the server is installed.

Table 10 shows the contents of the CPRB for an IN function request.

Table 10. IN function, CPRB contents	
CPRB Field	Content/Description
Verb type	X'01'
Function code	IN
Request DATA length	0
Request PARMLIST length	0
Reply PARMLIST length	2
Reply DATA length	0
Server return code	X'nnnnnnnn' (see note 1.)
Replied PARMLIST length	0 or 2 (see note 2.)
Replied DATA length	0
Originator resource name	SPV
Destination workstation ID	C'cc'
Server name	SERVNAME

Reply PARMLIST Values		
Offset	Length	Content
0	2 bytes	<ul style="list-style-type: none"> <li>X'0000': no timer generated requests (TT) are issued</li> <li>X'xxxx': interval time. Multiples of 50 milliseconds for TT function (see note 2.)</li> </ul>

## Notes:

1. The server must set the server return code. Any server return code other than X'00000000' shows an error.

When an error occurs, the server return code is displayed and the client/server mechanism considers the server as being off-line. The server then receives an ES request and is released after the next GETREQ (see “End of service (ES function)” on page 57).

The return codes that a server provides at loading time differ from the return codes provided by the client/server mechanism. This allows you to determine the origin of the error.

2. The server can request periodical timer requests from its client/server mechanism. To do this, you need to:
  - a. Define the Replied PARMLIST length as X'0002'
  - b. Specify the interval time in the first word of the Reply PARMLIST

The interval time is specified in multiples of 50 milliseconds. For example, X'0003' initiates a periodical request from the client/server mechanism each 150 milliseconds.

## Timer-generated request (TT function)

Besides the requests that a server receives from clients (or servers acting as clients), it can make periodical requests of the client/server mechanism.

This is useful if a server needs to perform any process periodically, independently of client requests.

The client/server mechanism issues the control function TT to the server entry queue, each time the specified interval time expires.

The server can request or cancel TT requests and start or modify the interval time:

- On the reply to the IN request.
 

When the server receives the first request (the IN function), it can specify an interval time in the Reply PARMLIST and return it with a zero server return code. This tells the client/server mechanism to start sending TT requests to the server at the specified intervals of time.
- By sending the asynchronous request Z4 (see the “asynchronous request-asking for TT request (Z4 function)” section in the “Supervisor local functions” chapter of the *LANDP Programming Reference*.

## TT (timer-generated request)

With this function you can start, stop, or modify the interval time at which TT requests are sent to the server.

- On the reply to a previous TT request.

As with the reply to the IN request you can specify in the Reply PARMLIST an interval time at which TT requests are to be sent.

**Note:** Because of the server overhead reading the queue, and the characteristics of a multiple-tasking system, the time at which servers receive the request may not be completely predictable.

When a server receives TT requests periodically the server reads the requests (calling the GETREQ routine) and does not receive another TT until it has sent the RMTRPLY for the current TT.

Table 11 shows the contents of the CPRB for an IN function request.

Table 11. TT function, CPRB contents	
CPRB Field	Content/Description
Verb type	X'01'
Function code	TT
Request DATA length	0
Request PARMLIST length	2
Reply DATA length	0
Reply PARMLIST length	2
Replied DATA length	0
Replied PARMLIST length	0 or 2
Originator resource name	SPV
Server name	SERVNAME

Request PARMLIST Values		
Offset	Length	Content
0	2 bytes	X'nnnn' Current interval time. Multiples of 50 milliseconds.

Reply PARMLIST Values		
Offset	Length	Content
0	2 bytes	X'nnnn' New interval time. Multiples of 50 milliseconds.

On reply, the server can update the interval time for receiving TT requests by modifying the content of the first word of the Reply PARMLIST



## Workstation disconnection (\*\* function)

The client/server mechanism issues this request to notify its resident servers that communication with another workstation is lost. This may happen because LANDP was unloaded in that workstation or LANDP for AIX system, or because that workstation or system was turned off.

The notification about lost communication allows the servers to cancel any pending request and to release any reserved resource belonging to any process residing in the workstation or system with which communication has been lost.

The server must check whether there was a process in the disconnected workstation or LANDP for AIX system, using any resource managed by the server at the time the connection-lost signal is received. If this is the case, the server must release these resources to make them available to other processes.

Table 12 shows the contents of the CPRB for a workstation disconnection (\*\*) function request.

<i>Table 12. Workstation disconnection (**) function, CPRB contents</i>	
<b>CPRB Field</b>	<b>Content/Description</b>
Verb type	X'01'
Function code	**
Request DATA length	0
Request PARMLIST length	0
Reply DATA length	0
Reply PARMLIST length	0
Replied DATA length	0
Replied PARMLIST length	0
Originator workstation ID	C'cc'—the workstation with which communication was lost
Originator resource name	SPV
Originator process ID	X'0000'
Server name	SERVNAME

## Process disconnection (\*\* function)

This request is issued by the client/server mechanism to notify its resident servers of a disconnected LANDP process. This allows the servers to cancel any pending requests or release any reserved resource belonging to the disconnected LANDP process.

A process is considered to be disconnected when:

- The process is a LANDP client and the process issues the EJ (End of Job) function
- The process is a LANDP for OS/2, Windows NT, or AIX client and the process terminates, either normally or abnormally

## connection functions

- The process is a LANDP for OS/2, Windows NT, or AIX server and it is unloaded (see description of the ES function for clients in *LANDP Programming Reference*, chapter entitled “Supervisor local functions”).
- The process is a LANDP for OS/2, Windows NT, or AIX server and terminates abnormally

Table 13 shows the contents of the CPRB for a process disconnection (\*\*) function request.

Table 13. Process disconnection (**) function, CPRB contents	
CPRB Field	Content/Description
Verb type	X'03'
Function code	**
Request DATA length	0
Request PARMLIST length	0
Reply DATA length	0
Reply PARMLIST length	0
Replied DATA length	0
Replied PARMLIST length	0
Originator workstation ID	C'cc'—the workstation where the process was loaded
Originator resource name	C'ccccccc'—the process that originated the request
Origin process ID	X'xxxx'
Server name	SERVNAME

### Workstation connection (&& function)

Each time LANDP is started or restarted in a workstation or LANDP for AIX system, the configured LAN sessions are automatically established or re-established.

The client/server mechanism issues this request to notify the resident servers of a new workstation or LANDP for AIX system that has been connected to the LAN.

Table 14 on page 63 shows the contents of the CPRB for a workstation connection (&&) function request.

<i>Table 14. Workstation connection (&amp;&amp;) function, CPRB contents</i>	
<b>CPRB Field</b>	<b>Content/Description</b>
Verb type	X'01'
Function code	&&
Request DATA length	0
Request PARMLIST length	0
Reply DATA length	0
Reply PARMLIST length	0
Replied DATA length	0
Replied PARMLIST length	0
Origin workstation ID	C'cc'—the connected or reconnected workstation
Originator resource name	SPV
Originator process ID	X'0000'
Server name	SERVNAME

### Process connection (&& function)

Servers receive this request from their local client/server mechanism when a server from which they can request services connects to LANDP.

A server is considered to be connected when it replies with a successful return code (X'00000000') to the IN function.

Table 15 on page 64 shows the contents of the CPRB for a process connection (&&) function request.

Table 15. Process connection (&&) function, CPRB contents	
CPRB Field	Content/Description
Verb type	X'03'
Function code	&&
Request PARMLIST length	0
Request DATA length	0
Reply PARMLIST length	0
Reply DATA length	0
Replied DATA length	0
Replied PARMLIST length	0
Originator workstation ID	C'cc'—the workstation where the process is loaded
Originator resource name	C'ccccccc'—the name of the server that is loaded
Originator process ID	X'xxxx'
Server name	SERVNAME

---

### Sending asynchronous events

An asynchronous event can be issued by a server that operates without a regular or predictable time relationship with a client. Such an application program is event-driven.

An example where you might use an asynchronous event is when a client interacts with a personal identification number (PIN) pad device and the PIN pad server. The client sends a request to the LANDP PIN pad server to start the PIN pad so that it can accept input from the user. However, the user might not enter the PIN number correctly (for example, entering only three of the required four digits) and the PIN entry activity is not completed. In this situation the PIN pad server and the client remain active and unusable.

To overcome such problems, LANDP supports asynchronous events, enabling application programs and user servers to be event-driven. In the PIN pad example, an asynchronous event could be the completion of a PIN pad entry, or the expiry of a timer set by the application program to limit the time for a PIN pad entry.

LANDP allows clients to wait for asynchronous events with the WM (wait multiple) function; see the “Wait for asynchronous events (WM function)” section in the “Supervisor local functions” chapter of the *LANDP Programming Reference*. WM allows the client to remain in an idle state until any of the specified events happen in the LANDP workgroup. After an event is notified, the client regains control and can process the event or return to a wait state. Asynchronous events ensure that processing time in a multiple-tasking environment is efficiently used.

The server uses the RMTAREQ routine, to request the function Z5 (see the “Asynchronous event notification or cancellation (Z5 function)” section in the “Supervisor local functions” chapter of the *LANDP Programming Reference*) without the reset parameter to notify the client that a specific asynchronous event occurred. If the client is in idle state, the event is passed to the client. Otherwise the event is queued by the client/server mechanism until the client issues a WM function. If the client does not enter the idle state and therefore does not receive the event, the server is then responsible for removing the event. To do this the server calls RMTAREQ and requests the function Z5, with the reset parameter. It is recommended that the server cancels events when they are no longer valid, since the server does not know whether the client has received the notification from the client/server mechanism queue through the WM function. If the client has already received this event the client/server mechanism discards the cancel notification. With LANDP for OS/2 and Windows NT, you can use the ZN function (see the “Extended asynchronous event notification (ZN function)” section in the “Supervisor local functions” chapter of the *LANDP Programming Reference*) instead of Z5.

Using the example of the PIN pad device, the process flow is as follows:

1. The client requests the AR (Arm the PIN Pad for Data Input) function to start the device to accept user input.
2. The server receives the AR request, starts the device, and replies to the requester.
3. The client receives the reply and requests the WM function to wait for the event (and possibly other events). The client then enters the idle state.
4. At some time, the server accepts the user input and signals this event using RMTAREQ, function Z5 or ZN (under LANDP for OS/2 and Windows NT).
5. The client regains control and requests the RD (Read from the PIN Pad) function to obtain data from the server.
6. The server receives the RD request and returns the data to the client.

When the server notifies or cancels an event, it has to specify the client as the destination within the Z5 or ZN request. The server identifies the client by inspecting the CPRB, received with the client's AR request. The server specifies the client using the fields *ehcdest\_pc\_id* and *ehcpid\_dest* in the CPRB before calling RMTAREQ. These fields correspond to the *ehcpc\_id* and *ehcpid\_origin* fields the server received with the AR request.

---

## Server-to-server calls

As explained in Chapter 1, “Clients and servers” on page 1, servers can act as clients, using the same programming interface as used by clients. A server can request services from another server which can call another server, and so on. The servers can be located in the same or different machines in the LANDP workgroup. However, in LANDP for DOS, if both servers are in the same machine, they must be identified in the EHCUSER.CFG file, an ASCII file with separate records containing the name of each server. If a server uses RMTREQ while processing a request, then it must use a different CPRB from that received in a previous GETREQ.

## LANDP for DOS servers

To prevent deadlock situations, the following rules apply for server-to-server calls:

- Circular calls are allowed for LANDP for OS/2, Windows NT, and AIX servers with more than one thread. Here, the additional threads are dedicated to service the incoming request generated this way, while one of the threads is blocked on the RMTREQ. Take care not to loop forever in circular calls.
- Circular calls are *not* allowed for LANDP servers with only one procedure (this includes all LANDP for DOS servers). Server ONE cannot call server TWO, which calls server THREE, which calls server ONE.

---

## Writing LANDP for DOS servers

When writing a LANDP for DOS server, consider the following information related to the special nature of the DOS environment.

### Memory management considerations

In LANDP for DOS, a server is a Terminate and Stay Resident (TSR) program. LANDP provides transparent support at loading time for LANDP for DOS servers to call DOS services without having DOS re-entrancy problems that can occur with TSR programs.

There are, however, several important things to remember related to DOS memory management services.

- To run a TSR program, DOS must know the size of the program at run time. The server must calculate the memory it occupies at run time. The server can discard the memory needed by initialization routines at load time by putting them at the end of the server and declaring a size from the beginning of the server to the beginning of the initialization code and data. This is why SRVINIT needs the parameter *size*.
- By default, DOS allocates all available memory to a program when it is initiated from the command line, so the server has all available DOS memory management services during its load time phase. The load time lasts from when the server starts executing up to the call to SRVINIT. During this time, the server can freely shrink or expand itself, or request blocks of memory, or release them.
- At run time, when the server receives a request, it can still use DOS memory management services, but they are likely to fail with a *not enough memory* error, as all available memory may have been assigned to a running foreground application. Remember that even though the client may not be using such memory, DOS allocates all of it to the client.
- Thus servers should not use DOS memory management services at run time. All memory needed at run time must be allocated at load time.

For servers developed using the IBM C/2 compiler, the above requirements give rise to the following considerations:

- A C/2 program is ordered in CODE segments, DATA segments, the STACK segment, and the HEAP. (See the appropriate compiler books for detailed descriptions.) You can control the amount of program stack and the size of the

HEAP either at link time (with the /STACK and /CPARMAXALLOC options) or after generating the executable file with the EXEMOD utility supplied with the compiler (options /STACK and /MAX).

If you are concerned with the server size at run time, then you can change the sizes of these areas, but you must be very careful with the sizes you choose. The default STACK size for IBM C/2 programs is 2048 bytes. The default HEAP size (assuming there is enough memory when the server is loaded) is:

HEAP size = 65535 - (DEFAULT DATA SEGMENT size) - STACK size

So, even for very small servers, the default size for DATA plus STACK and HEAP is 64 KB. Adding the CODE segments and additional DATA segments, the “default” server needs more than 64 KB.

- Tuning the STACK size is possible by examining the source code of the server. Estimate the “worst case” stack requirements by following every possible path in the server code of every possible request coming to the server. The stack requirements depend on the number and size of parameters passed to called routines during the processing, the number and size of local variables defined in each called routine, and the type of calls (near or far, depending on the memory model used when compiling).

It is recommended not to use the /Gs compiler option (to eliminate stack checking) during the testing phase of the server. A stack overflow could cause unpredictable results.

The available stack size when a request is received through GETREQ is nearly the same as the available stack when the server called SRVINIT. LANDP “remembers” the stack that was in use when calling SRVINIT and resets it every time a request is received.

- Internally the C/2 compiler uses the HEAP for dynamic memory allocation when using routines like:

malloc, calloc, \_nmalloc, or \_fmalloc

Some run-time library routines also use the HEAP, because they call the memory allocation routines when they need memory for their internal buffers or variables (examples of these are the stream I/O routines). It is not possible to predict which library routine calls the memory allocation routines internally.

The C/2 compiler handles memory management internally. Therefore DOS memory management calls need not be used except when there is insufficient memory available in the HEAP to satisfy a request. Here, C/2 attempts to expand the HEAP size by a fixed amount (8 KB) and issues DOS memory management calls to satisfy the current request. Later calls are then again handled internally by C/2 memory management until the remaining HEAP memory has been exhausted. At this point the HEAP memory must be expanded by another 8 KB. You can use the C/2 memory management routines at run time (or routines that call them), when there is enough HEAP space available. DOS memory calls are not allowed at run time.

You can reduce the HEAP size using the options mentioned above (for example, LINK with /CPARMAXALLOC:1 and EXEMOD with /MAX 1 both reduce the HEAP

size to 0). You can write a server that does not use the HEAP at run time, but you must not use the memory allocation routines or routines that call these in turn.

If you still need dynamic memory allocation at load time, you can access the DOS memory management services directly, bypassing the compiler memory management and leaving the program size intact. Then you only use the memory you need and can return the memory not used to DOS.

### Interrupt handling

Sometimes a server needs to capture an interrupt vector (either software or hardware interrupt vector). For example, a server managing a communications adapter needs to capture the hardware interrupt that the adapter generates. When the server receives control because the interrupt has been generated, it bypasses the LANDP mechanism that allows the server to perform operations not normally allowed to TSRs (for example, accessing DOS).

Consider these restrictions when developing the server's interrupt handler:

- The interrupt vector has to be captured when it receives the IN request. The interrupt vector cannot be captured at load time, before calling SRVINIT.
- The server must release the interrupt vector when it receives the ES request. When unloading LANDP for DOS, the client/server mechanism sends ES requests to the servers in reverse order—the first server that received the IN request is the last one to receive the ES. This ensures the proper order for releasing an interrupt vector.

### Expanded memory considerations

The LOADERE program allows servers to be loaded into expanded memory, which makes more memory available for application programs in that workstation. LOADERE is used with LANDP for DOS servers and user-written servers. For more information about LOADERE, see the *LANDP Installation and Customization* book. Consider the following rules for writing a server to be loaded in expanded memory:

- The server size must be smaller than 64 KB. The maximum number of contiguous expanded memory specification (EMS) pages (16 KB each) is four.
- If a server needs a large block of memory for its operation, exceeding the 64 KB limit, it must allocate this memory at load time using the DOS memory management services. The memory is allocated in low memory, but the server can always access it.
- If the server captures software interrupt vectors, LOADERE takes care of it automatically and changes the interrupt vector from normal memory to EMS when the interrupt happens at run time. When the interrupt happens, LOADERE maps the server from EMS to normal memory and gives control to the server's software interrupt handler. The server receives control as if it was running in normal memory. However, between the software interrupt being generated and the server getting control, the server must not rely on interrupts not being enabled, because the map and unmap EMS calls needed for changing the interrupt vector enable interrupts.



- The server cannot capture hardware interrupts (or software interrupts which are issued inside a hardware one), because the EMS calls that map and unmap the server enable interrupts, possibly causing reentrancy problems.

---

## **Writing LANDP for OS/2 servers**

When writing LANDP for OS/2 servers, you can benefit from the advanced capabilities of the operating system which are not available when running on DOS. The consequences are:

- The server is more powerful and has better performance
- The server is not portable at the source code level

If servers do not need to be portable at the source code level, then powerful servers can be developed using OS/2 unique features in combination with LANDP for OS/2 support, as explained below.

## **Using multiple threads**

By using several threads in the server to process several requests at the same time, you can increase the overall performance of the server, and also lower the standard deviation for the mean response time for requests from application programs.

LANDP for OS/2 does not impose any restrictions on the use of threads in servers. LANDP for OS/2 supports several threads calling the LANDP routines at the same time (including GETREQ and RMTRPLY). Thread serialization to access the server's request queue is handled by LANDP for OS/2. However, LANDP for OS/2 does not handle the problem of trying to access the server's own resources (memory, files, and so on) from several threads, at the same time as a request is being processed (a typical problem in any multi-threaded application). It is your responsibility to ensure that proper sharing rules exist. If more than one thread is waiting for GETREQ, it cannot be predicted which thread will get the next request. This depends on the OS/2 dispatching algorithm.

LANDP for OS/2 also allows a request to be answered (through RMTRPLY) from a thread different from the one that originally received the request (through GETREQ). The only restriction is that both threads must belong to the same server. Even the order of replying to requests can be different from the order in which requests were received. LANDP for OS/2 automatically matches the RMTRPLYS with the appropriate GETREQs.

The above considerations give the application programmer flexibility when implementing a multi-threaded server. Server structures can range from very simple to being very complex. For example:

1. Each thread calls GETREQ and RMTRPLY, handling the entire request process. All of the threads contend for requests by calling GETREQ. This is the simplest structure. Request queueing and serialization is entirely performed by LANDP for OS/2. The threads can be created at initialization by the main thread. All the threads are identical.
2. There is only a single thread calling GETREQ. When this thread gets a request, it analyzes and then passes the request to another thread (one of several "service"

threads) which uses OS/2 native primitives. The single thread then goes back and calls GETREQ again.

When a request has been serviced, one of the “service” threads replies through the RMTRPLY.

The complexity of this approach is greater than in the previous case:

- The synchronization between the thread calling GETREQ and the “service” threads must be performed using explicit calls to OS/2 inter-process communication primitives.
- The application programmer has to decide what to do when a request has already been received by the first thread and no more free “service” threads are available. It could return an error to the client, create a new thread for servicing the request, or write the request back into the queue for later processing.

### Using wait multiple (WM) in a server

LANDP for OS/2 servers can request the wait multiple (WM) function for handling asynchronous events. However, the WM function cannot be requested by LANDP for DOS servers. This feature can be very useful when writing a server acting as client, that calls a second server, and where the server acting as client uses asynchronous events for its operation. For example, it is useful when developing a server that calls the SNA server, the PIN Pad server, or the Magnetic Stripe Reader server. Events like host messages or the user entering data in the PIN Pad could be asynchronously detected through the WM function, instead of having some polling mechanism.

Remember, that the WM function is a blocking call. This means that the server acting as client remains idle until one of the events specified in the asynchronous event list occurs (or the timeout for the request expires). A server with only one thread that requests the WM function can prevent other clients from using this server.

A LANDP for OS/2 server acting as client can use the WM function in the same way (with no restrictions) as a normal client. For a detailed description of the call, see the “Wait for asynchronous events (WM function)” section in the “Supervisor local functions” chapter of the *LANDP Programming Reference*. Note the following:

- As well as receiving notifications as posted OS/2 or Windows 3.1 messages, LANDP clients receive asynchronous event notifications using the WM function mechanism. However, LANDP for OS/2 servers acting as clients have two further possibilities: calling GETREQ with a list of pending asynchronous events (see “Additional GETREQ options” on page 52), or using the WM function.
- Take care if the same event list is specified for both GETREQ and WM. If two threads are blocked, one in GETREQ and the other in WM, it cannot be predicted which one will “own” the event.
- Usually, specifying asynchronous events in GETREQ simplifies the server acting as client when it has to handle “general” events. These are events not tied specifically to any request being processed at that time, but are mixed with “normal” requests.

WM events can be used to handle events specific to the current request, and which are of no interest to any other thread the server may have.

For example, process connection (&&), which signals that another server can be used, is a general event suitable for handling in GETREQ. However, a request that needs to access, say, the SNA server, should use the WM function to wait for the host reply. This event is specified as the single event to wait for in the WM function. Process connections are received through GETREQ and host events (for one specific SNA session) are received with the WM.

There can be several outstanding GETREQs together with several outstanding WM functions. It is recommended that the sets of events specified in each call are mutually exclusive, that is, no event is specified in more than one list at the same time. Remember that supplying a null list to WM is like specifying all possible events. Also, specifying events as SNA## is like specifying SNA01, SNA02, and so on. This is because the ## expands into a list, so you might inadvertently get non-mutually exclusive lists.

---

### Writing LANDP for Windows NT servers

When writing LANDP for Windows NT servers, you can benefit from the advanced capabilities of the operating system which are not available when running on DOS. The consequences are:

- The server is more powerful and has better performance
- The server is not portable at the source code level

If servers do not need to be portable at the source code level, then powerful servers can be developed using Windows NT unique features in combination with LANDP for Windows NT support, as explained below.

### Using multiple threads

By using several threads in the server to process several requests at the same time, you can increase the overall performance of the server, and also lower the standard deviation for the mean response time for requests from application programs.

LANDP for Windows NT does not impose any restrictions on the use of threads in servers. LANDP for Windows NT supports several threads calling the LANDP routines at the same time (including GETREQ and RMTRPLY). Thread serialization to access the server's request queue is handled by LANDP for Windows NT. However, LANDP for Windows NT does not handle the problem of trying to access the server's own resources (memory, files, and so on) from several threads, at the same time as a request is being processed (a typical problem in any multi-threaded application). It is your responsibility to ensure that proper sharing rules exist. If more than one thread is waiting for GETREQ, it cannot be predicted which thread will get the next request. This depends on the Windows NT dispatching rules.

LANDP for Windows NT also allows a request to be answered (through RMTRPLY) from a thread different from the one that originally received the request (through GETREQ). The only restriction is that both threads must belong to the same server. Even the order of

replying to requests can be different from the order in which requests were received. LANDP for Windows NT automatically matches the RMTRPLYs with the appropriate GETREQs.

The above considerations give the application programmer flexibility when implementing a multi-threaded server. Server structures can range from very simple to being very complex. For example:

1. Each thread calls GETREQ and RMTRPLY, handling the entire request process. All of the threads contend for requests by calling GETREQ. This is the simplest structure. Request queueing and serialization is entirely performed by LANDP for Windows NT. The threads can be created at initialization by the main thread. All the threads are identical.
2. There is only a single thread calling GETREQ. When this thread gets a request, it analyzes and then passes the request to another thread (one of several “service” threads) which uses Windows NT native primitives. The single thread then goes back and calls GETREQ again.

When a request has been serviced, one of the “service” threads replies through the RMTRPLY.

The complexity of this approach is greater than in the previous case:

- The synchronization between the thread calling GETREQ and the “service” threads must be performed using explicit calls to Windows NT inter-process communication primitives.
- The application programmer has to decide what to do when a request has already been received by the first thread and no more free “service” threads are available. It could return an error to the client, create a new thread for servicing the request, or write the request back into the queue for later processing.

### Using wait multiple (WM) in a server

LANDP for Windows NT servers can request the wait multiple (WM) function for handling asynchronous events. However, the WM function cannot be requested by LANDP for DOS servers. This feature can be very useful when writing a server acting as client, that calls a second server, and where the server acting as client uses asynchronous events for its operation. For example, it is useful when developing a server that calls the SNA server, the PIN Pad server, or the Magnetic Stripe Reader server. Events like host messages or the user entering data in the PIN Pad could be asynchronously detected through the WM function, instead of having some polling mechanism.

Remember that the WM function is a blocking call. This means that the server acting as client remains idle until one of the events specified in the asynchronous event list occurs (or the timeout for the request expires). A server with only one thread that requests the WM function can prevent other clients from using this server.

A LANDP for Windows NT server acting as client can use the WM function in the same way (with no restrictions) as a normal client. For a detailed description of the call, see

the “Wait for asynchronous events (WM function)” section in the “Supervisor local functions” chapter of the *LANDP Programming Reference*. Note the following:

- As well as receiving notifications as posted Windows messages, LANDP clients receive asynchronous event notifications using the WM function mechanism. However, LANDP for Windows NT servers acting as clients have two further possibilities: calling GETREQ with a list of pending asynchronous events (see “Additional GETREQ options” on page 52), or using the WM function.
- Take care if the same event list is specified for both GETREQ and WM. If two threads are blocked, one in GETREQ and the other in WM, it cannot be predicted which one will “own” the event.
- Usually, specifying asynchronous events in GETREQ simplifies the server acting as client when it has to handle “general” events. These are events not tied specifically to any request being processed at that time, but are mixed with “normal” requests.

WM events can be used to handle events specific to the current request, and which are of no interest to any other thread the server may have.

For example, process connection (&&), which signals that another server can be used, is a general event suitable for handling in GETREQ. However, a request that needs to access, say, the SNA server, should use the WM function to wait for the host reply. This event is specified as the single event to wait for in the WM function. Process connections are received through GETREQ and host events (for one specific SNA session) are received with the WM.

There can be several outstanding GETREQs together with several outstanding WM functions. It is recommended that the sets of events specified in each call are mutually exclusive, that is, no event is specified in more than one list at the same time. Remember that supplying a null list to WM is like specifying all possible events. Also, specifying events as SNA## is like specifying SNA01, SNA02, and so on. This is because the ## expands into a list, so you might inadvertently get non-mutually exclusive lists.

---

## Writing LANDP for AIX servers

When writing LANDP for AIX servers, you can benefit from specific features that are provided by LANDP for AIX.

### LANDP for AIX server structure

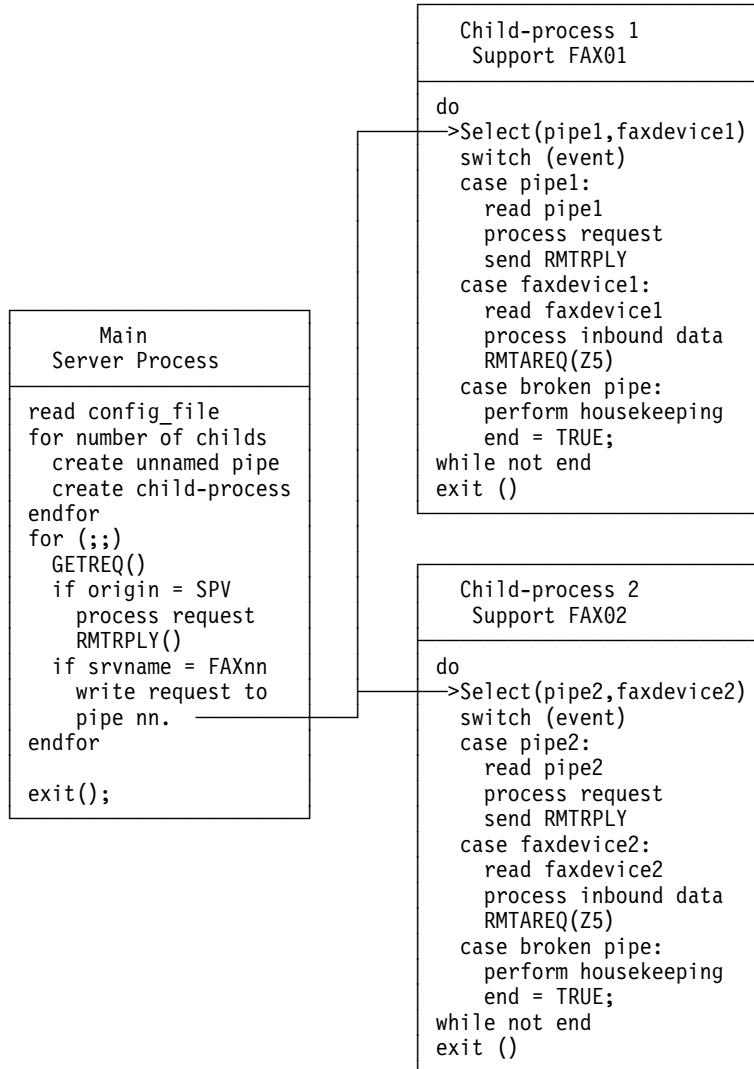
Servers can be structured in two possible ways:

1. A server can be written so that everything is performed within the same process. This type of server can easily be ported to all LANDP platforms.
2. A server can be written by using the AIX multiple-processing facilities. You need this support when you have to perform parallel processing (such as controlling two devices at the same time).

## Using AIX multiple-processing facilities

Provided you follow the rules provided here, your servers can be structured to receive requests in one process and send replies or asynchronous requests from a different process. This type of server structure is useful when writing servers that support more than one service at the same time.

In the following example, a server with name FAX## supports multiple FAX devices, named FAX01, FAX02, FAX03, to FAXnn:



This server structure allows the parent-process and child-process to simultaneously handle all requests, and avoids the need for blocking concurrent memory updates using semaphores. The structure leads to a saving of resources such as shared memory,

semaphores, and so on, and maximizes server performance. This is possible because the inter-process communication by queues or pipes prevents you from blocking the tasks to avoid concurrent updates to the same (shared) resources.

## Calling SRVINIT

The file name of your server may differ from the name of its services. This is fully explained in "Call SRVINIT (server initialization)" on page 49. In LANDP for AIX you must specify the service names you are supporting with the SRVINIT options list:

```
#include "ehcdefc.h"
EHC_SRVINIT_OPTS ehc_srvinopts;

.      .      .
.      .      .

memset(&ehc_srvinopts, '\0', sizeof(EHC_SRVINIT_OPTS));
ehc_srvinopts.struc_size = sizeof(EHC_SRVINIT_OPTS);
ehc_srvinopts.service_name_list = "FAX##  ";
```

## Reply buffer allocation

A LANDP for AIX server can send a reply from a child-process which is different from the server main-process which received the request. The server must provide Reply PARMLIST and DATA buffers to handle the reply.

If the Reply PARMLIST and DATA buffers have already been made available by your program, you enter their addresses in the CPRB. RMTRPLY sends these Reply PARMLIST and DATA areas to the specified reply destination.

The two possibilities for assigning the reply buffers are:

- Allocate special buffers:

```
ehc_error = GETREQ (&cprb, EHC_RESERVED);
cprb.ehcrpamad = malloc ( 100 );
cprb.ehcrdataad = malloc ( 200 );
sprintf(cprb.ehcrpamad, "This is the reply param area", 100);
sprintf(cprb.ehcrdataad, "This is the reply data area ", 200);
cprb.ehcrepldplen = 100;
cprb.ehcreplddlen = 200;
ehc_error = RMTRPLY (&cprb, EHC_RESERVED);
free(cprb.ehcrpamad);
free(cprb.ehcrdataad);
```

- Use your own buffers:

```
ehc_error = GETREQ (&cprb, EHC_RESERVED);
cprb.ehcrpamad = &user_status_structure;
cprb.ehcrdataad = &user_data_buffer;
cprb.ehcrepldplen = sizeof(user_status_structure);
cprb.ehcreplddlen = sizeof(user_data_buffer);
ehc_error = RMTRPLY (&cprb, EHC_RESERVED);
```

**Note:** GETREQ provides Reply PARMLIST and DATA buffers that you can use in a single-process server.

### Server child support

LANDP for AIX allows user servers to be structured into several processes, supporting two different types of parent-child relationship. You should be aware of their differences and understand which type better suits your requirements. For both types of parent-child relationship, use the AIX-fork subroutine to create any child-process after you have called SRVINIT in the parent-process. After the fork, the two types of parent-child relationship and their corresponding rules, are as follows:

#### Parent-process

Can perform any common API function call, independent of the child's behavior.

#### Child-process

Can remain in the parent's LANDP process hierarchy. Now the child process:

- Can call RMTRPLY for the parent
- Cannot call GETREQ or RMTREQ

The child-process can become a LANDP process of its own outside the parent's process hierarchy by calling SRVINIT or the IN function of RMTREQ. Thereafter, the child-process:

- Can issue any common API function call
- Is independent of the parent
- Cannot call RMTRPLY for the parent

### Server events

You may have the requirement that the server child-process needs to notify the parent-process. Because during a GETREQ the server parent-process is mostly in a waiting state, LANDP for AIX supports the sending of *server-events* to the server, by calling RMTAREQ.

The only restriction is that the server parent-process cannot call RMTREQ because, while doing so, all incoming asynchronous messages that are not defined by LANDP as originating from the supervisor (SPV) are discarded.



## Chapter 4. LANDP–DCE application programming interface

This chapter describes how to write Distributed Computing Environment (DCE) client and server applications using the LANDP–DCE Application Programming Interface of **LANDP for AIX**. This interface allows client applications in the DCE environment to make remote calls to servers in the LANDP environment, and client applications in a LANDP environment to request services from servers in a DCE environment.

The interoperability between the LANDP and DCE environments is provided through a LANDP for AIX workstation. LANDP client applications and servers can run in any LANDP workstation.

Using the LANDP–DCE interface a DCE client application can make remote calls to LANDP servers and to DCE servers that export the LANDP–DCE interface. LANDP servers receive requests from DCE clients using the LANDP Common API.

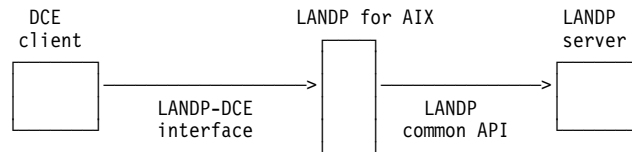


Figure 2. DCE Client Accessing LANDP Server

A DCE server conforming to the LANDP–DCE interface can provide services to both LANDP clients and DCE clients. LANDP clients access DCE servers using the LANDP common API.

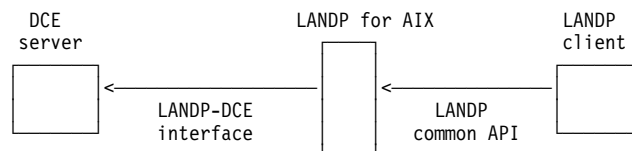


Figure 3. LANDP Client Accessing DCE Server with LANDP Interface

LANDP clients can access DCE servers that do not conform to the LANDP–DCE interface. To do this, a client uses an intermediate user-written server that has the LANDP–DCE interface and that acts as a client of the DCE server to be accessed. A single intermediate server can be used to access several DCE servers.

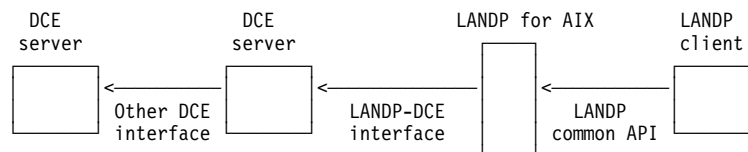


Figure 4. LANDP Client Accessing DCE Server with Non-LANDP–DCE Interface

---

### The LANDPDCE.IDL and LANDPDCE.ACF files

The `landpdce.idl` file contains the definition of the LANDP–DCE interface in Interface Definition Language (IDL) syntax. The `landpdce.acf` file contains additional interface attributes, which modify the behavior of the interface, in Attribute Configuration File (ACF) syntax.

LANDP for AIX provides these files in the `/usr/lpp/landp/c` directory.

The definitions in the `landpdce.idl` file include:

- The data structure, `EHC_CPRB`, used for information exchange between the DCE program and the LANDP program
- The function call, `LANDP_service`, used by DCE clients to make remote calls to servers and by DCE servers to receive calls from clients
- Function calls for context handling: `LANDP_get_context` and `LANDP_context_rundown`

The `landpdce.idl` and `landpdce.acf` files are used to obtain the LANDP–DCE client and server stubs, which are the interfaces between the DCE client and server applications and the DCE run-time routines.

---

### Writing a LANDP–DCE client

This section describes how to write a DCE client that can make remote calls to LANDP servers.

LANDP for AIX provides in:

`/usr/lpp/landp/c/samples/dce_client.c`

a sample DCE client that can be used as a skeleton or template to write DCE clients.

### Obtaining a binding handle for LANDP services

To enable accessing LANDP services, a DCE client must bind first to the LANDP services. A single binding gives access to all LANDP servers.

A DCE client binds to LANDP services either using Naming Services or, when Naming Services are not available, using string bindings.

#### Binding using naming services

LANDP services export bindings to two entries in the *name-space*

`././subsys/LANDP/default`  
`././subsys/LANDP/LANsuffix`

where `LANsuffix` is the LANDP workgroup name entered at LANDP customization.

If a DCE client has access to a single LANDP workgroup, it can use the `default` entry name in the name-space. However, if a DCE client has access to more than one LANDP workgroup, it should use the entry name with `LANsuffix` to get a binding

handle. It can even get binding handles for several LANsuffix entries and make concurrent calls to LANDP servers in different LANDP workgroups.

The `././subsys/LANDP` directory must exist in the name-space.

DCE clients using Naming Services may import bindings with the `rpc_ns_binding_import` calls or with the `rpc_ns_binding_lookup` calls.

DCE clients importing bindings with these functions must supply the following values for the required parameters:

Parameter	Value
Entry name syntax	<code>rpc_c_ns_syntax_default</code>
Entry name	<code>././subsys/LANDP/default</code> , or <code>././subsys/LANDP/LANsuffix</code>
Interface specification	<code>LANDPDCE_v1_0_c_ifspec</code>
Object uuid	NULL, or an explicit uuid value for a specific instance of the LANDP–DCE interface

**Note:** Each instance of the LANDP–DCE interface uses a dynamically generated object uuid, which is used when exporting the binding to the name-space. When a DCE client imports bindings using NULL as the object uuid, Naming Services provide bindings for any entry matching the interface specification, regardless of the object uuid. Using a specific value for the object uuid provides the bindings that match also that object uuid value. The object uuid values exported to an entry in the name-space can be obtained using Cell Directory Services Control Program (*cdsccp*).

A sample DCE client may use the following DCE remote procedure calls to get access to all LANDP servers in the LANDP workgroup `bcn`:

```
rpc_ns_binding_import_begin(rpc_c_ns_syntax_default,
                           "/./subsys/LANDP/bcn",
                           LANDPDCE_v1_0_c_ifspec,
                           NULL,
                           bcn_impctx,
                           status)

rpc_ns_binding_import_next(bcn_impctx,
                           status)

rpc_ns_binding_import_done(bcn_impctx,
                           status)
```

## Binding using string bindings

If Naming Services are not available, DCE clients can bind with LANDP services using string bindings. A string binding is a character representation of the binding handle.

On the LANDP for AIX workstation that provides the LANDP–DCE interoperability, the system administrator can find the string bindings for LANDP services by looking with

SMIT (the System Management Interface Tool) at the end-point mappings. The entry or entries with the annotation LANDP DCE V1.0 LANsuffix provide the LANDP–DCE interoperability for the LANDP workgroup LANsuffix.

### Obtaining a LANDP context

LANDP servers and the LANDP client/server mechanism manage context information. A DCE client making requests to LANDP servers or to the LANDP client/server mechanism needs to get LANDP context information, and use it in all LANDP service calls. DCE clients get LANDP context information using DCE context handles.

DCE clients obtain a LANDP context using a function call defined in the LANDP–DCE interface. The function call used to obtain a LANDP context handle is:

```
LANDP_get_context(bh, ch)
```

where:

- bh is the binding handle obtained for LANDP services. This is an input parameter of `handle_t` type.
- ch is the context handle provided by LANDP services to make remote calls. This is an output parameter of `LANDP_context*` type.

The function provides a return code of `error_status_t` type. Two types of errors can be returned:

- DCE errors (communication status or fault status)
- LANDP context errors

For information on these errors, see the *LANDP Problem Determination* book.

Each function call for a LANDP service uses the context handle obtained in this request.

A single LANDP context handle is used for the requests to all LANDP servers.

### Requesting LANDP services

DCE clients make remote calls to LANDP servers using the interface defined in `landpdce.idl`. The function call used by DCE clients to access any of the LANDP servers is:

```
LANDP_service(ch, cprb_ptr)
```

where:

- ch is the LANDP context handle obtained with the `LANDP_get_context` function call. This is an input parameter of `LANDP_context*` type.
- cprb\_ptr is a pointer to the control block defined in `landpdce.idl`. This is an input and output parameter of `EHC_CPRBP` type. The description and contents of the fields is equivalent to the CPRB described in “Connectivity programming request block (CPRB)” on page 5.

The errors returned by the function call can be found in:

- The function return code (error\_status\_t type)
  - DCE errors (communication status or fault status)
  - LANDP context errors
- The corresponding fields of the cprb\_ptr parameter
  - LANDP router return codes
  - LANDP server return codes

For information on these errors, see the *LANDP Problem Determination* book.

A DCE client that makes a request is suspended until the reply arrives or the timeout expires. Both LANDP and DCE timeout mechanisms can control the elapsed time.

- DCE timeout is a communication timeout (time to send the request to a server)
- LANDP timeout is a service timeout (time to wait for the reply from a server)

The supervisor local functions that a DCE client can use are:

<b>AA</b>	Ask for asynchronous events.
<b>EJ</b>	Disconnect an application program.
<b>ES</b>	Unload IBM LANDP.
<b>T0 through T8</b>	Activate and deactivate timers.
<b>WM</b>	Wait for asynchronous events.

For more information on these functions, see the “Supervisor local functions” chapter of the *LANDP Programming Reference*.

The asynchronous events provided in the reply to the WM and AA functions are server events, timer events, and process connection and disconnection events. For more information on these events, see “LANDP event notification support” on page 36.

## Releasing a LANDP context

The DCE client releases a LANDP context by issuing an EJ supervisor local function. If the DCE client finishes without issuing the EJ function, LANDP services release the context information.

## DCE client structure

The following example shows the structure of a DCE client.

```
...
rpc_ns_binding_import_begin
rpc_ns_binding_import_next
keep binding handle (bh)
rpc_ns_binding_import_done
...
```

```
rc=LANDP_get_context(bh, ctxh)
check rc
...
cprb.ehcfunct=IR_FUNCTION
cprb.ehcserv="MYSHF"
parmlist=concat(myfile, myformat)
dataarea=myrecord
rc=LANDP_service(ctxh, cprb)
check rc
If cprb.ehcservrc=OK then
...
cprb.ehcfunct=EJ_FUNCTION
cprb.server="SPV"
LANDP_service(ctxh, cprb)
...
```

---

### Writing a LANDP–DCE server

This section describes how to write a DCE server that can receive requests from LANDP clients and from DCE clients.

DCE servers serving requests from LANDP clients have:

- A server code, with the remote procedure calls for setting up the server
- A manager code, with the routines for the remote calls: `LANDP_get_context`, `LANDP_service`, and `LANDP_context_rundown`
- A DCE server stub, obtained from the `landpdce.idl` file

You can develop DCE servers to be accessible by both LANDP and DCE clients using the LANDP–DCE interface.

LANDP for AIX provides in `/usr/lpp/landp/c/samples/dce_server.c` a sample DCE server that shows the remote procedure calls required to access this server from LANDP clients. You can modify this set of function calls if there are specific requirements for a type of binding, usage of well-known endpoints, or protocol sequences.

### Binding

DCE servers can make their bindings available by exporting them to Naming Services. DCE servers can also be accessible using string bindings. LANDP services get binding handles for all DCE servers using the method (Naming Services or string binding) defined for each server in LANDP customization.

#### DCE server exporting bindings to naming services

DCE servers that conform to the LANDP–DCE interface must use `LANDPDCE_v1_0_s_ifspec` as the interface specification when exporting their bindings to Naming Services.

The rest of the parameters used when exporting the bindings must match the definitions for the server made during LANDP customization.

During LANDP customization, you specify every DCE server using the DEFSESV vector. For DCE servers that export bindings to the name-space, specify the following in this vector:

**Keyword Contents**

DCESNAME DCE server CDS entry name.

DCESSYNT DCE server CDS entry name syntax.

DCESUUID DCE server object uuid.

The following table shows a sample of the information as provided in LANDP customization.

LANDP User Server Name	DCE Server Entry Name	DCE Server Entry Name Syntax	Object UUID
DPTPRT	./prt/dpt_1734	0 (rpc_c_ns_syntax_default)	
QLTPRT	./prt/quality	3 (rpc_c_ns_syntax_dce)	12345678 -1234-5678-90ab -1234567890ab
MAIL	./mail_server		

### DCE server using string bindings

LANDP services bind with DCE servers that do not export their bindings to the name-space using string bindings. The string binding for these servers must be provided during LANDP customization.

During LANDP customization you specify every DCE server using the DEFSESV vector. You specify the string bindings for servers that do not export their bindings to the name-space by using the DCESTRBI keyword.

The following table shows a sample of the information as provided in LANDP customization.

LANDP User Server Name	String Binding
DPTPRT	ncadg_ip_udp:16.20.16.27[2001]
QUALPRT	12345678-1234-5678-90ab-1234567890ab@ncadg_ip_udp:HighRisc

### Providing a LANDP context in the server manager routines

The LANDP client/server mechanism manages context information. A DCE server using the LANDP–DCE interface must provide a LANDP context to LANDP clients.

DCE servers provide LANDP context using the interface defined in `landpdce.idl`. The function to provide a LANDP context handle is:

```
LANDP_get_context(bh, ch)
```

where:

`bh` is the binding handle. This is an input parameter of `handle_t` type.

`ch` is the context handle provided to the LANDP client. This is an output parameter of `LANDP_context*` type.

LANDP clients do not manage directly LANDP contexts provided by DCE servers. IBM LANDP manages them internally and associates them with the LANDP context information.

The function must provide a return code of `error_status_t` type that can be used for reporting context handle errors.

DCE servers that do not manage context information must return a non-NULL context handle in the `LANDP_get_context` function. DCE servers that manage context information should be aware that the LANDP context information, workstation ID, and Process ID can be obtained from the `cprb_ptr` parameter supplied in each `LANDP_service` request.

### Providing services to LANDP clients in the server manager routines

DCE servers obtain requests from LANDP clients using the interface defined in `landpdce.idl`. The manager routine used by DCE servers to obtain LANDP requests is:

```
LANDP_service(ch, cprb_ptr)
```

where:

`ch` is the LANDP context handle that the client application uses. This is an input parameter of `LANDP_context*` type.

`cprb_ptr` is a pointer to the control block defined in the `landpdce.idl`. This is an input and output parameter of `EHC_CPRBP` type. The description and contents of the fields is equivalent to the Connectivity Programming Request Block as described in Chapter 3, "Writing your own server programs" on page 47.

The function provides a return code of `error_status_t` type that can be used for reporting context handle errors.

The code to service the request must be part of the `LANDP_service` routine. When DCE servers have finished processing the request from the LANDP client, they reply by leaving the `LANDP_service` routine. The `cprb` must contain the Reply DATA.

### Releasing LANDP context in the server manager routines

When a LANDP client having a context handle for a DCE server finishes processing or releases its LANDP context, the `LANDP_context_rundown` routine of the server is invoked with that context handle.



A DCE server must release any context information for the LANDP client on its LANDP\_context\_rundown routine.

## DCE server structure

The following example shows the structure of a DCE server accessible from LANDP clients:

```

LANDP_get_context(bh, ctxh)
{
    Look for a free entry in Context Handle table
    If (no free entry) return(error)
    Keep context information
    Assign context handle
}

LANDP_service(hdl, cprb_ptr)
{
    Look for context information
    Process request
}

LANDP_context_rundown(ctx)
{
    Release context information
}

main()
{
    Create Context Handle table
    rpc_server_register_if
    rpc_server_use_protseqs
    rpc_server_inq_bindings
    rpc_ep_register_no_replace
    rpc_ns_binding_export
    rpc_server_listen
    rpc_server_binding_unexport
    rpc_ep_unregister
}

```



## Chapter 5. Object-oriented application programming

LANDP provides *wrappers* or language *bindings* that allow object-oriented programs written in C++, Java, or Smalltalk, to request LANDP services. Both clients and servers can be written using the object-oriented languages supported by LANDP.

The object-oriented paradigm closely matches the LANDP client/server mechanism concept, and can therefore be easily used with LANDP.

Classes and examples are shipped on the LANDP Version 5 product CD-ROM in source code format. The C++ classes can be used in LANDP for DOS, OS/2, and Windows NT. The Smalltalk class can be used in LANDP for OS/2 and Windows NT.

---

### Writing application programs using C++

The LANDP class library contains class hierarchies that can be used for writing both clients and servers.

The classes are in the following directories:

EHCD5000/SAMPLES/CPP for DOS

EHCO5000/SAMPLES/CPP for OS/2

EHCN5000/SAMPLES/CPP for Windows NT

### LandpRequest class

The LandpRequest class is the general *base class* for a client to send a request to a server. It encapsulates the implementation of the Connectivity Programming Request Block (CPRB) and RMTREQ function, into one unique interface object. The CPRB and RMTREQ are supplied with the LANDP common API.

Using the methods contained in this class of objects called LandpRequest, the client can query and define the common API fields and send a request to the server. The RMTREQ is therefore translated into an *object-oriented call*. LandpRequest also contains the methods to receive the reply from the server, and to access the information returned with this reply.

You can use LandpRequest as a base class to create *subclasses* for individual servers. The subclasses use methods contained in the LandpRequest class.

Following are the LandpRequest methods:

#### **LandpRequest Constructor**

Initializes the required fields of the CPRB with default values.

#### **~LandpRequest Destructor**

Disposes of the CPRB.

### **SetupRequest**

The SetupRequest method provides a way of resetting request information to the default values. For the base class, the values are reset to zero. It is intended to be overridden by descendant classes for a specific re-initialization of the CPRB.

The following methods provide a way of defining the information to be used in a request:

### **SetServerName**

Sets the server name with the given string, truncating or padding the string if required.

### **SetFunctionName**

Sets the function code with the value corresponding to the given string.

### **SetFunctionCode**

Sets the function code with the given value.

### **SetRequestData**

Sets the Request DATA address and the Request DATA length fields.

### **SetRequestParmlist**

Sets the Request PARMLIST address and the Request PARMLIST length fields.

### **SetReplyData**

Sets the Reply DATA address and the Reply DATA length.

### **SetReplyParmlist**

Sets the Reply PARMLIST address and the Reply PARMLIST length.

### **SetTimeOutValue**

Sets maximum wait time (in seconds) for the reply to this request.

The following method provides a way of sending a request and waiting for a reply:

### **SendRequest**

Sends the prepared request to LANDP.

The following methods provide a way of accessing the information that was returned with the reply:

### **GetServerReturnCode**

Obtains the *server return code*.

### **GetRouterReturnCode**

Obtains the *router return code*.

### **GetReturnCode**

Checks first the *router return code* and, if this is correct, checks the *server return code*.

### **GetReturnCodeAsString**

Returns a string containing the readable return code.

**GetRepliedParmListLength**

Returns the Replied PARMLIST length.

**GetRepliedDataLength**

Returns the Replied DATA length.

**RequestFromLandp class**

The RequestFromLandp class is the class that a server uses to receive a client request from the LANDP client/server mechanism, and to reply to the request.

This class contains the methods that a server requires to access request fields and to update reply fields. The client on the other hand, requires the methods to define request fields and to access reply fields. A general example of the use of this class, together with the LandpServer class, is given in “Example of RequestFromLandp and LandpServer class use” on page 91.

The RequestFromLandp class provides the following methods:

**GetRequest**

Provides a way for the server to wait for, and obtain a request, from the client/server mechanism.

**SendReply**

Provides a way for the server to send a reply to the client/server mechanism.

**SetupReply**

Provides a way for the server to reset the reply information to default values. The base class resets the Replied DATA length, Replied PARMLIST length, and *server return code* to zero. These values can then be redefined by descendant classes, when a specific initialization of the reply is required.

The following methods provide a way for the server to access the information contained in a request that has been received from the client/server mechanism:

**GetFunctionCode****GetRequestParmlistLength****GetRequestDataLength****GetReplyParmlistLength****GetReplyDataLength****GetRequestParmlistAddress****GetRequestDataAddress****GetReplyParmlistAddress****GetReplyDataAddress****GetRequesterId**

## object-oriented programming

The following methods provide a way for the server to define the reply fields contained in a reply that is to be sent to the client/server mechanism:

**SetServerReturnCode**

**SetRepliedDataLength**

**SetRepliedParmlistLength**

### LandpServer class

The LandpServer class provides the structure for all LANDP servers. A server written in C++ must use an instance of a class derived from LandpServer class. This derived class should add the specific server service routines to the LandpServer class. A general example of the use of this class, together with the RequestFromLandp class, is given in “Example of RequestFromLandp and LandpServer class use” on page 91.

The LandpServer class reflects the general technique of LANDP server programming: SRVINIT, followed by a GETREQ loop, and then a RMTRPLY (see Chapter 3, “Writing your own server programs” on page 47). The LandpServer class is the basis for creating a user-written class to write a server.

The Run method provides the server's main loop. It must be called in the primary function of the LANDP server. The Run method should not be overridden in any of the descendant classes.

The InitRequestFromLandp method is used by the LandpServer base class to instantiate an object of the class RequestFromLandp. Usually, descendant classes override it, to further instantiate from other classes.

The GetReceivedRequest method provides a way for the server to return the pointer to the request that has been received.

The ProcessRequest method is intended to be overridden by all its descendants. The statements that identify the function codes that are supported by the server should be added in the overridden method. The overridden method should call the inherited ProcessRequest, before returning, to process the default function codes.

The only requests that are processed on the ProcessRequest method of the LandpServer class, are those that come with the IN, ES, \*\*, and && functions:

**ProcessInitialize**

**ProcessEndService**

**ProcessConnection**

**ProcessDisconnection**

A server that is going to process these functions needs to override these methods and call the inherited method before returning.

**Example of RequestFromLandp and LandpServer class use**

The following example (LICSTST.CPP) illustrates the use of the RequestFromLandp and LandpServer classes for creating a LANDP date and time server:

```
#include "LICSrv.HPP"

                                // Declare a subclass of LandpServer
                                // that contains the specific
                                // behavior of the server

class SampleServer : public LandpServer
{
    virtual long ProcessRequest(int fc);
                                // Mandatory override of
                                // the request processor

    long ProcessGetDate(void);
    long ProcessGetTime(void);
                                // Methods to carry out the
                                // server's function set
};

long SampleServer::ProcessRequest(int fc)
{
    switch (fc) {
                                // Invoke appropriate service
                                // function

        case fcGetDate: return ProcessGetDate();
        case fcGetTime: return ProcessGetTime();

                                // Or pass it to the super-class
                                // for default processing
        default : return LandpServer::ProcessRequest(fc);
    };
};

long SampleServer::ProcessGetDate()
{
    ... return(0);
};

long SampleServer::ProcessGetTime()
{
    ... return(0);
};

                                // The program body simply
                                // instantiates a SampleServer
                                // object and invokes its inherited
                                // Run() method

main()
{
    SampleServer thisServer;
```

```
        thisServer.Run();  
    };
```

---

### Writing application programs using Smalltalk

The LANDP Smalltalk class libraries contain objects that can be used for writing LANDP client applications using IBM VisualAge Smalltalk.

The classes are supplied in the following files:

ldpvast4.cls for VisualAge Smalltalk Professional 4.0

ldpvast5.cls for VisualAge Smalltalk Professional 4.02 and VisualAge Smalltalk Enterprise 4.5 and 5.0

There are simple examples of the classes in use in landpage.txt.

### LandpRequest class

The LandpRequest class is the base class.

The new method returns a new instance of a LandpRequest. The initialize method loads EHCOS2.DLL (under OS/2) or EHCWINNT.DLL (under Windows NT) and obtains area blocks for the common API data and parameters.

The remoteRequest method does the call to LANDP.

The LANDP interface from Smalltalk is carried out as a modal interface. All the state variables are set by the setting methods, and are performed by the remoteRequest method, which interfaces EHCOS2.DLL (under OS/2) or EHCWINNT.DLL (under Windows NT) through the RMTREQ entry point.

You can define multiple instances of LandpRequest and each of these instances can be used as a Smalltalk object.

*The methods that can be used are as follows:*

new	Obtain a new instance
initialize	Initialize the environment
pcid	Obtain the PC identifier
remoteRequest	Call LANDP for service
functionName: aString	Set the function code in characters
functionCode: aString	Set the function code in hexadecimal
serverName: aString	Set the server name
requestData: aString	Set the Request DATA
requestParmList: aString	Set the Request PARMLIST
requestParmList: aString from: anInteger	Set the Request PARMLIST at offset
repliedData	Obtain the Reply DATA
repliedParmList	Obtain the Reply PARMLIST
repliedDataLength	Obtain the Replied DATA length



<code>repliedParmListLength</code>	Obtain the Replied PARMLIST length
<code>replyDataLength: anInteger</code>	Set the Reply DATA length
<code>replyParmListLength: anInteger</code>	Set the Reply PARMLIST length
<code>timeOutValue: anInteger</code>	Set the timeout value for the request
<code>serverReturnCode</code>	Check the server return code
<code>routerReturnCode</code>	Check the router return code
<code>returnCode</code>	Check any error return code
<code>finalize</code>	Terminate

The following example shows how to use the LANDP banking printer server (PR47X2##), using session number 01. The following program statements must be contained in a method, and then executed:

```
BankingPrinter:= LandpRequest new.
BankingPrinter initialize.
BankingPrinter serverName: 'SPV'.
BankingPrinter functionName: 'IN'.
BankingPrinter remoteRequest.
BankingPrinter serverName: 'PR47X201';
                    functionName: 'WR'.
```

To print a line, you must only then execute one statement:

```
BankingPrinter requestData: 'Text to be printed'; remoteRequest.
```

The following statement allows you to obtain data sent by the printer.

`BankingPrinterServer` is another instance of `LandpRequest` class.

```
BankingPrinterServer repliedData.
```

The following method provides a way of testing the result, by examining the string that is returned from:

```
BankingPrinter serverReturnCode.
```

Also, splitting the Request PARMLIST or Reply PARMLIST area into fields that are relevant for each server improves the transparency between LANDP services and Smalltalk.

The method "`requestParmList: aString from: anInteger`" enters a value in a Request PARMLIST field, using a given Offset. Using this method you can split the complete Request PARMLIST area by fields, and then have separate methods for setting and obtaining the value of a specific server parameter. The layout remains the same as the rest of the LANDP parameters. You can similarly split the complete Reply PARMLIST, Request DATA, and Reply DATA areas.



---

## Chapter 6. LANDP support for Java

The LANDP support for Java enables LANDP clients and servers to be written in the Java programming language on both OS/2 and Windows NT. Applications can be written once and moved between platforms without the need for recompiling. The Landp support for Java allows you to write Java applications, applets, and servlets that can access LANDP servers and services.

The LANDP Java support consists of the following files:

LDPJAVA.JAR	A Java Archive File containing the LANDP Java classes
LDPJDOCS.ZIP	Javadoc documentation
LDPJMAN.EXE	The LANDP Java Manager
LDPJDISP.EXE	The LANDP Java Dispatcher
LDPJAVA.DLL	DLL that implements the native methods of the LANDP Java classes

For information on installing LANDP Java support, refer to the sections headed "Installation requirements for Java support" in the chapters entitled "Preparing OS/2 workstations" and "Preparing Windows NT workstations" in the *LANDP Installation and Customization*

---

### Support for Version 4 classes

LANDP Version 5 Java support includes support for the previous LANDP classes provided as an add-on to Version 4 from the LANDP web site. Landp.class is included in LDPJAVA.JAR, but has been marked as deprecated. The LDPJNT.DLL and LDPJOS2.DLL files have been replaced by LDPJAVA.DLL. To run your existing LANDP Java applications under the Version 5 Java support, put LDPJAVA.JAR in your class path to replace the previous classes and replace either LDPJNT.DLL or LDPJOS2.DLL with LDPJAVA.DLL.

---

### VisualAge for Java support

The LANDP classes are fully compatible with IBM VisualAge for Java version 3 and can be imported from LDPJAVA.JAR into the VisualAge repository.

To import the LANDP classes, perform the following steps:

1. Create a new project entitled for example "IBM LANDP support"
  - From the main menu choose Selected|Add|Project
  - Select "Create a new project named".
  - Enter the project name.
2. Import the contents of LDPJAVA.JAR into the project
  - From the File menu, select Import...
  - When prompted to select an import source select Jar file and click Next.

- Enter the path and name of the Jar file, for example: `c:\landp\ldpjava.jar`
- Select both `.class` and resource as the types of files to import.
- Where prompted to "Enter name of a project to import into" enter the name of the project created in step 1.
- Click Finish

When writing LANDP applications or applets within VisualAge for Java and testing them within the IDE, the project into which you imported the LANDP classes must be included in the application classpath:

- Select the class of the application containing the `main()` or `start()` method.
- From the Selected menu choose "Run" and then "Check Class Path..."
- Make sure "Project Path" is clicked and then press the "Compute Now" button.
- The project containing the LANDP classes should be added to the Project Path. If not press the "Edit" button to add it manually.
- When complete press "OK".

---

## Java client development

The LANDP Java classes are contained in LDPJAVA.JAR, and enable LANDP clients to be written in Java. The classes are:

### **Cprb**

A class representing the CPRB. It contains variables for each field in the CPRB and methods to get and set those variables.

### **RmtReq**

A class used to send a CPRB to the supervisor. This class can be used in one of two modes:

- It can send requests to LANDP under the process ID of the Java Virtual Machine (JVM) in which the class is loaded.
- It can request its own process ID in which to send requests via the LANDP Java Manager (see below)

The mode can be set by supplying either of the static variables `RmtReq.MODE_JVM_PID` or `RmtReq.MODE_OWN_PID` to the class constructor. For more information on the two modes and their uses, see "Support for multiple client applications within a JVM" on page 97.

The `RmtReq` class has four main methods:

### **register**

The `register` method must be called before any requests are sent when `MODE_OWN_PID` has been specified. It registers the application with the LANDP Java Manager.

**setCprb**

This method sets the CPRB to be sent.

**send**

This method sends the CPRB to LANDP and returns the CPRB received back.

**unregister**

Unregisters the application from the LANDP Java Manager.

**LandpUtils**

Contains utility methods for performing various operations including return code conversion.

**LandpException**

An exception class with sub-classes as follows:

**InvalidCprbException**

Thrown by the send method of RmtReq when a CPRB is incomplete or invalid.

**BadRouterRCException**

Thrown by the send method of RmtReq when a request is sent to the supervisor but returned with a non-zero router return code.

**BadServerRCException**

Thrown by the send method of RmtReq when a request is sent to the supervisor but returned with a non-zero server return code.

**LandpCommunicationException**

Thrown for various reasons when communication with the LANDP Java Manager has failed.

For more detailed information on the LANDP classes, refer to the Java documentation contained in the file LDPJDOCS.ZIP

---

## Support for multiple client applications within a JVM

LANDP servers identify traditional client applications by their process identifier. However, in a JVM, all classes loaded run under the same process ID. When two LANDP applications are loaded into the same JVM, there is the potential for LANDP to fail to recognise them as separate entities.

The LANDP Java Manager is designed to avoid this problem. The Manager intercepts requests from registered LANDP Java applications and dispatches them to LANDP from different process IDs. By using the LANDP Java Manager, it is possible to run LANDP applications concurrently from within one JVM. This is particularly useful when you are writing an application that acts on behalf of a number of users, for instance a Java Servlet.

The LANDP Java Manager can be started from the command line by executing the command LDPJMAN.EXE

Each instance of the `RmtReq` class within a JVM is seen as a separate LANDP application. When constructing an instance of `RmtReq`, specify one of the following static variables as the mode in which the class will be used:

#### **`RmtReq.MODE_JVM_PID`**

In this mode the application will send requests from the PID of the JVM in which it is running. If you are certain that your application will be the only LANDP application within that JVM, then using this mode will result in better performance. When an application is running in `MODE_JVM_PID`, it does not require the services of the LANDP Java Manager.

#### **`RmtReq.MODE_OWN_PID`**

If this mode is specified, requests sent from this class will be sent from a unique process ID obtained from the LANDP Java Manager.

If no mode is specified in the `RmtReq` constructor, `MODE_JVM_PID` is assumed by default. When in `MODE_OWN_PID`, before requesting LANDP services, register your instance of `RmtReq` with the LANDP Java Manager. The `register()` method of `RmtReq` allows you to do this, for example:

```
RmtReq req = new RmtReq(RmtReq.MODE_OWN_PID);

boolean registered = false;
try{
    registered = req.register();
    ...
} catch(LandpCommunicationException lce){
    handleException(lce);
}
```

When your `RmtReq` class successfully registers with the Manager, it can send requests to LANDP servers. The `register` method does not perform the IN request to the LANDP supervisor, therefore, after registering, first send the IN request. Requests are sent by creating a `Cprb` object and populating the relevant fields. The `Cprb` can then be sent to LANDP by calling the `send` method of `RmtReq` as follows:

```

Cprb cprb = new Cprb("SPV", "IN");
cprb.setQParamLength(26);
cprb.setQDataLength(0);
cprb.setRParamLength(26);
cprb.setRDataLength(0);
cprb.setQParam(new byte[1024]);
cprb.setQData(new byte[1024]);
cprb.setRParam(new byte[1024]);
cprb.setRData(new byte[1024]);

req.setCprb(cprb);

try{
    cprb = req.send();
    ...
} catch(InvalidCprbException ice){
    handleException(ice);
} catch(BadRouterRCEException bre){
    handleException(bre);
} catch(BadServerRCEException bse){
    handleException(bse);
} catch(LandpCommunicationException lce){
    handleException(lce);
}
...

```

All further requests to LANDP servers sent via the same RmtReq object will be seen by LANDP to be coming from the same process ID. When the final request has been sent, the RmtReq object must unregister from the LANDP Java Manager. This is performed by calling the unregister() method in much the same way as the register method():

```

boolean unregistered = false;
try{
    unregistered = req.unregister();
    ...
} catch(LandpCommunicationException lce){
    handleException(lce);
}
...

```

Again, the unregister() method does not automatically perform an EJ request to the LANDP supervisor. Application programs must do this before calling unregister().

Each RmtReq instance using MODE\_OWN\_PID within a JVM is essentially an individual LANDP client application. As such, it is allocated resources by both the LANDP Java Manager and LANDP itself. To ensure that these resources are cleaned up correctly, you should not re-use RmtReq objects, nor use them as static class variables. Once the unregister() method of a RmtReq instance has been called, the instance should be left to be garbage-collected by the JVM and not re-used.

---

## Exception handling

As shown above, the `register()`, `unregister()` and `send()` methods of `RmtReq` can generate various exceptions. In the case of the `BadRouterRCEException`, `BadServerRCEException` and `LandpCommunicationException`, the exception classes include a `getReturnCode()` method that returns a long containing the relevant return code contained in the CPRB, or resulting from the LANDP Java Manager. By using the Java try catch constructs, you can control how the application catches and deals with exceptions.

In the case of the `BadRouterRCEException` and `BadServerRCEException` classes, the return code represents the actual return code from the router or server accessed by the request. When one of these exceptions is thrown, the reply CPRB from the request that caused the exception can be obtained by calling the `getCprb()` function of `RmtReq`.

The return code of a `LandpCommunicationException` provides little information of use to the application programmer but is useful to service personnel. When a `LandpCommunicationException` occurs it generally implies that the LANDP Java Manager has failed in some way and should be restarted. However, the return code value 205 suggests that the LANDP Java Manager is not running.

---

## Writing servlets to access LANDP

The LANDP classes can be utilised to write Java Servlets that can be used with an appropriate application server such as IBM Websphere™ to provide access to LANDP services through a web interface.

By providing the logic to the application in a servlet, browser based clients on non-LANDP workstations can effectively have access to LANDP services. Typically, the servlet will need to maintain one `RmtReq` object for each client it is serving.

---

## Writing applets to access LANDP

The Java applet security model restricts the rights of applet code running on a client to making a connection back to the server from which it was downloaded. By default, Applets are not allowed to access resources on the client. The `RmtReq` class needs access to the `LDPJAVA.DLL` dynamic link library and attempting to instantiate a `RmtReq` object within an applet causes a security exception.

There are two general ways around this problem that enable applets to run. Firstly, by digitally signing the applet and providing a policy file that grants the applet permission to load `LDPJAVA.DLL` on the client, the applet is able to access LANDP services as normal. This approach requires LANDP to be running on the workstation on which the applet runs.

In some cases, for instance where the applet is run on a non-LANDP platform such as a thin client type workstation, it is not possible to run LANDP on the client. In this case, an option is to provide a proxy through which the applet can direct LANDP requests. If this proxy resides on the server from which the applet is downloaded, there is no



requirement to alter the default security model. The applet can use a number of ways to communicate with the proxy including sockets, RMI, and plain HTTP requests. The client can send the Cprb it wishes to send across the wire to the proxy as a serialized object. The proxy then sends the Cprb on behalf of the client applet and passes the results back. Two example implementations of a proxy server are included in the samples provided.

---

## Writing LANDP servers in Java

The LandpServer class is provided to enable user servers to be written in Java.

The LandpServer class does not do anything itself except handle all initialisation and get LANDP remote requests as they appear and return the result back. It handles the basic IN, ES, && and \*\* functions that all servers must handle. As such, the server could be installed 'as is' in a LANDP configuration. Of course there would not be much point to this exercise since the LandpServer does not do anything constructive. Therefore it is assumed that the LandpServer class will be extended by the developer to do a specific task.

The LandpServer class is an abstract class. This means that user servers must implement an abstract method that has been defined in the LandpServer class. The abstract method in question is called checkFunction(). Java server code must implement this method. The method is called when the LandpServer code receives a request from the supervisor by a call to GETREQ. The checkFunction() method must parse the functionID to determine if it is one that is handled in this server. If checkFunction() is not implemented, the Java compiler will generate a compile time error message.

User servers written in Java must have a name that can be registered with LANDP so that the LANDP supervisor can pass requests to it. The server name will be the first 8 characters of the class name of the server. The server name is padded with spaces to a length of 8 characters internally. If the name of the server was 'ABCDEFGHJK' for instance, then the server name would be 'ABCDEFGH'. A consequence of this is that LANDP user servers written in Java cannot be part of a package hierarchy, but must reside instead within the default package. A packaged server, for instance `com.ibm.landp.server.Foo` would be erroneously registered with LANDP as a server named `COM.IBM`.

The basic LandpServer supports 2 error codes. They are:

- P1     Function not supported.
- P3     Reply Data length error.

These error codes are constants in the LandpServer class and are defined as follows:

```
final int ERR_FUNCTION_SUPP = 16797745; /* P1 Function not supported */
final int DATA_LENGTH_ERROR = 16797747; /* P3 Reply data length error */
```

Server specific error codes can be created and returned by assigning to the `server_error` field of LandpServer.

For an example implementation of a user server in Java refer to the samples included with the product.

---

## Chapter 7. Writing programs using VisualAge for COBOL

You can write two types of LANDP program using VisualAge for COBOL:

- GUI (graphical user interface) programs under OS/2
- Non-GUI programs

This chapter explains how.

---

### Writing GUI programs under OS/2

Using VisualAge for COBOL, it is possible to design a wide variety of GUI applications with buttons, scrollable entry boxes, menus, and so on. As a result, it is only possible to offer general advice to developers of LANDP GUI applications. One possible approach is to complete the example program in the *VisualAge for COBOL Getting Started* manual (see the chapter "Build Your First VisualAge for COBOL GUI Application") and then build a simple LANDP client.

One suggestion for a simple client could consist of three buttons to make an IN (Initialize) call, an II (Inquire Information) call and an EJ (Disconnect an Application Program) call to the supervisor. At each stage, the router and server return codes could be displayed in read-only entry fields.

Keep the following in mind when you build a LANDP GUI application:

- To access the CPRB you must have the following in your source:  
COPY "EHCDEFVA.CBL".
- You must make the same changes in the compile notebook that are required for non-GUI applications (see below)

GUI programs under OS/2 or Windows NT are event driven. This means that, for example, a GUI program cannot use the WM (Wait Multiple) function to wait for keyboard events. See "Event notification using graphical user interface (GUI) message posting" on page 39 for more information on event notification and GUI programs.

---

### Writing non-GUI programs

LANDP provides a sample server and a sample client to help you write non-GUI LANDP VisualAge for COBOL applications. They are called SAMPSESV.CBL and SAMP-CLI.CBL respectively.

These programs contain the line

```
COPY "EHCDEFVA.CBL".
```

This line copies the file EHCDEFVA.CBL (which can be found in the EHCO500 or EHCN500 sub-directory) into the source code. This file contains the CPRB, options control blocks, some common error codes and miscellaneous values.

For annotated listings of SAMPSEV.CBL and SAMP-CLI.CBL, see "Sample application (COBOL, OS/2 AND Windows NT)" on page 149.

---

### VisualAge for COBOL compilation settings

Before attempting to compile or link your applications or the supplied sample applications (on OS/2 or Windows NT), make the following changes to the VisualAge for COBOL compilation settings:

- On the "Syntactic" tab, check that the "Process COPY, BASIS, and REPLACE statements" option is selected.
- In the "Enter copy file search path" text field, enter the path to the directory that contains EHCDEFVA.CBL (for example, C:\EHC\EHC0500 or EHCN500)
- Select the "As-is" option of "Resolve program names"
- On the "Link" tab, fill in the path and file name of the appropriate LANDP.LIB file under "Enter library/object file name(s)". For example, C:\EHC\EHC0500\EHCOS232.LIB for OS/2 or C:\EHC\EHCN500\EHCWINNT.LIB for Windows NT.

When using VisualAge for COBOL on Windows NT, make the following further change:

- On the "System" tab, select "CDECL" as the Call Interface Convention. This calling convention will be used in your application for all calls to external functions. If you wish to call external functions with other calling conventions, use the CALLINTERFACE directive to override this setting.

## Chapter 8. VisualAge Generator Application Programming Interface

VisualAge Generator is a tool used to build and deploy multi-tier client/server applications. VisualAge Generator Developer for OS/2 and Windows NT is one of a set of workstation-based products for application development providing definition, test, generation and runtime support for both client/server and stand-alone applications. With it you can define, test and generate character-based and GUI applications.

VisualAge Generator Developer is integrated on top of VisualAge Smalltalk and allows you to define definitions and store them in the VisualAge Smalltalk repository. As you develop an application, VisualAge Generator Developer provides a graphical structure diagram that shows a hierarchical structure of the components of the application. VisualAge Generator also provides a test facility for you to interactively test your application and remove errors even as you define the application. After testing you can generate a COBOL or C++ application suitable for your execution environment.

---

### Overview

The VisualAge Generator library contains a number of part types that are used to generate an application. Some of the types that you can use are:

<b>GUI</b>	An event-driven application that contains one or more windows that represent the graphical user interface of an application.
<b>Program</b>	A set of related definitions and instructions that VisualAge Generator can generate into an executable form. The program is the specification of the order in which processes, statement groups, and other programs are run.
<b>Process</b>	A block of logic consisting of a set of processing statements surrounding a central input or output (I/O) operation.
<b>Record</b>	A collection of data items (a data structure) organized to serve a specific purpose. A working-storage record is a special type of record used to hold temporary data, including data being passed to another program.
<b>Data item</b>	A single unit of information in a record or table.
<b>Linkage table</b>	A table required if a program contains external calls. The linkage table specifies the call and conventions used.

### The LANDP Dynamic Link Library

The LANDP / VisualAge Generator Application Programming Interface (API) is handled by a Dynamic Link library (DLL) called EHCVGEN.DLL located in the EHCO500 (for OS/2) and EHCN500 (for Windows NT) directories of the LANDP product. You should copy EHCVGEN.DLL to the directory in which the LANDP run-time files were installed on the workstations on which your VisualAge Generator application will run. This directory should be included in your LIBPATH environment variable on OS/2 or your PATH environment variable on Windows NT.

## VisualAge generator

In order to access the DLL, you must use a VisualAge Generator Linkage Table. The file LINKAGE.TAB is supplied with LANDP and can be imported into your VisualAge Generator application. The contents of the file are:

```
:CALLLINK
  APPLNAME=EZSRPI
  BITMODE=32
  LIBRARY=EHCVGGEN
  LINKTYPE=DYNAMIC
  PARMFORM=OSLINK
```

### Calling LANDP servers from VisualAge Generator application programs

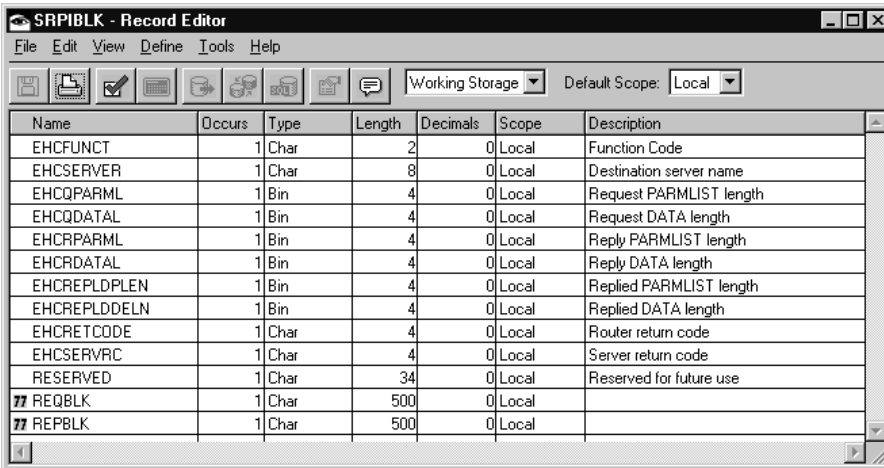
Calls to a LANDP server from a VisualAge Generator application are made as follows:

```
CALL EZSRPI SRPIBLK, REQBLK, REPBLK (NONCSP;
```

In this call EZSRPI is the main entry point in the DLL and SRPIBLK, REQBLK and REPBLK are records. The NONCSP option indicates that the called application exists outside the library. The SRPIBLK contains the major CPRB fields, the REQBLK contains the Request PARMLIST followed by the Request DATA and the REPBLK contains the Reply PARMLIST followed by the Reply DATA.

**Note:** The size and layout of the PARMLIST and DATA areas varies depending on the server being called.

Figure 5 is an example of a SRPIBLK record where the REQBLK and REPBLK have been included as level 77 data items. You may choose different names and descriptions for the first 11 items but the type and size must be as shown here. The size of the REQBLK and REPBLK depends on the LANDP servers which you intend to call. (In the following tables and examples the names shown here will be used.)



The screenshot shows the 'SRPIBLK - Record Editor' window. It has a menu bar (File, Edit, View, Define, Tools, Help) and a toolbar with icons for file operations and editing. Below the toolbar, there are two dropdown menus: 'Working Storage' and 'Default Scope: Local'. The main area contains a table with the following columns: Name, Occurs, Type, Length, Decimals, Scope, and Description.

Name	Occurs	Type	Length	Decimals	Scope	Description
EHC FUNCT	1	Char	2	0	Local	Function Code
EHC SERVER	1	Char	8	0	Local	Destination server name
EHC QPARML	1	Bin	4	0	Local	Request PARMLIST length
EHC QDATAL	1	Bin	4	0	Local	Request DATA length
EHC RPARML	1	Bin	4	0	Local	Reply PARMLIST length
EHC RDATAL	1	Bin	4	0	Local	Reply DATA length
EHC REPLDLEN	1	Bin	4	0	Local	Replied PARMLIST length
EHC REPLDLEN	1	Bin	4	0	Local	Replied DATA length
EHC RETCODE	1	Char	4	0	Local	Router return code
EHC SERVRC	1	Char	4	0	Local	Server return code
RESERVED	1	Char	34	0	Local	Reserved for future use
77 REQBLK	1	Char	500	0	Local	
77 REPBLK	1	Char	500	0	Local	

Figure 5. VisualAge Generator SRPIBLK definition

## Calling functions within the DLL

In addition to allowing the VisualAge Generator application to call LANDP servers, the DLL also contains some functions of its own.

### Handling of bit-oriented data

EHCVCEN.DLL contains two functions called HC and CH. These functions convert the ASCII representation of hexadecimal digits into their half-byte numeric equivalents or vice versa. For example '3' (X'33') becomes X'3' and X'C' becomes 'C' (X'43'). The data to be converted is put into the Request PARMLIST and the converted data is returned in the Reply PARMLIST.

**Note:** If the CH function is used and the half-byte cannot be converted into the ASCII representation of a valid hexadecimal digit it will be converted into a NULL (X'00').

The conversions are as follows:

HC	CH
00110000 ('0') ----> 0000	0000 ----> 00110000 ('0')
00110001 ('1') ----> 0001	0001 ----> 00110001 ('1')
00110010 ('2') ----> 0010	0010 ----> 00110010 ('2')
00110011 ('3') ----> 0011	0011 ----> 00110011 ('3')
00110100 ('4') ----> 0100	0100 ----> 00110100 ('4')
00110101 ('5') ----> 0101	0101 ----> 00110101 ('5')
00110110 ('6') ----> 0110	0110 ----> 00110110 ('6')
00110111 ('7') ----> 0111	0111 ----> 00110111 ('7')
00111000 ('8') ----> 1000	1000 ----> 00111000 ('8')
00111001 ('9') ----> 1001	1001 ----> 00111001 ('9')
01000001 ('A') ----> 1010	1010 ----> 01000001 ('A')
01000010 ('B') ----> 1011	1011 ----> 01000010 ('B')
01000011 ('C') ----> 1100	1100 ----> 01000011 ('C')
01000100 ('D') ----> 1101	1101 ----> 01000100 ('D')
01000101 ('E') ----> 1110	1110 ----> 01000101 ('E')
01000110 ('F') ----> 1111	1111 ----> 01000110 ('F')

Table 16 shows the contents of the SRPIBLK when using the HC function:

Table 16 (Page 1 of 2). SRPIBLK contents for HC function		
Field name	Description	Value
EHCFUNCT	Function	HC
EHCSEVER	Server	EZHC2CH
EHCQPARML	Request parmlist length	Length of hexadecimal string
EHCQDATAL	Request data length	0
EHCRCPARML	Reply parmlist length	Length of character string Note: this must be half the value in EHCQPARML

<i>Table 16 (Page 2 of 2). SRPIBLK contents for HC function</i>		
Field name	Description	Value
EHCRCRDATA	Reply data length	0
EHCRCRPLDPLEN	Replied parm list length	Length of character string
EHCRCRPLDDLEN	Replied data length	0
EHCRCRRCODE	Router return code	(1)
EHCRCRSRVC	Server return code	(1)
RESERVED		(2)
REQBLK	Request parm list+data	(3)
REPBLK	Reply parm list+data	(3)
<b>Notes:</b> <b>(1)</b> See the section on return codes later in the chapter <b>(2)</b> Reserved for LANDP internal use <b>(3)</b> Depends on the data to be converted		

Table 17 shows the contents of the SRPIBLK when using the CH function:

<i>Table 17. SRPIBLK for CH function</i>		
Field name	Description	Value
EHCRCRFUNCTION	Function	CH
EHCRCRSERVER	Server	EZHC2CH
EHCRCRQPARML	Request parm list length	Length of character string
EHCRCRQDATA	Request data length	0
EHCRCRPARML	Reply parm list length	Length of hexadecimal string Note: this must be twice the value in EHCRCRQPARML
EHCRCRDATA	Reply data length	0
EHCRCRPLDPLEN	Replied parm list length	Length of hexadecimal string
EHCRCRPLDDLEN	Replied data length	0
EHCRCRRCODE	Router return code	(1)
EHCRCRSRVC	Server return code	(1)
RESERVED		(2)
REQBLK	Request parm list+data	(3)
REPBLK	Reply parm list+data	(3)
<b>Notes:</b> <b>(1)</b> See the section on return codes later in the chapter <b>(2)</b> Reserved for LANDP internal use <b>(3)</b> Depends on the data to be converted		



### Translation from ASCII to EBCDIC and EBCDIC to ASCII

EHCVDGEN.DLL provides two functions called AE and EA. These functions translate strings of ASCII characters into EBCDIC and strings of EBCDIC characters into ASCII respectively. These functions are provided solely to support migration from applications written using LANDP Version 4 support for VisualAge Generator Developer for OS/2 V2.2. If you are writing a new VisualAge Generator application and need to perform ASCII-EBCDIC or EBCDIC-ASCII translation you should use the support provided within VisualAge Generator itself.

Table 18 shows the contents of the SRPIBLK when using the AE and EA functions.

<i>Table 18. SRPIBLK for AE and EA functions</i>		
Field name	Description	Value
EHCFUNCT	Function	AE or EA
EHCSEVER	Server	EZAE2EA
EHCQPARML	Request parmlist len.	0
EHCQDATAL	Request data length	Length of string to be translated
EHCRCPARML	Reply parmlist length	0
EHCRCDATAL	Reply data length	Length of translated string Note: this must be the same as EHCQDATAL
EHCREPLDPLEN	Replied parmlist len.	0
EHCREPLDDLEN	Replied data length	Length of translated string
EHCRCETCODE	Router return code	(1)
EHCSEVERC	Server return code	(1)
RESERVED		(2)
REQBLK	Request parmlist+data	(3)
REPBLK	Reply parmlist+data	(3)
<b>Notes:</b> (1) See the section on return codes later in the chapter (2) Reserved for LANDP internal use (3) Depends on the data to be translated		

Figure 6 on page 110 and Figure 7 on page 111 show the tables used in the translation.

	X'00'	X'01'	X'02'	X'03'	X'04'	X'05'	X'06'	X'07'
X'00'	X'00'	X'01'	X'02'	X'03'	X'04'	X'05'	X'06'	X'07'
X'08'	X'08'	X'09'	X'0A'	X'0B'	X'0C'	X'0D'	X'0E'	X'0F'
X'10'	X'10'	X'11'	X'12'	X'13'	X'B6'	X'B5'	X'16'	X'17'
X'18'	X'18'	X'19'	X'1A'	X'1B'	X'1C'	X'1D'	X'1E'	X'1F'
X'20'	X'40'	X'4F'	X'7F'	X'7B'	X'5B'	X'6C'	X'50'	X'7D'
X'28'	X'4D'	X'5D'	X'5C'	X'4E'	X'6B'	X'60'	X'4B'	X'61'
X'30'	X'F0'	X'F1'	X'F2'	X'F3'	X'F4'	X'F5'	X'F6'	X'F7'
X'38'	X'F8'	X'F9'	X'7A'	X'5E'	X'4C'	X'7E'	X'6E'	X'6F'
X'40'	X'7C'	X'C1'	X'C2'	X'C3'	X'C4'	X'C5'	X'C6'	X'C7'
X'48'	X'C8'	X'C9'	X'D1'	X'D2'	X'D3'	X'D4'	X'D5'	X'D6'
X'50'	X'D7'	X'D8'	X'D9'	X'E2'	X'E3'	X'E4'	X'E5'	X'E6'
X'58'	X'E7'	X'E8'	X'E9'	X'4A'	X'E0'	X'5A'	X'5F'	X'6D'
X'60'	X'79'	X'81'	X'82'	X'83'	X'84'	X'85'	X'86'	X'87'
X'68'	X'88'	X'89'	X'91'	X'92'	X'93'	X'94'	X'95'	X'96'
X'70'	X'97'	X'98'	X'99'	X'A2'	X'A3'	X'A4'	X'A5'	X'A6'
X'78'	X'A7'	X'A8'	X'A9'	X'C0'	X'BB'	X'D0'	X'A1'	X'41'
X'80'	X'68'	X'DC'	X'51'	X'42'	X'43'	X'44'	X'47'	X'48'
X'88'	X'52'	X'53'	X'54'	X'57'	X'56'	X'58'	X'63'	X'67'
X'90'	X'71'	X'9C'	X'9E'	X'CB'	X'CC'	X'CD'	X'DB'	X'DD'
X'98'	X'DF'	X'EC'	X'FC'	X'70'	X'B1'	X'80'	X'41'	X'B4'
X'A0'	X'45'	X'55'	X'CE'	X'DE'	X'49'	X'69'	X'9A'	X'9B'
X'A8'	X'AB'	X'AF'	X'BA'	X'B8'	X'B7'	X'AA'	X'8A'	X'8B'
X'B0'	X'41'	X'41'	X'41'	X'41'	X'41'	X'65'	X'62'	X'64'
X'B8'	X'41'	X'41'	X'41'	X'41'	X'41'	X'B0'	X'B2'	X'41'
X'C0'	X'41'	X'41'	X'41'	X'41'	X'41'	X'41'	X'46'	X'66'
X'C8'	X'41'	X'41'	X'41'	X'41'	X'41'	X'41'	X'41'	X'9F'
X'D0'	X'8C'	X'AC'	X'72'	X'73'	X'74'	X'DA'	X'75'	X'76'
X'D8'	X'77'	X'41'	X'41'	X'41'	X'41'	X'6A'	X'78'	X'41'
X'E0'	X'EE'	X'59'	X'EB'	X'ED'	X'CF'	X'EF'	X'A0'	X'AE'
X'E8'	X'8E'	X'FE'	X'FB'	X'FD'	X'8D'	X'AD'	X'BC'	X'BE'
X'F0'	X'CA'	X'8F'	X'BF'	X'B9'	X'B6'	X'B5'	X'41'	X'9D'
X'F8'	X'90'	X'BD'	X'B3'	X'41'	X'FA'	X'EA'	X'41'	X'41'

Figure 6. Conversion table, ASCII to EBCDIC

	X'00'	X'01'	X'02'	X'03'	X'04'	X'05'	X'06'	X'07'
X'00'	X'00'	X'01'	X'02'	X'03'	X'04'	X'05'	X'06'	X'07'
X'08'	X'08'	X'09'	X'0A'	X'0B'	X'0C'	X'0D'	X'0E'	X'0F'
X'10'	X'10'	X'11'	X'12'	X'13'	X'14'	X'15'	X'16'	X'17'
X'18'	X'18'	X'19'	X'1A'	X'1B'	X'1C'	X'1D'	X'1E'	X'1F'
X'20'	X'20'	X'21'	X'22'	X'23'	X'24'	X'25'	X'26'	X'27'
X'28'	X'28'	X'29'	X'2A'	X'2B'	X'2C'	X'2D'	X'2E'	X'2F'
X'30'	X'30'	X'31'	X'32'	X'33'	X'34'	X'35'	X'36'	X'37'
X'38'	X'38'	X'39'	X'3A'	X'3B'	X'3C'	X'3D'	X'3E'	X'3F'
X'40'	X'20'	X'FF'	X'83'	X'84'	X'85'	X'A0'	X'C6'	X'86'
X'48'	X'87'	X'A4'	X'5B'	X'2E'	X'3C'	X'28'	X'2B'	X'21'
X'50'	X'26'	X'82'	X'88'	X'89'	X'8A'	X'A1'	X'8C'	X'8B'
X'58'	X'8D'	X'E1'	X'5D'	X'24'	X'2A'	X'29'	X'3B'	X'5E'
X'60'	X'2D'	X'2F'	X'B6'	X'8E'	X'B7'	X'B5'	X'C7'	X'8F'
X'68'	X'80'	X'A5'	X'DD'	X'2C'	X'25'	X'5F'	X'3E'	X'3F'
X'70'	X'9B'	X'90'	X'D2'	X'D3'	X'D4'	X'D6'	X'D7'	X'D8'
X'78'	X'DE'	X'60'	X'3A'	X'23'	X'40'	X'27'	X'3D'	X'22'
X'80'	X'9D'	X'61'	X'62'	X'63'	X'64'	X'65'	X'66'	X'67'
X'88'	X'68'	X'69'	X'AE'	X'AF'	X'D0'	X'EC'	X'E8'	X'F1'
X'90'	X'F8'	X'6A'	X'6B'	X'6C'	X'6D'	X'6E'	X'6F'	X'70'
X'98'	X'71'	X'72'	X'A6'	X'A7'	X'91'	X'F7'	X'92'	X'CF'
X'A0'	X'E6'	X'7E'	X'73'	X'74'	X'75'	X'76'	X'77'	X'78'
X'A8'	X'79'	X'7A'	X'AD'	X'A8'	X'D1'	X'ED'	X'E7'	X'A9'
X'B0'	X'BD'	X'9C'	X'BE'	X'FA'	X'9F'	X'F5'	X'F4'	X'AC'
X'B8'	X'AB'	X'F3'	X'AA'	X'7C'	X'EE'	X'F9'	X'EF'	X'F2'
X'C0'	X'7B'	X'41'	X'42'	X'43'	X'44'	X'45'	X'46'	X'47'
X'C8'	X'48'	X'49'	X'F0'	X'93'	X'94'	X'95'	X'A2'	X'E4'
X'D0'	X'7D'	X'4A'	X'4B'	X'4C'	X'4D'	X'4E'	X'4F'	X'50'
X'D8'	X'51'	X'52'	X'D5'	X'96'	X'81'	X'97'	X'A3'	X'98'
X'E0'	X'5C'	X'E1'	X'53'	X'54'	X'55'	X'56'	X'57'	X'58'
X'E8'	X'59'	X'5A'	X'FD'	X'E2'	X'99'	X'E3'	X'E0'	X'E5'
X'F0'	X'30'	X'31'	X'32'	X'33'	X'34'	X'35'	X'36'	X'37'
X'F8'	X'38'	X'39'	X'FC'	X'EA'	X'9A'	X'EB'	X'E9'	X'FF'

Figure 7. Conversion table, EBCDIC to ASCII

### Return codes

The following server error codes are returned by EHCVGEN.DLL in addition to those returned by LANDP servers:

#### **P1 (X'01005031')**

- The server is EZAE2EA but the function is neither AE nor EA.
- The server is EZHC2CH but the function is neither HC nor CH.

#### **P2 (X'01005032')**

- The REQBLK or REPBLK parameter (or both) is missing from the EZSRPI call.
- The server is EZAE2EA but the request and reply data lengths differ.
- The server is EZHC2CH but the request data length is not zero or the request parm list length is zero.
- The server is EZHC2CH and the function is CH but the reply parm list length is not twice the request parm list length.
- The server is EZHC2CH and the function is HC but the reply parm list length is not half the request parm list length.

#### **P4 (X'01005034')**

- The server is EZHC2CH and the function is HC but the request parm list contains invalid hexadecimal characters.

### Testing applications

Once you have defined your program, including appropriate processes and records, you can test it from within the VisualAge Generator environment. In order to test you need to specify the linkage table to use. From the VisualAge Organiser, select the Options menu followed by Preferences. On the "VAGEN - Test Linkage" tab, select the EZSRPI linkage table.

### Generating an application

To generate your application, follow the instructions in the VisualAge Generator Generation Guide. Set the linkage table to EZSRPI by opening the Generation Options dialog and choosing the Input Options tab.

Once your application has been generated and prepared, you can run it, ensuring that EHCVGEN.DLL is in your LIBPATH environment variable on OS/2 or your PATH environment variable on Windows NT.

## Chapter 9. LANDP for OS/2 REXX application programming interface

LANDP for OS/2 provides extensions to OS/2 REXX to support LANDP requests to, and replies from, REXX programs. These extensions are handled by the dynamic link library EHCREXX.DLL. Before using any LANDP function call you must register the LANDP functions to be used:

```
/* Register the LANDP functions */

call rxfuncadd 'GETREQ' , 'EHCREXX', 'RXGETREQ'
call rxfuncadd 'GETRPLY', 'EHCREXX', 'RXGETRPLY'
call rxfuncadd 'RMTAREQ', 'EHCREXX', 'RXRMTAREQ'
call rxfuncadd 'RMTREQ', 'EHCREXX', 'RXRMTREQ'
call rxfuncadd 'RMTRPLY', 'EHCREXX', 'RXRMTRPLY'
call rxfuncadd 'SRVINIT', 'EHCREXX', 'RXSRVINIT'
```

You can register any or all of the LANDP functions using the method given above. The only rule you must observe is to register the function before using it.

Alternatively, you can register all the LANDP functions together:

```
call rxfuncadd 'EHCLOADFUNCS', 'EHCREXX', 'EHCLOADFUNCS'
call EHCLOADFUNCS
```

To deregister the LANDP functions:

```
/* Deregister the LANDP functions */

call rxfuncdrop 'GETREQ'
call rxfuncdrop 'GETRPLY'
call rxfuncdrop 'RMTAREQ'
call rxfuncdrop 'RMTREQ'
call rxfuncdrop 'RMTRPLY'
call rxfuncdrop 'SRVINIT'
```

If you used the EHCLOADFUNCS call to load all the LANDP functions, you can use the following method:

```
call EHCUNLOADFUNCS
```

REXX stem variables are used to construct and inspect LANDP structures used during LANDP function calls. Stem variables representing a LANDP structure have a name in the form:

```
<name>.<LANDP structure field name>
```

where <name> is a user-defined variable name and <LANDP structure field name> is the name of a LANDP structure field name as defined in EHCDEFH.H (for example, EHCFUNCT).

**Note:** There is one exception. The SRVINIT options field, which holds the service names, is called `service_names` and *not* `service_name_list`.

User-defined variable names must not exceed 32 characters in length.

Use this form where the LANDP structure contains a pointer to another structure (for example, the RMTREQ options structure contains a pointer to a NOWAIT structure):

```
<name>.<LANDP structure name>.<LANDP structure field name>
```

For example:

```
rmtreq_opts.nowait_parmad.reply_handle
```

The struct\_size and reserved fields in the LANDP options structures are initialized automatically.

The LANDP REXX interface supports stem variables representing the following LANDP structures:

- EHC\_CPRB
- EHC\_GETREQ\_OPTS
- EHC\_GETRPLY\_OPTS
- EHC\_NOWAIT\_PARM
- EHC\_RMTREQ\_OPTS
- EHC\_SRVINIT\_OPTS

Use a REXX number where the definition in EHCDEFC.H refers to an integer. For example:

```
cprb.ehctimeout = 0
```

Use a REXX string where the definition refers to a character array. For example:

```
cprb.ehcserver = 'SPV      '
```

Use a REXX string where the CPRB definition in EHCDEFC.H refers to a pointer (parameter and data addresses - request and reply). For example:

```
cprb.ehcqparmad = 'L'
```

Here is a fragment of code that makes an AA call to a server called SERVER, using the RMTREQ (remote request) call with the NoWait option. One character is supplied in the Request PARMLIST area and up to five characters may be received in the Reply PARMLIST area.

```

cprb.ehcfunct = 'AA'
cprb.ehcserver = 'SERVER '
cprb.ehcqparm1 = 1
cprb.ehcqdata1 = 0
cprb.ehcrparm1 = 5
cprb.ehcrdata1 = 0

cprb.ehcqparmad = '?'

rmtreq_opts.nowait_parmad.reply_handle = ''
rmtreq_opts.nowait_parmad.nowait_flags = 0
rmtreq_opts.nowait_parmad.window_handle = 0
rmtreq_opts.nowait_parmad.message_id = 0

call RMTREQ cprb, rmtreq_opts

```

A sample server (SAMPSESV.CMD) and client (SAMP-CLI.CMD) can be found in the samples directory under the EHCO500 sub-directory of the main LANDP installation directory.





## Chapter 10. Testing your application programs

The system verification programs are:

- SVPCPRB.EXE for LANDP for DOS and LANDP for OS/2
- SVPCPRBN.EXE for LANDP for Windows NT
- DCZYXSVP and DCZYSVP for LANDP for AIX
- SVPCPRBW.EXE for LANDP for DOS (and Windows 3.1/3.11)

These programs enable you to edit the CPRB and issue calls to the RMTREQ routine. You can issue your application requests by filling in all the necessary input fields in the system verification programs panels. You get in return the LANDP program component response, which is displayed on the same panel.

The programs for use in a Windows 3.1 or Windows NT environment are described in “Testing with Windows 3.1/3.11 or Windows NT” on page 121.

The system verification programs supplement the standard personal computer system diagnostic procedures and the diagnostic procedures for the adapters. You can use them for the following purposes:

- Verifying the correct operation of the LANDP program components without creating a special application
- Testing your client applications
- Testing your server programs
- Training in application writing

You must load the LANDP program components, from which you want to request services, before starting the system verification programs.

Before the first verification program is run, you must enter an IN request to SPV, with lengths 26, 0, 26, and 0. See “Using your own SVPCPRB exit server” on page 124 for more information.

**LANDP for AIX:** The system verification program DCZYSVP supports conversion of multiple-byte character set (MBCS) character fields from AIX code pages into host and PC code pages and conversely. The MBCS convertor is determined by the LANG environment variable:

LANG value	AIX code page	Host code page	PC code page
ko_KR	IBM-eucKR	IBM-933	IBM-949
zh_TW	IBM-eucTW	IBM-937	IBM-938

## Defining a specific test

This panel is displayed after calling the SVPCPRB.EXE or DCZYSVP program:

SYSTEM VERIFICATION PROGRAM

INPUT
Function: yy
Timeout: yy
Server name: yyyyyyy
Request parameter length: yy
Request data length: yyyy
Reply parameter length: yy
Reply data length: yyyy
Request parameter area:
yy
Request data area:
yy
yy
yy
yyyyyyyyyyyyyyyyyyyy
Reply parameter area:
xx
Reply data area:
xx
xx
xx
xxxxxxxxxxxxxxxxxxxx
Enter=Process
Offset 0
Esc/F3=End
F4=ASC-EBC
F5=EBC-ASC
F6=Char-Hex
F7=Reply->Request
Hex-Ch=CHR
Insert OFF

OUTPUT
PC identifier: xx
Router return code: xxxx
Server return code: xxxx
Replied parameter length: xx
Replied data length: xxxx
Elapsed time (secs.): xxx.xx

DCZYSVP has a slightly different layout and its function key assignments are different:

Enter=Process
Offset 0
F3=End
F7=Rep->Req
F4=ASC-HOST
F8=ASC-PC
F5=HOST-ASC
F9=PC-ASC
F6=CHR-HEX
Hex-Ch=CHR
Insert OFF

When the fields are filled in with suitable data, the entered **INPUT** Function uses that data for performance. The data is used for one function only. Then, the system verification program stops and displays the results. This sequence can be repeated as many times as you require.

When using this panel, you should take into account that:

- The fields initialized with zeros must have numeric contents.
- The fields initialized with blanks can have any alphanumeric data.
- Only the first 256 bytes of both Request DATA and Reply DATA are displayed, and only 256 bytes of data can be entered. However, DATA lengths can be specified up to a maximum size of 4096 bytes.
- Only Request DATA and Reply DATA area fields can be converted from ASCII to EBCDIC, and from EBCDIC to ASCII.

- Only the following fields can be edited in hexadecimal format:
  - Function
  - Server Return Code
  - Router Return Code
  - Request PARMLIST Area
  - Request DATA Area
  - Reply PARMLIST Area
  - Reply DATA Area

In hexadecimal mode, only 128 bytes are displayed.

The fields used for entering information are:

<b>Function</b>	Function code.
<b>Timeout</b>	Timeout for the request, in seconds.
<b>Server name</b>	Requested resource name.
<b>Request parameter length</b>	Length of the input parameter area.
<b>Request data length</b>	Length of the input data area.
<b>Reply parameter length</b>	Length of the output parameter area.
<b>Reply data length</b>	Length of the output data area.
<b>Request parameter area</b>	Parameters supplied to the server.
<b>Request data area</b>	Data supplied to the server.

The fields used for displaying information are:

<b>PC identifier</b>	Workstation identifier.
<b>Router return code</b>	Return code X'00000000', in hexadecimal representation, and 'OK', in character representation, mean successful routing. For other return codes, refer to the <i>LANDP Problem Determination</i> book.
<b>Server return code</b>	Return code X'00000000', in hexadecimal representation, and 'OK', in character representation, mean successful completion. For other return codes, refer to the <i>LANDP Problem Determination</i> book.
<b>Replied parameter length</b>	Length of the significant output parameter area.
<b>Replied data length</b>	Length of the significant output data area.
<b>Reply parameter area</b>	Parameters returned by the server.
<b>Reply data area</b>	Data returned to the application.
<b>Elapsed time</b>	Performance time, in seconds. When greater than 15 minutes, the field is filled with an asterisk (*).

## testing programs

When you select hexadecimal mode for LANDP for DOS or OS/2, this is the panel layout:

```
SYSTEM VERIFICATION PROGRAM
INPUT Function: yyyy Timeout: yy      OUTPUT PC identifier: xx
        Server name: yyyyyyyy         Router return code: xxxxxxxx
        Request parameter length: yy   Server return code: xxxxxxxx
        Request data length: yyyy      Replied parameter length: xx
        Reply parameter length: yy     Replied data length: xxxx
        Reply data length: yyyy        Elapsed time (secs.): xxx.xx

Request parameter area:
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
Request data area:
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
Reply parameter area:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Reply data area:
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxx
Enter=Process   Esc/F3=End   F4=ASC-EBC   F5=EBC-ASC   F6=Char-Hex
Offset 0       F7=Reply->Request   Hex-Ch=HEX   Insert OFF
```

## Using the keyboard

The keys used to enter data for the SVPCPRB.EXE and DCZYSVP program panels, and to start and control test performance, are listed below:

- |                  |  |
|------------------|--|
| <b>Ctrl+End</b>  | Erases from the cursor position to the end of the field (not DCZYSVP).   |
| <b>Tab</b>       | Moves from one field to another, backward or forward, depending on whether the shift key is pressed at the same time or not (not DCZYSVP). |
| <b>Enter</b>     | Starts a single function performance.  |
| <b>F3 or Esc</b> | Ends the system verification program.  |
| <b>F4</b>        | Converts from ASCII to EBCDIC. In DCZYSVP, F4 converts from ASCII to host format (see page 117).   |
| <b>F5</b>        | Converts from EBCDIC to ASCII. In DCZYSVP, F5 converts from host format to ASCII (see page 117).   |
| <b>F6</b>        | Toggles between hexadecimal format and character format.   |
| <b>F7</b>        | Moves the first Replied DATA length bytes from the Reply DATA area to the Request DATA area.   |
| <b>F8</b>        | In DCZYSVP, converts from ASCII to PC format (see page 117).   |
| <b>F9</b>        | In DCZYSVP, converts from PC format to ASCII (see page 117).   |

<b>Home</b>	Moves the cursor directly to the INPUT Function field from anywhere on the panel (not DCZYSVP).
<b>Ins</b>	Toggles between insert and overwrite modes.
<b>Delete</b>	Operates as usual.
<b>Backspace</b>	Operates as usual.

## DCZYXSVP test program

DCZYXSVP is the Motif version of DCZYSVP. It is functionally similar, but has an additional feature—the ability to save and recall CPRBs. These are available through the Save and Recall items of the File menu bar choice.

---

## Testing with Windows 3.1/3.11 or Windows NT

The SVPCPRBW.EXE program for Windows 3.1/3.11 and the SVPCPRBN.EXE program for Windows NT applications are similar to that for OS/2 and DOS, but with some changes. Their main characteristics are:

- The Windows graphical user interface
- Function-by-function execution of server and supervisor local calls
  - No application code is required
- Verify correct operation of the system:
  - Correct CPRB fields
  - Correct PARMLIST and DATA areas
  - Correct sequence of calling functions
  - Return codes to be handled by the application
- Training tool for application programmers
- Diagnostics tools for system operators
- Can be invoked from any Windows 3.1/3.11 or Windows NT system and can use loaded servers
- Display LANDP system information:
  - Servers and applications status
  - Local machine information

## Invocation

To run the tool under Windows 3.1/3.11, copy the following files to a directory on your system from the EHCD500 directory of the main LANDP directory:

SVPCPRBW.EXE  
SVPCPRBW.HLP

The tool can then be run by selecting the Run... option from the Program Manager File menu and entering the full path of the file, for example `c:\landp\svpcprbw.exe`.

For Windows NT, copy the following files from the EHCN500 directory:

## testing programs

SVPCPRBN.EXE  
SVPCPRBN.CNT  
SVPCPRBN.HLP  
EHCXLATE.DLL

The tool can then be run by selecting the Run... option from the Start menu and entering the full path of the file, for example `c:\landp\svpcprbn.exe`, or by double clicking the icon from Windows Explorer.

## Menu options

There are two main options:

- Verify:** Options to show LANDP system information and to send the CPRB to the server. The option to show system information is not supported on Windows NT.
- Help:** User help and product information

### LANDP system information

If you select the System Information item from the Verify menu, you are shown the status of servers and applications, and global information about the local machine.

L.Access indicates whether you have local access to the item. S/A shows whether it is a server or an application.

### Send CPRBs

If you choose to send CPRBs from the Verify menu, you are shown the following dialog:

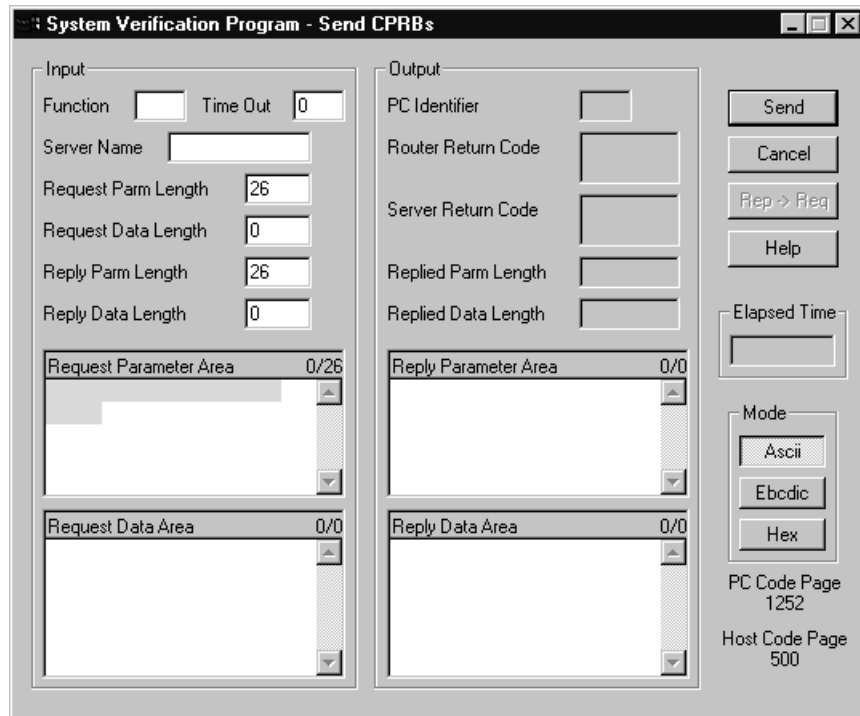


Figure 8. System verification program - send CPRBs

X fields are alphanumeric. 9 fields are numeric.

The push buttons are:

- Send:** sends the CPRB to the server
- Cancel:** ends the dialog
- Rep → Req:** copies the Reply DATA area into the Request DATA area
- Help:** gives user help information.

The “Mode” radio buttons are:

- Character:** displays the Request DATA and Reply DATA areas, the Request PARMLIST and Reply PARMLIST areas, and the *server* and *router return code* fields in character mode
- Hexadecimal:** displays the same fields in hexadecimal mode.

The “Character Code” radio buttons are:

- ASCII:** displays the Request DATA and Reply DATA areas in ASCII
- EBCDIC:** displays the same fields in EBCDIC.

DATA and PARMLIST areas are restricted to a maximum of 4096 bytes. You can see only the first 1365 bytes in hexadecimal mode when using SVPCPRBW.EXE.

### Parameter and data entry fields

The default data entry mode for these fields is overtype and is denoted by a block cursor. The mode can be toggled between overtype and insert by using the 'Insert' key.

On the OS/2 SVPCPRB utility and the Windows 3.1 SVPCPRBW utility, the options to change parameter and data areas (between ASCII, EBCDIC, and hexadecimal) carry out conversion on the data itself.

In the Windows NT SVPCPRBN utility, the mode buttons do not convert the existing data, but alter the way in which keystrokes are interpreted.

In ASCII mode, data is displayed as though it were ASCII characters. For example, a data byte of 0x45 is displayed as 'E', and therefore an uppercase E from the keyboard is stored as 0x45.

In EBCDIC mode, an uppercase E from the keyboard is stored as 0xC5, the EBCDIC code for that character.

Because data is never translated, it is possible to switch modes repeatedly without corrupting the data. However, the correct entry mode must be selected before entering data. For example, to send data to the host in EBCDIC, select the EBCDIC mode button before entering data.

---

### Using your own SVPCPRB exit server

You can use your own SVPCPRB exit server (EHCSVPUE) with the LANDP for DOS and OS/2 SVPCPRB to check user requests before the request is issued by SVPCPRB.

When an EHCSVPUE server is defined in a workgroup, the SVPCPRB sends the CPRB built from the user input to the EHCSVPUE server before issuing the CPRB to the requested server. The EHCSVPUE server can check and modify the CPRB before replying to SVPCPRB.

The EHCSVPUE exit server can be used to restrict user access to other servers. For example, you can prevent users writing to a shared file but allow them to read the same file.



**Example:** The IN function is issued, after which the following window appears:

System Verification Program - Send CPRBs

INPUT	OUTPUT
Server Name: SPV Function: IN Time Out: 0 Request Parm Length: 26 Request Data Length: 0 Reply Parm Length: 26 Reply Data Length: 0	PC Identifier: Z0 Elapsed Time: 00'00.05'' Router Return Code: OK Server Return Code: OK Replied Parm Length: 5 Replied Data Length: 0
Request Parameter Area -----	Reply Parameter Area ----- 400001
Request Data Area -----	Reply Data Area -----

< Send >  
< Cancel >  
< Rep -> Req >  
< Help >

Mode  
  
(x) Character  
( ) Hexadecimal  
  
Offset: 999  
  
Char Code  
  
(x) ASCII  
( ) EBCDIC

**testing programs**

## Chapter 11. Sample application programs

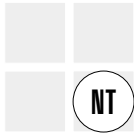
This chapter provides annotated lists of some of the supplied sample programs. The chapter can be used for reference or tutorial purposes.

The contents are:

- “Sample application (C, Windows NT),” client and server
- “Sample application (COBOL, OS/2 AND Windows NT)” on page 149, client and server
- “Sample client application (COBOL), DOS and OS/2” on page 170, client only

---

### Sample application (C, Windows NT)



The client and server application programs work together. The annotated listings are:

- “Sample client application LDPCMAIN.C”
- “Sample user server LDPSMAIN.C” on page 139

These programs are written in Microsoft Visual C++.

### Sample client application LDPCMAIN.C

Pages 128 to 136 show a sample WINDOWS NT program that illustrates the use of the LANDP Common Application Programming Interface (CAPI). The code of this sample is supplied in directory EHCN500\SAMPLES\CLIENT.

To highlight the interface to LANDP, the application is kept deliberately simple. The most complex applications use the same interface. This sample client is used in conjunction with the sample server LDPSMAIN.C. (See “Sample user server LDPSMAIN.C” on page 139.)

The program contains a standard Windows NT entry-point module LDPCMAIN.C (page 129) and an application module LDPCPROC.C (following pages) that is called by LDPCMAIN.C. Figure 9 on page 128 shows the flows in LANDP, LDPCPROC, and LDPSMAIN.

#### LDPCMAIN.C:

1. Defines a class.
2. Creates a window.
3. Displays the window.
4. Gets messages from the window and calls LDPCPROC.C to process each message as it is received.

## sample client program, C, Windows NT

**LDPCPROC.C** is called by **LDPCMAIN.C**, and, depending on the message, processes as follows:

- When the window is created, issues a LANDP initialize function (IN) and a LANDP start posting events function (SP) to the LANDP supervisor.
- When a key is pressed in the window created, issues a user defined function (KP) to a user-defined server.
- When an Asynchronous Event notification (Z5) is received by LANDP from the server, a Windows message is generated and passed to the client. The client then issues LANDP query function (QE) to query the event received.
- When the window is destroyed, issues a stop posting events function (TP).

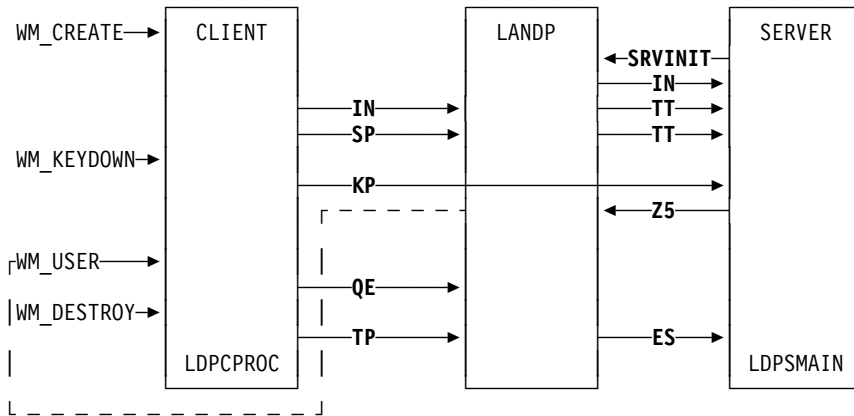


Figure 9. Sample Windows application - events and function calls in client and server

### Windows NT entry-point module LDPCMAIN.C

LDPCMAIN.C (listed on facing page) is a standard Windows NT entry module. The numbers of the following notes refer to the highlighted numbers in the listing:

1. `ldpcproc.h` is a header file containing the prototype of the function `LDPWndProc` (note 1 on page 132), which receives messages from Windows NT. `ldpcproc.h` is listed on page 138.
2. This code displays a window, updates the window, translates data entered in the window, and sends messages (`DispatchMessage`) to `LDPCPROC.C`,

```

/* Include file and preamble */
#include <windows.h>

1
#include "ldpcproc.h"
/* Main entry point, standard Windows code, no LANDP code */
int PASCAL WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpszCmdLine,
                   int nCmdShow)
{
    WNDCLASS wc;
    HWND hWnd;
    MSG msg;

    wc.lpszClassName = "LDPCClass";
    wc.hInstance = hInstance;
    wc.style = CS_VREDRAW | CS_HREDRAW;
    wc.lpfnWndProc = LDPWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hIcon = NULL;
    wc.lpszMenuName = NULL;
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);

    if (!RegisterClass(&wc))
    {
        MessageBox(NULL, "LANDP Class Register Failed", "LDPCMAIN Error", MB_OK);
    }

    hWnd = CreateWindow("LDPCClass",
                       "LDP Test",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT,
                       CW_USEDEFAULT,
                       CW_USEDEFAULT,
                       CW_USEDEFAULT,
                       NULL,
                       NULL,
                       hInstance,
                       NULL);

2
    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);
    while(GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return 0;
}

```

### Windows NT application module LDPCPROC.C

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page.

1. The two include files are:
  - windows.h, the standard Windows NT header file, which is in the system directory
  - ehcdefc.h (see page 9) is assumed to be in the same directory as the sample application, and contains the following structures:
    - Connectivity programming request block (CPRB)
    - RMTREQ options control block (EHC\_RMTREQ\_OPTS)
    - GETRPLY options control block (EHC\_GETRPLY\_OPTS)
2. For each LANDP function used in the program, define the function code as the corresponding ASCII hexadecimal value.
3. Define mycprb as the CPRB for this program. EHC\_CPRB is a type defined in ehcdefc.h (see note 1).
4. Define two character arrays, parm\_area and data\_area, which are used by the client program for sending and receiving parameters and data in its communication with the server.
5. Define the character array supervisor to contain the name of the supervisor. The length of the array is 9 to allow for the length of the CPRB field ehcsrvr (8 bytes) plus the end-of-string null byte. The assigned value "SPV ", is padded with five blanks to fill the EHC\_SERVER field.
6. Define the character array myldpsrv to contain the name of the server. The length of the array is 9 to allow for the length of the CPRB field ehcsrvr (8 bytes) plus the end-of-string null byte. The assigned value "LDPSMAIN" needs no padding to fill the ehcsrvr field.
7. Define the variable retcode as unsigned long. The program uses this variable to store the return code from LANDP function calls.
8. To pass and receive parameter data for the Start Posting Events (SP) function call, define a structure with one structure variable mySPparms. The structure has two members, the window handle and the message identity.
9. To contain application data for the Start Posting Events (SP) function call, define a character array SPdata with the value MYLDPSMAIN. This is a predefined value that is recognized by the server application.

```

/* 1 Include file and preamble
#include <windows.h>
#include "ehcdefc.h"

/* 2 Define CPRB function codes (Supervisor and Server requests) and events */
#define IN 0x494E
#define SP 0x5350
#define TP 0x5450
#define QE 0x5145
#define KP 0x4B50

/* 3 Declare a CPRB for processing requests */
EHC_CPRB mycprb;

/* 4 Declare parameter and data areas for use with the CPRB */
char parm_area[26];
char data_area[26];

/* 5 Define name of supervisor for all supervisor calls */
char supervisor[9] = "SPV    ";

/* 6 Define name of the server for all server calls */
char myldpserv[9] = "LDPSMAIN";

/* 7 Declare return code */
unsigned long retcode;

/* 8 Declare a structure to contain the parameter data of the Start Posting
/* Events (SP) function call
struct {
    HWND WndHandle;    /* Window handle to receive message */
    UINT MessageType; /* Message identity */
} mySPparms;

/* 9
Define the data sent in the Start Posting Events (SP) function call
char SPdata[11] = "MYLDPSMAIN"; /* data agreed with server application */

```

## Windows NT application module LDPCPROC.C (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page. (For the names of CPRB fields, see page 5).

1. This is standard Windows NT code to receive a call from the entry module and initiate this program. The switch statement uses the value of the UINT parameter, `uMessage`, to determine which case routine is executed.
2. The case statement identifies the code to be executed for a window creation event.

The following four lines:

- Put the address of `parm_area` (defined on page 130, note 4) into both the request and reply parameter address areas of the CPRB.
  - Put the address of `data_area` (defined on page 130, note 4) into both the request and reply data address areas of the CPRB.
3. For the Initialize function, no parameters or data are involved, therefore set the four CPRB parameter and data length fields to zero. Set the `ehcfuncnt` field to the Initialize code `IN` (defined on page 130, note 2).
  4. The Initialize function is a request to the LANDP supervisor, so set the CPRB field `ehcserver` to the value of the predefined variable `supervisor`.
  5. This line is the standard LANDP function call and is the same for all requests in the program.
    - `retcode` (defined on page 130, note 7) holds the return code.
    - `RMTREQ` is the common LANDP API routine.
    - `EHC_RESERVED` is a required parameter to be entered as shown.
    - `mycprb` (defined on page 130, note 3) is the address of the CPRB, which is always required by `RMTREQ`. The contents of the CPRB distinguish one request from another. For this request, the CPRB specifies `IN` as the function code and the supervisor as the server.
  6. Issue an error message if the function request fails.
  7. For the Start Posting Events function (SP), set the CPRB as follows:
    - Set the request parameter length to 8 bytes. The parameters are the Windows handle and the message identity in `mySPparms` (defined on page 130, note 8), with a combined length of 8 bytes.
    - Set the request data length to 10 bytes, the length of `SPdata` (defined on page 130, note 9).
    - Set the reply parameter and data lengths to 0 bytes.
    - Set the function code to `SP` (defined on page 130, note 2).
  8. These five statements prepare the CPRB for the function request.
    - Copy the character array `supervisor` (defined on page 130, note 5), into the server field of the CPRB.
    - Assign the window handle to the `WndHandle` member of the structure variable `mySPparms`.
    - Assign the message identity to the `MessageType` member of the structure variable `mySPparms`.
    - Copy `mySPparms` to the request parameter field addressed by the CPRB.
    - Copy `SPdata` to the request data field addressed by the CPRB. `SPdata` contains a predefined value that is recognized by the server application. (defined on page 130, note 9).



9. This is the standard LANDP call (*note 5 above*). The CPRB has been set as described in notes 7 and 8 above. If the request fails, an error message is issued as in note 6 above.

---

```

LRESULT CALLBACK LDPWndProc(HWND hWnd, UINT uMessage,
1      WPARAM wParam, LPARAM lParam)
{
    LRESULT lResult = 0L;
    HDC hDC;
    PAINTSTRUCT ps;
    switch (uMessage)
    {
/* 2 Window Creation Event */
    case WM_CREATE:
        mycprb.ehcqparamd = (char*)param_area;
        mycprb.ehcrparamd = (char*)param_area;
        mycprb.ehcqdataad = (char*)data_area;
        mycprb.ehcrdataad = (char*)data_area;
/* 3 Initialize with LANDP and check return codes */
        mycprb.ehcqparaml = 0;          /* Request parameter length */
        mycprb.ehcqdatal = 0;          /* Request data length */
        mycprb.ehcrparaml = 0;         /* Reply parameter length */
        mycprb.ehcrdatal = 0;          /* Reply data length */
        mycprb.ehcfunct = IN;          /* Function ID */

4      memcpy(mycprb.ehcserver,supervisor,8); /* Destination resource name */
5      retcode = RMTREQ (&mycprb, EHC_RESERVED);
6      if (retcode != 0)
        {
            MessageBox(NULL, "LANDP IN TO SPV Failed", "LDPCMAIN Error" ,MB_OK);
        }
/* 7 SP Function call*/
        /* Call the Start Posting Events function and check return codes */
        mycprb.ehcqparaml = 8;          /* Request parameter length */
        mycprb.ehcqdatal = 10;         /* Request data length */
        mycprb.ehcrparaml = 0;         /* Reply parameter length */
        mycprb.ehcrdatal = 0;          /* Reply data length */
        mycprb.ehcfunct = SP;          /* Function ID */
8      memcpy(mycprb.ehcserver,supervisor,8); /* Destination resource name */
        mySPparms.WndHandle = hWnd;    /* Window handle to receive message*/
        mySPparms.MessageType = WM_USER; /* Message identity */
        memcpy(mycprb.ehcqparamd,&mySPparms,sizeof(mySPparms));
        memcpy(mycprb.ehcqdataad,SPdata,10); /* data agreed with server */
9      retcode = RMTREQ (&mycprb, EHC_RESERVED);
        if (retcode != 0)
        {
            MessageBox(NULL, "LANDP SP TO SPV Failed", "LDPCMAIN Error" ,MB_OK);
        }
    }
}

```

### Windows NT application module LDPCPROC.C (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page. (For the names of CPRB fields, see page 5).

1. Contiguous case statements identify the start of a routine that is to be executed for any of the conditions specified by the statements. Here, the conditions are the pressing of the left, middle, or right mouse button, or of any keyboard key.
2. This is a function call to a user-written server, LDPSMAIN.C. No parameters or data are passed, so set all the CPRB parameter and data length fields to zero. Set the function code to KP (*defined on page 130, note 2*), recognized by LDPSMAIN as Key-Pressed.
3. Copy the character array myldpserv (*defined on page 130, note 6*), which contains the value "LDPSMAIN", to the CPRB server field.
4. Set the CPRB request and reply parameter and data address fields to the addresses of parm\_area and data\_area (*defined on page 130, note 4*).
5. Issue the standard LANDP call (*explained in note 5, page 132*). This call is directed to a user-defined server LDPSMAIN.C, but it is exactly the same as the calls to the supervisor. The contents of the CPRB define the type of function, the server, and the input and output parameter and data fields.
6. If the function call fails, issue a message.
7. The case statement identifies a routine to be executed for a user event notification. For the Query Event function (QE), set the CPRB as follows:
  - Set the request parameter length to 8 bytes. The parameters are the Windows handle and the message identity in mySPparms (*defined on page 130, note 8*) with a combined length of 8 bytes.
  - Set the request data length to 2 bytes, the length of wParam.
  - Set the reply parameter length to 0 bytes.
  - Set the reply data length to 10 bytes.
  - Set the function code to QE (*defined on page 130, note 2*).
8. The following five statements:
  - Copy the character array supervisor (*defined on page 130, note 5*), into the server field of the CPRB.
  - Assign the window handle to the WndHandle member of the structure variable mySPparms.
  - Assign the message identity to the MessageType member of the structure variable mySPparms.
  - Copy mySPparms to the request parameter field addressed by the CPRB.
  - Copy wParam to the request data field addressed by the CPRB.
9. Issue the standard LANDP call (*explained in note 5, page 132*).
10. Issue a message based on the return code from the call.

```

1  /* Keyboard key or Mouse Button pressed Event                                */
    case WM_LBUTTONDOWN: /* Any mouse button pressed */
    case WM_MBUTTONDOWN:
    case WM_RBUTTONDOWN:
    case WM_KEYDOWN:     /* Any keyboard key is pressed */

2  /* KP Function call - gives client process id to Server (LDPSMAIN)          */
    mycprb.ehcqparm1 = 0;          /* Request parameter length          */
    mycprb.ehcqdata1 = 0;          /* Request data length                */
    mycprb.ehcrparm1 = 0;          /* Reply parameter length             */
    mycprb.ehcrdata1 = 0;          /* Reply data length                  */
    mycprb.ehcfunct = KP;          /* Function ID                        */

3  memcpy(mycprb.ehcserver,myldpserv,8); /* Destination resource name          */
4  mycprb.ehcqparmad = (char*)parm_area;
    mycprb.ehcrparmad = (char*)parm_area;
    mycprb.ehcqdataad = (char*)data_area;
    mycprb.ehcrdataad = (char*)data_area;
5  retcode = RMTREQ (&mycprb, EHC_RESERVED);
6  if (retcode != 0)
    {
        MessageBox(NULL, "LANDP KP TO LDPSMAIN Failed", "LDPCMAIN Error",MB_OK);
    }
    lResult = DefWindowProc(hWnd,uMessage, wParam, lParam);
    break;

7  /* Event Triggered by Server's Asynchronous Event Notification            */
    case WM_USER:
        /* Query the Event Received                                          */
        mycprb.ehcqparm1 = 8;          /* Request parameter length          */
        mycprb.ehcqdata1 = 2;          /* Request data length                */
        mycprb.ehcrparm1 = 0;          /* Reply parameter length             */
        mycprb.ehcrdata1 = 10;         /* Reply data length                  */
        mycprb.ehcfunct = QE;          /* Function ID                        */

8  memcpy(mycprb.ehcserver,supervisor,8); /* Destination resource name          */
    mySPparms.WndHandle =hWnd;
    mySPparms.MessageType = WM_USER;
    memcpy(mycprb.ehcqparmad,&mySPparms,sizeof(mySPparms));
    memcpy(mycprb.ehcqdataad,&wParam,2);
9  retcode = RMTREQ (&mycprb, EHC_RESERVED);
10 if (retcode != 0)
    {
        MessageBox(NULL, "LANDP QE TO SPV Failed", "LDPCMAIN Error"      ,MB_OK);
    }
    else
    {
        MessageBox(NULL, "LANDP EVENT WM_USER TRIGGERED", "LDPCMAIN STATUS",MB_OK);
    }
    break;

```

### Windows NT application module LDPCPROC.C (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page. (For the names of CPRB fields, see page 5).

1. The screen repainting event is handled without any LANDP calls.
2. The case statement identifies a routine to be executed for a program ending event. For the Terminate Program function (TP):
  - Set the request parameter length to 8 bytes. The parameters are the Windows handle and the message identity in mySPparms (*defined on page 130, note 8*), with a combined length of 8 bytes.
  - Set the request data length to 10 bytes, the length of SPdata (*defined on page 130, note 9*).
  - Set the reply parameter length to 0 bytes.
  - Set the reply data length to 0 bytes.
  - Set the function code to TP (*defined on page 130, note 2*).
3. The following five statements:
  - Copy the character array supervisor (*defined on page 130, note 5*), into the server field of the CPRB.
  - Assign the window handle to the WndHandle member of the structure variable mySPparms
  - Assign the message identity to the MessageType member of the structure variable mySPparms
  - Copy mySPparms to the request parameter field addressed by the CPRB.
  - Copy the character array SPdata (*defined on page 130, note 9*) to the request data field addressed by the CPRB.
4. This is the standard LANDP call (*explained in note 5, page 132*).
5. If the TP function call fails, issue a message.
6. This is the standard Windows function to close down message loop and end program LDPCPROC and then LPDCMAIN.
7. This is the standard Windows processing for any event for which there is no case statement.
8. Terminate program.

```

/* Screen needs repainting Event */
1 case WM_PAINT:
    HDC = BeginPaint(hWnd, &ps);
    TextOut(HDC, 10, 10, "Press Any Mouse or Keyboard Key", 31);
    EndPaint(HDC, &ps);
    break;

2 /* Program Ending Event */
case WM_DESTROY:
    mycprb.ehcqparm1 = 8; /* Request parameter length */
    mycprb.ehcqdata1 = 10; /* Request data length */
    mycprb.ehcrparm1 = 0; /* Reply parameter length */
    mycprb.ehcrdata1 = 0; /* Reply data length */
    mycprb.ehcfunct = TP; /* Function ID */

3 memcpy(mycprb.ehcserver, supervisor, 8); /* Destination resource name */
    mySPparms.WndHandle = hWnd;
    mySPparms.MessageType = WM_USER;
    memcpy(mycprb.ehcqparamad, &mySPparms, sizeof(mySPparms));
    memcpy(mycprb.ehcqdataad, SPdata, 10);
4 retcode = RMTREQ (&mycprb, EHC_RESERVED);
5 if (retcode != 0)
    {
        MessageBox(NULL, "LANDP TP TO SPV Failed", "LDPCMAIN Error", MB_OK);
    }
6 PostQuitMessage(0);
    break;

7 /* All other Events */
default:
    lResult = DefWindowProc(hWnd, uMessage, wParam, lParam);
    break;
    }
8 return lResult;
}

```

### Header files

#### SERVICE.H

This header file `server.h` contains the prototypes of two functions:

- `ServiceStart` (*note 4 on page 142*)
- `ServiceStop` (*note 7 on page 146*)

`ServiceStart` and `ServiceStop` are supplied in an object file. “Building the sample server” on page 178 explains how to build this object with this module.

```
extern SECURITY_DESCRIPTOR  Svc_Sd;           // Security Descriptor
extern SECURITY_ATTRIBUTES  Svc_Sa;           // Security Attributes
extern BOOL                 Svc_Service;      // Service indicator
```

```
int      _cdecl ServiceStart(long Argc, char **Argv);
void     _cdecl ServiceStop(void);
```

#### LDPCPROC.H

The header file `ldpcproc.h` contains the prototype of the function `LDPWndProc` (*note 1 on page 132*).

```
LRESULT CALLBACK LDPWndProc(HWND hWnd, UINT uMessage,
                             WPARAM wParam, LPARAM lParam);
```

## Sample user server LDPSMAIN.C

Pages 140 to 147 show a sample NT server program that demonstrates how the LANDP Common Application Programming Interface (CAPI) can be used to develop LANDP server applications. The code of this sample is supplied in directory EHCN500\SAMPLES\SERVER.

To highlight the interface to LANDP, the server is kept deliberately simple. The most complex servers use the same interface. This sample server is used in conjunction with the client program LDCPMAIN.C. (See "Sample client application LDPCMAIN.C" on page 127.)

### LDPSMAIN.C:

1. Issues a LANDP server initialize function (SRVINIT).
2. Loops, issuing a get request (GETREQ), until an Unload LANDP function (ES) is received.
3. In the loop:
  - a. Sets the reply to the IN message to get Timer Ticks from the supervisor every 5 seconds. This allows the server to do periodic processing independently of client messages.
  - b. Processes the KP message from the client (LDPCMAIN). When receiving KP, issues an Asynchronous Event Notification Z5 call to the client. A message box is displayed when KP is received. To see the message box the Service must have the "Allow Service to Interact with Desktop" check box set from services in the control panel.
  - c. The server ends when ES is received.

ES is raised when a user enters ehcfree ldpsmain.exe from the command line.

### Windows NT sample user-written server, LDPSMAIN.C

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page.

1. The first six include files are standard C or Windows NT files, and should be in the library search path.

ehcdefc.h contains the structure of the LANDP connectivity programming request block (CPRB) and is assumed to be in the same directory as the sample application.

service.h is required for service functions, and is assumed to be in the same directory as the sample application.

2. For each LANDP function used, define the function code as the corresponding ASCII hexadecimal value. Only Z5 is explicitly issued by this program. The others are issued by the server recognition process or the client and are received as event codes in the ehcfunct field of this program's CPRB.
3. EHC\_CPRB defines mycprb as the CPRB for this program.
4. Define two character arrays, parm\_area and data\_area, which can be used for sending and receiving parameters and data in client communication.
5. EHC\_CPRB defines myacprb as a second CPRB for this program. This is necessary because communication with the client is not synchronized with the timer-controlled processing.
6. Define two character arrays, aparm\_area and adata\_area, which can be used for sending and receiving parameters and data. This is necessary because communication with the client is not synchronized with the timer-controlled processing.



```

/* Include files                                                    */
1
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <process.h>

#include "ehcdefc.h"

/* Server Functions from service.obj                                */
#include "service.h"

/* Define CPRB function codes (Supervisor and Server requests) and events */
2
#define IN 0x494E
#define ES 0x4553
#define Z5 0x5A35
#define KP 0x4B50
#define TT 0x5454
#define AM 0x2626
#define AS 0x2A2A

/* Declare a CPRB for processing requests                            */
3
EHC_CPRB mycprb;

/* Declare parameter and data areas for use with the CPRB          */
4
char parm_area[26];
char data_area[26];

/* Declare an asynchronous CPRB for processing requests             */
5
EHC_CPRB myacprb;          /* asynchronous request cprb */

/*****
/* Declare parameter and data areas for use with the asynchronous CPRB */
6
/*****
char aparm_area[26];
char adata_area[26];

```

### Windows NT sample user-written server, LDPSMAIN.C (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page.

1. Define the character array `supervisor` to contain the name of the supervisor. The length of the array is 9 to allow for the length of the CPRB field `ehcserver` (8 bytes) plus the end-of-string null byte.
2. Define the character array `Resource0origin` to contain the name of this server. The length of the array is 9 to allow for the length of the CPRB field `ehcserver` (8 bytes) plus the end-of-string null byte.
3. Define the Windows NT Timer Tick `TT` event interval to be 5 seconds.
4. The prototype of the function `ServiceStart` is in the `server.h` header file (*described on page 138*).

The `ServiceStart` function is provided only in the supplied `service.obj` file. You can use this file to create your own servers in the Windows NT environment.

5. Set up five fields required by the `SRVINIT` call. The syntax requires all the fields, even though some are ignored in Windows NT.
    - `retcode` is the return code, which is also used in other calls
    - `size`, which is ignored in Windows NT
    - `init_error`, which contains zero if no errors are found
    - The far address of a routine in a 16-bit environment, which is ignored in Windows NT
    - `optionsptr`, which is the address of additional options
  6. Before initializing this user-written server, initialize `init_error` to zero. Because there are no additional options, set `optionsptr` to `EHC_RESERVED`, as required by `LANDP`. Then issue the `SRVINIT` call with the prepared parameters.
  7. These four lines:
    - Put the address of `parm_area` (*defined on page 140, note 4*) into both the request and reply parameter address areas of the CPRB.
    - Put the address of `data_area` (*defined on page 140, note 4*) into both the request and reply data address areas of the CPRB.
  8. Initialize a loop with the `while` statement. The loop:
    - Continues until an ES (End of Service) message is received.
    - Issues a standard server `GETREQ` call to obtain the next pending request:
      - `&mycprb` (*defined on page 140, note 3*) is the address of the CPRB.
      - There are no additional parameters, therefore the `LANDP` field `EHC_RESERVED` is used instead of the address of `EHC_GETREQ_OPTS`.
    - Uses a switch statement to execute only the code appropriate to the event received.
    - When ES is received, executes no code, and exits on the next iteration.
- The end of the loop is at **6** on page 147. When ES is received, control passes to **7** on page 147.

```

/* Define name of supervisor for all supervisor calls */
1 char supervisor[9] = "SPV    ";

/* Define name of this server */
2 char ResourceOrigin[9] = "LDPSMAIN";

/* Set up the Timer Tick TT event interval */
3 short TT_interval = 5;          /* timer tick interval - 5 seconds */

4
int _cdecl ServiceStart ( long argc, char *argv [])
{
5     unsigned short retcode;          /* return code */
     unsigned short size;             /* ignored in nt and os2 */
     unsigned short init_error;       /* if 0 no errors found */
     void EHC_PTR *routine;           /* ignored in nt and os2 */
     EHC_SRVINIT_OPTSP optionsptr;    /* additional options */

/* Call SRVINIT to initialize the server */
6     init_error = 0;
     optionsptr = EHC_RESERVED;
     retcode=SRVINIT( size, init_error,routine, optionsptr);

     /* Set the CPRB data and parm address to valid addresses */
7     mycprb.ehcqparmad = (char*)parm_area;
     mycprb.ehcrparmad = (char*)parm_area;
     mycprb.ehcqdataad = (char*)data_area;
     mycprb.ehcrdataad = (char*)data_area;

     /* Loop Until End of Service Message is received */
8     while (mycprb.ehcfunct != ES) {
         /******
         /* Call GETREQ to obtain pending requests */
         /******
         retcode = GETREQ(&mycprb, EHC_RESERVED);
         /******

```

### Windows NT sample user-written server, LDPSMAIN.C (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page.

1. The switch statement takes the function code in the incoming CPRB to determine which case routine to execute.
2. In replying to the KP call, the CPRB used is that passed by the client. Set to zero the only three fields that **can** be set by a server when replying to a service request. (See “CPRB fields used and set by servers” on page 48.) These fields are the server return code, the server-supplied replied parameter list length, and the server-supplied replied data length. All variable information is in the CPRB passed as the first parameter. The second parameter must always be EHC\_RESERVED.
3. Start to prepare the asynchronous CPRB (*note 5, page 140*) for asynchronous event notification call (Z5).
  - Initialize all fields in the CPRB to zero.
  - Set the server name length to 8 bytes.
  - Set the server name to the value of supervisor (*note 1, page 142*).
  - Set the function code to Z5 (*defined note 2, page 140*).
4. Set the destination workstation ID (the Z5 call) to the originator workstation ID from the incoming CPRB (the KP call).
5. Set the originator process ID (of this server) using \_getpid.
6. Set the destination process ID (the Z5 call) to the originator process ID from the incoming CPRB (the KP call).
7. Set the originator resource name to ResourceOrigin (*note 2, page 142*).
8. Set the request parameter length to 3 bytes, and other parameter and data lengths to zero.
9. Copy the explicit constant "MY1" into the parameter area.
10. Set the parameter and data area addresses to the addresses of aparm\_area and adata\_area (*note 6, page 140*). The same areas are used for request and reply.
11. Issue a standard asynchronous **client** call (here the server, by sending a call to the supervisor, is acting as a client). The parameters are as for a synchronous call RMTREQ (*note 5, page 142*), except that the second parameter is always EHC\_RESERVED.

```

1      /* Switch on type of request received */
switch (mycprb.ehcfunct) {
case KP:

2          /* Issue a reply to the KP request */

mycprb.ehcservrc = 0;
mycprb.ehcrepldplen = 0;
mycprb.ehcreplddlen = 0;
retcode = RMTRPLY(&mycprb, EHC_RESERVED);

3      /* Issue the Asynchronous Event Notification (Z5 Function call) */
memset(&myacprb,0,sizeof(EHC_CPRB)); /* initialize to 0 */
myacprb.ehcservnamlen=8;
memcpy(myacprb.ehcserver,supervisor,8); /* send to supervisor */
myacprb.ehcfunct = Z5; /* Set function ID to Z5 */

4      /* get client pc id from the KP request */
memcpy(myacprb.ehcddest_pc_id,mycprb.ehcpc_id,2);

5      /* get the process id for this server */
myacprb.ehcpid_origin = _getpid();

6      /* get client process id from the KP request */
myacprb.ehcpid_dest = mycprb.ehcpid_origin;

7

memcpy(myacprb.ehcresource_origin,ResourceOrigin,8);

8

myacprb.ehcqparm1 = 3; /* Request parameter length */
myacprb.ehcqdata1 = 0; /* Request data length */
myacprb.ehcrparm1 = 0; /* Reply parameter length */
myacprb.ehcrdata1 = 0; /* Reply data length */

/* the parm area values are set by agreement with the client */
9 /* except a "0" at offset 2 resets a previous notification */
memcpy(aparm_area,"MY1",3); /* Reason Codes */
/* Set the CPRB data and parm address to valid addresses */

10 myacprb.ehcqparmad = (char*)aparm_area;
myacprb.ehcrparmad = (char*)aparm_area;
myacprb.ehcqdataad = (char*)adata_area;
myacprb.ehcrdataad = (char*)adata_area;

11

retcode = RMTAREQ (&myacprb, EHC_RESERVED);
MessageBox(NULL,"KP Message Received from Client","LDPSMAIN",MB_OK);
break;

```

### Windows NT sample user-written server, LDPSMAIN.C (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page.

1. In replying to the server recognition function (IN), set the CPRB as follows:
  - Set the timer interval `TT_interval` (*note 3, page 142*) to 50ms.
  - Copy the timer interval `TT_interval` to the reply parameter area addressed by the CPRB.
  - Set the server-supplied replied parameter list length to 2 bytes, the length of `TT_interval`.
  - Set the client-supplied reply parameter list length to 2 bytes, the length of `TT_interval`.
  - Set the server return code to zero.
  - Set the server-supplied replied data length to zero.

Issue the standard server reply call.

2. This is a placeholder for possible processing to deal with the timer interval expiry event (TT). As the sample is coded, this event is not processed and receives the default reply.
3. This is a placeholder for possible processing to deal with the workstation or process connection event (AM). As the sample is coded, this event is not processed and receives the default reply.
4. This is a placeholder for possible processing to deal with the workstation or process disconnection event (AS). As the sample is coded, this event is not processed and receives the default reply.
5. `default` identifies a routine in the switch block that is executed for an expression that does not match any of the case expressions. The routine here sets the mandatory server fields to zero, and issues a server reply call.
6. This is the end of the loop initialized at **8** on page 143.
7. This is exit code when ES is received.

*The last two lines of code are never reached in normal processing. They are activated from the GUI panel created by the `service.obj` code.*

8. The prototype of the function `ServiceStop` is in the `server.h` header file (*described on page 138*).
9. This is exit code when service is stopped in the `ServiceStop` module.

**1**

```

case IN: /* server recognition IN function */
/* request timer ticks at 5 second intervals */
TT_interval=(TT_interval*100)/5; /* convert to 50ms intervals */
memcpy(mycprb.ehcrpamad,&TT_interval,2);
mycprb.ehcrepldplen = 2;
mycprb.ehcrparml =2;
mycprb.ehcservrc = 0;
mycprb.ehcreplddlen = 0;
retcode = RMTRPLY(&mycprb, EHC_RESERVED);
break;

```

**2**

```

case TT: /* Timer Tick - do periodic processing here */
/* receives the timer tick at the requested interval */
/* uncomment the next line to see the timer ticks */
/* MessageBox(NULL, "TT Message Received", "LDPSMAIN",MB_OK ); */

```

**3**

```

case AM: /* && - Workstation or process connection */
/* Here the server could add the workstation or process details */
/* to a list. The state of each workstation or process could be */
/* maintained during a complex message flow */

```

**4**

```

case AS: /* ** - Workstation or process disconnection */
/* Here the workstation or process details could be removed from */
/* the list */

```

**5**

```

default:
/*****
/* Issue a reply to all requests eg &&, **, ES etc. */
*****/
mycprb.ehcservrc = 0;
mycprb.ehcrepldplen = 0;
mycprb.ehcreplddlen = 0;
retcode = RMTRPLY(&mycprb, EHC_RESERVED);
break;
} /* endswitch */

```

**6**

```

} /* endwhile */

```

**7**

```

return 0;

```

```

}

```

**8**

```

void _cdecl ServiceStop(void)

```

```

{

```

**9**

```

    ExitProcess(0);

```

```

}

```

**sample user-written server, C, Windows NT**



# Sample application (COBOL, OS/2 AND Windows NT)



The client and server application programs work together. The annotated listings are:

- “Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL”
- “Sample server (COBOL, OS/2 and Windows NT) SAMPSEV.CBL” on page 164

These programs are written in COBOL. Figure 10 shows the calls and functions in the client and server.

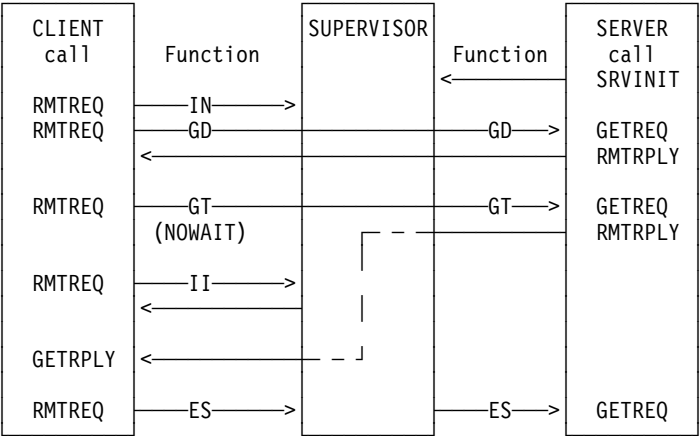


Figure 10. Functions and calls in client and server

## Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL

The code of this sample is supplied in the directories EHCN500\SAMPLES\CLIENT AND EHCO500\SAMPLES\CLIENT. The SAMP-CLI sample client program:

- Issues a LANDP initialize function.
- Issues a 'Get Date' function to the sample server.
- Issues a 'Get Time' function to the sample server using the NoWait option.
- Issues an 'Inquire Information' function to the supervisor.
- Issues a GETRPLY to get the results from the 'Get Time' function.
- Issues an 'End of Service' function to the supervisor.

Messages are displayed on the screen to show the results of the calls. If a call fails with a bad router or server return code, an 'End of Service' function is issued and the program ends.

**Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL  
(continued)**

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page, which shows the Identification, Environment, and Data Divisions of the sample program.

1. These are the mandatory Identification and Environment divisions. The only non-keyword is the program name "SAMP-CLI.CBL". The rest of this page is the Data Division, Working-Storage Section.
2. Define three variables, request and reply parameter areas, and a reply data area, respectively. All are 26 characters long and can hold any valid character.
3. Define two 8-character variables initialized to the names of the supervisor and the sample user-defined server.
4. Copy the LANDP-supplied copybook that includes the CPRB structure.
5. Define an errors flag ERROR-FLAG, and a conditional variable ERRORS-FOUND that is set to *true* if ERROR-FLAG has the value Y.
6. Define variables to hold the required function codes (all byte-reversed).

```

1  IDENTIFICATION DIVISION.
    *=====
    PROGRAM-ID. "SAMP-CLI.CBL".

    ENVIRONMENT DIVISION.
    *=====
    CONFIGURATION SECTION.
    *-----
    INPUT-OUTPUT SECTION.
    *-----

    DATA DIVISION.
    *=====
    WORKING-STORAGE SECTION.

2  * Parameter areas
    01  REQPARMAREA                PIC X(26) VALUE IS SPACES.
    01  REPPARMAREA                PIC X(26) VALUE IS SPACES.
    01  REPDATAAREA               PIC X(26) VALUE IS SPACES.

3  * Supervisor and sample server names
    01  SUPERVISOR                 PIC X(8) VALUE IS "SPV      ".
    01  SAMPLE-SERVER             PIC X(8) VALUE IS "SAMPSEVR".

4  * Copy the LANDP CPRB record structure etc.
    COPY "EHCDEFVA.CBL".

5  01  ERROR-FLAG                PIC X.
    88  ERRORS-FOUND              VALUE "Y".

6  * Supervisor and sample server functions
    * (note reversal of characters)
    01  END-OF-SERVICE-FUNCTION    PIC X(2) VALUE IS "SE".
    01  GET-DATE-FUNCTION          PIC X(2) VALUE IS "DG".
    01  GET-TIME-FUNCTION          PIC X(2) VALUE IS "TG".
    01  INITIALIZE-FUNCTION        PIC X(2) VALUE IS "NI".
    01  INQUIRE-INFORMATION-FUNCTION PIC X(2) VALUE IS "II".

```

## Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL (continued)

The numbers in the following notes refer to the numbers in the listing on the facing page. This is the main routine of the application, which is a series of nested `ifs` with no `else` clauses. The first error causes control to pass to **6**, the end-of-service routine.

Perform each subroutine in turn. Check the return code from each routine with an `if` statement. If there are no errors, perform the next routine. The checked value is the condition name `ERRORS-FOUND` associated with the conditional variable `ERROR-FLAG` (*note 5, page 150*). `ERROR-FLAG` is set in the `CHECK-RETURN-CODES` routine (*note 1, page 154*).

1. Issue an Initialize function call by performing `ISSUE-IN-FUNCTION` (*note 2, page 162*).

Check the result of the call by performing `CHECK-RETURN-CODES` (*note 1, page 154*). If there are no errors, perform `ISSUE-GD-FUNCTION`. If there are errors, exit the nested `ifs`.

2. Issue a Get Date function to the sample server by performing `ISSUE-GD-FUNCTION` (*note 3, page 154*). Check the result of the call by performing `CHECK-RETURN-CODES` (*note 1, page 154*).

If there are no errors:

- a. Display the date. Before issuing a `GET-DATE` function call, `ISSUE-GD-FUNCTION` has set `EHCRRDATAL` to 8 and set the `CPRB` reply data address to the address of `REPDATAAREA` (*notes 3.h and 3.i, page 154*).
- b. Perform `ISSUE-GT-FUNCTION`.

If there are errors, exit the nested `ifs`.

3. Issue a Get Time function call to the sample server by performing `ISSUE-GT-FUNCTION` (*note 2, page 158*). Check the result of the call by performing `CHECK-RETURN-CODES` (*note 1, page 154*). If there are no errors, perform `ISSUE-II-FUNCTION`. If there are errors, exit the nested `ifs`.
4. Issue an Inquire Information function to the supervisor by performing `ISSUE-II-FUNCTION` (*note 1, page 162*). Check the result of the call by performing `CHECK-RETURN-CODES` (*note 1, page 154*). If there are no errors, display the PC ID and perform `ISSUE-GETRPLY`. If there are errors, exit the nested `ifs`.
5. Issue a `GETRPLY` (Get Reply) by performing `ISSUE-GETRPLY` (*note 1, page 158*). This call gets the result of a previous `RMTREQ` (Remote Request) issued by `ISSUE-GT-FUNCTION` with the `NoWait` option. Check the result of the call by performing `CHECK-RETURN-CODES` (*note 1, page 154*). If there are no errors, display the time. Exit the nested `ifs`.
6. Issue an End of Service function by performing `ISSUE-ES-FUNCTION` (*note 2, page 154*). Check the result of the call by performing `CHECK-RETURN-CODES` (*note 1, page 154*).

Terminate the program.

```

        PROCEDURE DIVISION.
        *=====
1      * Issue IN function
          PERFORM ISSUE-IN-FUNCTION

          PERFORM CHECK-RETURN-CODES
          IF NOT ERRORS-FOUND THEN
2      * Get Date
          PERFORM ISSUE-GD-FUNCTION

          PERFORM CHECK-RETURN-CODES
          IF NOT ERRORS-FOUND THEN
            DISPLAY "The date (in YYYYMMDD format) is "
              REPDATAAREA (1:EHCRCRDATA)
3      * Get Time
          PERFORM ISSUE-GT-FUNCTION

          PERFORM CHECK-RETURN-CODES
          IF NOT ERRORS-FOUND THEN
4      * Inquire Information
          PERFORM ISSUE-II-FUNCTION

          PERFORM CHECK-RETURN-CODES
          IF NOT ERRORS-FOUND THEN
            DISPLAY "The PC ID is " REPDATAAREA (1:2)
5      * Get reply
          PERFORM ISSUE-GETRPLY

          PERFORM CHECK-RETURN-CODES
          IF NOT ERRORS-FOUND THEN
            DISPLAY "The time (in HHMMSS format) is "
              REPDATAAREA (1:EHCRCRDATA)
          END-IF
        END-IF
      END-IF
    END-IF
  END-IF

6      * End of Service
          PERFORM ISSUE-ES-FUNCTION

          PERFORM CHECK-RETURN-CODES

          STOP RUN.

```

### Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL (continued)

The numbers or letters in the following notes refer to the highlighted characters in the listing on the facing page.

1. CHECK-RETURN-CODES checks the router and server return codes, using CPRB fields.
  - a. If the router return code is not zero, display the function code being processed and the return code. UERERROK is declared and set to zero in the LANDP-supplied copybook EHCDEFVA.CBL (*note 4, page 150*).
  - b. If the server return code is not zero, display the function code and the return code.
2. ISSUE-ES-FUNCTION issues an ES (End of Service) function to the supervisor. Set CPRB fields as follows:
  - a. Set function to "SE", the byte-reversed value of END-OF-SERVICE-FUNCTION (*note 6, page 150*).
  - b. Set request parameter list length to zero.
  - c. Set request parameter list address to null.
  - d. Set request data length to zero. This causes the unloading of all servers (see description of the ES function in *LANDP Programming Reference*, chapter entitled "Supervisor local functions").
  - e. Set request data address to null.
  - f. Set reply parameter list length to zero.
  - g. Set reply parameter list address to null.
  - h. Set reply data length to zero.
  - i. Set reply data address to null.
  - j. Set server name length to 8 bytes.
  - k. Set server name to "SPV", the value of SUPERVISOR (*note 3, page 150*).
  - l. Issue the standard LANDP request call, passing:

The address in this program's storage of the CPRB. EHC-CPRB is the name of the CPRB structure in the LANDP-supplied copybook EHCDEFVA.CBL (*note 4, page 150*). This enables the supervisor to change the contents of the CPRB, making data available to the application.

The value of the LANDP-supplied keyword EHC-RESERVED.

```

1  * Check the router and server return codes
    CHECK-RETURN-CODES.
    a  IF EHCRETCODE IS NOT EQUAL TO UERERROK THEN
        DISPLAY "Router error issuing function " EHCFUNCT
        DISPLAY "Router return code is " EHCRETCODE-BIN
        MOVE "Y" TO ERROR-FLAG
    ELSE
    b  IF EHCSERVRC IS NOT EQUAL TO UERERROK THEN
        DISPLAY "Server error issuing function " EHCFUNCT
        DISPLAY "Server return code is " EHCSERVRC-BIN
        MOVE "Y" TO ERROR-FLAG
    END-IF
    END-IF.

2  * Issue an ES (End of Service) function to the supervisor
    ISSUE-ES-FUNCTION.
    a  MOVE END-OF-SERVICE-FUNCTION      TO EHCFUNCT
    b  MOVE ZERO                        TO EHCQPARML
    c  SET EHCQPARMAD                    TO NULL
    d  MOVE ZERO                        TO EHCQDATAL
    e  SET EHCQDATAAD                    TO NULL
    f  MOVE ZERO                        TO EHCRPARML
    g  SET EHCRPARMAD                    TO NULL
    h  MOVE ZERO                        TO EHCRDATAL
    i  SET EHCRDATAAD                    TO NULL
    j  MOVE 8                          TO EHCSERVNAMLEN
    k  MOVE SUPERVISOR                  TO EHCSERVER

    1  CALL "RMTREQ" USING BY REFERENCE EHC-CPRB
                                BY VALUE   EHC-RESERVED
    END-CALL.

```

### Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL (continued)

The numbers or letters in the following notes refer to the highlighted characters in the listing on the facing page.

1. ISSUE-GD-FUNCTION issues a GD (Get Date) function to the sample server. Set CPRB fields as follows:
  - a. Set function code to "DG", the value of GET-DATE-FUNCTION (*note 6, page 150*).
  - b. Set request parameter list length to zero.
  - c. Set request parameter list address to null.
  - d. Set request data length to zero.
  - e. Set request data address to null.
  - f. Set reply parameter list length to zero.
  - g. Set reply parameter list address to null.
  - h. Set reply data length to 8 bytes.
  - i. Set reply data address to address of REPDATAAREA (*note 2, page 150*).
  - j. Set server name length to 8 bytes.
  - k. Set server name to "SAMPSEV", the value of SAMPLE-SERVER (*note 3, page 150*).
  - l. Issue the standard LANDP request call, passing:

The address in this program's storage of the CPRB. EHC-CPRB is the name of the CPRB structure in the LANDP-supplied copybook EHCDEFVA.CBL (*note 4, page 150*). This enables the supervisor to change the contents of the CPRB, making data available to the application.

The value of the LANDP-supplied keyword EHC-RESERVED.



```

* Issue a GD (Get Date) function to the sample server
1  ISSUE-GD-FUNCTION.
  a  MOVE GET-DATE-FUNCTION          TO EHCFUNCT
  b  MOVE ZERO                      TO EHCQPARML
  c  SET EHCQPARMAD                  TO NULL
  d  MOVE ZERO                      TO EHCQDATAL
  e  SET EHCQDATAAD                  TO NULL
  f  MOVE ZERO                      TO EHCRPARML
  g  SET EHCRPARMAD                  TO NULL
  h  MOVE 8                         TO EHCRDATAL
  i  SET EHCRDATAAD                  TO ADDRESS OF REPDATAAREA
  j  MOVE 8                         TO EHCSERVNAMLEN
  k  MOVE SAMPLE-SERVER              TO EHCSERVER

  l  CALL "RMTREQ" USING BY REFERENCE EHC-CPRB
                                BY VALUE  EHC-RESERVED
END-CALL.

```

### Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL (continued)

The numbers or letters of the following notes refer to the highlighted characters in the listing on the facing page.

1. ISSUE-GETRPLY issues a GETRPLY (Get Reply) request to get the results of a previous RMTREQ (Remote Request) with the NoWait option, issued by ISSUE-GT-FUNCTION. Set CPRB fields as follows:
  - a. Set timeout per request to zero.
  - b. Set reply parameter list length to zero.
  - c. Set reply parameter list address to null.
  - d. Set reply data length to 6 bytes.
  - e. Set reply data address to address of REPDATAAREA (*note 2, page 150*).
  - f. Set the NOWAIT-PARMAD (No Wait Parameter Address) field of EHC-GETRPLY-OPTS (GETRPLY Options Control Block) to the address of EHC-NOWAIT-PARM, which contains the RMTREQ no wait parameter structure. For the layout of EHC-GETRPLY\_OPTS and EHC-NOWAIT-PARM, see page 32. The structures of EHC-GETRPLY\_OPTS and EHC-NOWAIT-PARM are included in EHCDEFVA.CBL, which is copied into working-storage (*note 4, page 150*).
  - g. Issue the standard LANDP request call, passing:

The address in this program's storage of the CPRB. EHC-CPRB is the name of the CPRB structure in the LANDP-supplied copybook EHCDEFVA.CBL (*note 4, page 150*). This enables the supervisor to change the contents of the CPRB, making data available to the application.

The address of EHC-GETRPLY\_OPTS, also included in EHCDEFVA.CBL.

```

* Issue a GETRPLY (Get Reply) to get the results of a previous
* RMTREQ (Remote Request) issued with the NoWait option.
1  ISSUE-GETRPLY.
    a  MOVE ZERO                                TO EHCTIMEOUT
    b  MOVE ZERO                                TO EHCRPARML
    c  SET EHCRPARMAD                           TO NULL
    d  MOVE 6                                    TO EHCRDATAL
    e  SET EHCRDATAAD                           TO ADDRESS OF REPDATAAREA

    f  SET NOWAIT-PARMAD OF EHC-GETRPLY_OPTS
        TO ADDRESS OF
        EHC-NOWAIT-PARM

    g  CALL "GETRPLY" USING BY REFERENCE EHC-CPRB
        BY REFERENCE EHC-GETRPLY-OPTS
    END-CALL.

```

### Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL (continued)

The numbers or letters of the following notes refer to the highlighted characters in the listing on the facing page.

1. ISSUE-GT-FUNCTION issues a GT (Get Time) function to the sample server. Set CPRB fields as follows:
  - a. Set function code to "TG", the value of GET-TIME-FUNCTION (*note 6, page 150*).
  - b. Set request parameter list length to zero.
  - c. Set request parameter list address to null.
  - d. Set request data length to zero.
  - e. Set request data address to null.
  - f. Set reply parameter list length to zero.
  - g. Set reply parameter list address to null.
  - h. Set reply data length to 6.
  - i. Set reply data address to address of REPDATAAREA (*note 2, page 150*).
  - j. Set server name length to 8 bytes.
  - k. Set server name to "SAMPSEV", the value of SAMPLE-SERVER (*note 3, page 150*).
  - l. Set the NOWAIT-PARMAD (No Wait Parameter Address) field of EHC-RMTREQ-OPTS (RMTREQ Options Control Block) to the address of EHC-NOWAIT-PARM, which contains the RMTREQ no wait parameter structure. For the layout of EHC-RMTREQ-OPTS and EHC-NOWAIT-PARM, see pages 31 and 32. The structures of EHC-RMTREQ-OPTS and EHC-NOWAIT-PARM are included in EHCDEFVA.CBL, which is copied into working-storage (*note 4, page 150*).
  - m. Issue the standard LANDP request call, passing:

The address in this program's storage of the CPRB. EHC-CPRB is the name of the CPRB structure in the LANDP-supplied copybook EHCDEFVA.CBL (*note 4, page 150*). This enables the supervisor to change the contents of the CPRB, making data available to the application.

The address of EHC-RMTREQ-OPTS, also included in EHCDEFVA.CBL.

\* Issue a GT (Get Time) function to the sample server using the  
 \* NoWait option of the RMTREQ (Remote Request) call

```

1  ISSUE-GT-FUNCTION.
    a  MOVE GET-TIME-FUNCTION          TO EHCFUNCT
    b  MOVE ZERO                      TO EHCQPARML
    c  SET EHCQPARMAD                 TO NULL
    d  MOVE ZERO                      TO EHCQDATAL
    e  SET EHCQDATAAD                 TO NULL
    f  MOVE ZERO                      TO EHCRPARML
    g  SET EHCRPARMAD                 TO NULL
    h  MOVE 6                         TO EHCRDATAL
    i  SET EHCRDATAAD                 TO ADDRESS OF REPDATAAREA
    j  MOVE 8                         TO EHCSERVNAMLEN
    k  MOVE SAMPLE-SERVER             TO EHCSERVER

    l  SET NOWAIT-PARMAD OF EHC-RMTREQ-OPTS
                                           TO ADDRESS OF
                                           EHC-NOWAIT-PARM

    m  CALL "RMTREQ" USING BY REFERENCE EHC-CPRB
                                           BY REFERENCE EHC-RMTREQ-OPTS
    END-CALL.
  
```

## Sample client (COBOL, OS/2 and Windows NT) SAMP-CLI.CBL (continued)

The numbers or letters of the following notes refer to the highlighted characters in the listing on the facing page.

1. ISSUE-II-FUNCTION issues an Inquire Information function to the supervisor. Set CPRB fields as follows:
  - a. Set function code to "II", the value of INQUIRE-INFORMATION-FUNCTION (*note 6, page 150*).
  - b. Move "L" to the request parameter list area. This parameter value requests the return of the PC ID. (See the table "Reply DATA values when 'L' entered in Request PARMLIST" in the "Inquire information (II function)" section in the "Supervisor local functions" chapter of the *LANDP Programming Reference*.)
  - c. Set request parameter list length to 1 byte.
  - d. Set request parameter list address to the address of REQPARMAREA (*note 2, page 150*).
  - e. Set request data length to 0 bytes.
  - f. Set request data address to null.
  - g. Set reply parameter list length to zero.
  - h. Set reply parameter list address to null.
  - i. Set reply data length to 10 bytes.
  - j. Set reply data address to address of REPDATAAREA (*note 2, page 150*).
  - k. Set server name length to 8 bytes.
  - l. Set server name to the value of SUPERVISOR (*note 3, page 150*).
- m. Issue the standard LANDP request call, passing:
  - The address in this program's storage of the CPRB. EHC-CPRB is the name of the CPRB structure in the LANDP-supplied copybook EHCDEFVA.CBL (*note 4, page 150*). This enables the supervisor to change the contents of the CPRB, making data available to the application.
  - The value of EHC-RESERVED, also included in EHCDEFVA.CBL.
2. ISSUE-IN-FUNCTION issues an Initialize function to the supervisor. Set CPRB fields as follows:
  - a. Set function code to "NI", the value of INITIALIZE-FUNCTION (*note 6, page 150*).
  - b. Set request parameter list length to 0 bytes.
  - c. Set request parameter list address to null.
  - d. Set request data length to 0 bytes.
  - e. Set request data address to null.
  - f. Set reply parameter list length to 6 bytes.
  - g. Set reply parameter list address to address of REPPARMAREA (*note 2, page 150*).
  - h. Set reply data length to 0 bytes.
  - i. Set reply data address to null.
  - j. Set server name length to 8 bytes.
  - k. Set server name to the value of SUPERVISOR (*note 3, page 150*).
  - l. Issue the standard LANDP request call as in **1m**.

\* Issue an II (Inquire Information) function to the supervisor  
 \* (Put an "L" in the request PARMLIST to obtain the PC ID)

```

1  ISSUE-II-FUNCTION.
   a  MOVE INQUIRE-INFORMATION-FUNCTION TO EHCFUNCT
   b  MOVE "L"                           TO REQPARMAREA
   c  MOVE 1                             TO EHCQPARML
   d  SET EHCQPARMAD                      TO ADDRESS OF REQPARMAREA
   e  MOVE ZERO                          TO EHCQDATAL
   f  SET EHCQDATAAD                     TO NULL
   g  MOVE ZERO                          TO EHCRPARML
   h  SET EHCRPARMAD                     TO NULL
   i  MOVE 10                            TO EHCRDATAL
   j  SET EHCRDATAAD                    TO ADDRESS OF REPDATAAREA
   k  MOVE 8                             TO EHCSERVNAMLEN
   l  MOVE SUPERVISOR                   TO EHCSERVER

   m  CALL "RMTREQ" USING BY REFERENCE EHC-CPRB
                                BY VALUE   EHC-RESERVED

      END-CALL.
    
```

\* Issue an IN (Initialize) function to the supervisor  
 ISSUE-IN-FUNCTION.

```

2  MOVE INITIALIZE-FUNCTION TO EHCFUNCT
   a  MOVE ZERO             TO EHCQPARML
   b  SET EHCQPARMAD        TO NULL
   c  MOVE ZERO            TO EHCQDATAL
   d  SET EHCQDATAAD       TO NULL
   e  MOVE 6               TO EHCRPARML
   f  SET EHCRPARMAD       TO ADDRESS OF REPPARMAREA
   g  MOVE ZERO            TO EHCRDATAL
   h  SET EHCRDATAAD      TO NULL
   i  MOVE 8               TO EHCSERVNAMLEN
   j  MOVE SUPERVISOR     TO EHCSERVER
   k

   l  CALL "RMTREQ" USING BY REFERENCE EHC-CPRB
                                BY VALUE   EHC-RESERVED

      END-CALL.
    
```

END PROGRAM "SAMP-CLI.CBL".

## Sample server (COBOL, OS/2 and Windows NT) SAMPSEV.CBL

This sample program shows how the LANDP Common Application Programming Interface (CAPI) can be used to develop LANDP server applications. This sample implements a very simple server but demonstrates the same principles that are used by functionally richer server applications. The code of this sample is supplied in the directories EHCN500\SAMPLES\SERVER AND EHCO500\SAMPLES\SERVER.

The program:

- Calls SRVINIT to register itself with LANDP.
- Repeats the following steps until the ES (End of Service) function is received:
  - Calls GETREQ to receive a request.
  - Performs various actions depending on the type of request.
    - GD (Get Date): checks parameters and returns the date.
    - GT (Get Time): checks parameters and returns the time.
    - Calls RMTRPLY to reply to the request.

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page. This listing shows the Identification, Environment, and Data Divisions of the sample program.

1. These are the mandatory Identification and Environment divisions. The only non-keyword is the program name "SAMPSEV.CBL". The rest of this page is the Data Division, Working-Storage Section and Linkage Section.
2. Copy the LANDP-supplied copybook that includes the CPRB structure.
3. Set up two parameters for the SRVINIT function. These represent the **size** and **process\_request** parameters that are required by LANDP for DOS, but ignored by LANDP for OS/2 and Windows NT. They are included for compatibility purposes.
4. Set up the `init_error` field used in the SRVINIT call (*note 2 page 166*) as the parameter that records an error detected during the initialization phase of the server.
5. Define variables to hold the required function codes (byte-reversed because the sample was tested on an Intel machine).
6. Set up a variable to hold the returned error code if a client call passes an incorrect reply data length in the CPRB.
7. Set variables to hold the correct returned data lengths for GET-DATE and GET-TIME.
8. Set up linkage-section definitions. These are applied to storage addresses passed by the client in the reply data address field of the CPRB (*note 3.i, page 154*).



```

1  IDENTIFICATION DIVISION.
    *=====
    PROGRAM-ID. "SAMPSERV.CBL".

    ENVIRONMENT DIVISION.
    *=====
    CONFIGURATION SECTION.
    *-----
    INPUT-OUTPUT SECTION.
    *-----

    DATA DIVISION.
    *=====
    WORKING-STORAGE SECTION.
    *-----

2  * Copy the LANDP CPRB record structure etc.
    COPY "EHCDEFVA.CBL".

3  * SRVINIT dummy parameters
    01  FOUR-BYTES                      PIC 9(8)
                                         USAGE IS COMP-5 VALUE IS ZERO.
    01  TWO-BYTES                      PIC 9(4)
                                         USAGE IS COMP-5 VALUE IS ZERO.

4  * SRVINIT parameter
    01  INIT-ERROR                      PIC 9(4) USAGE IS COMP-5.

5  * Functions handled by the server (note reversal of characters)
    01  END-OF-SERVICE                  PIC X(2) VALUE IS "SE".
    01  GET-DATE                        PIC X(2) VALUE IS "DG".
    01  GET-TIME                        PIC X(2) VALUE IS "TG".

6  * Error code for GET-DATE and GET-TIME calls (note byte reversal)
    01  BAD-REPLY-DATA-LENGTH          PIC X(4) VALUE IS X"5A5A0001".

7  * Lengths of returned data
    01  DATE-LENGTH                    PIC 9(8) USAGE IS COMP-5.
    01  TIME-LENGTH                    PIC 9(8) USAGE IS COMP-5.

    LINKAGE SECTION.
    *-----

8  * Storage (supplied by the client) for the results of
    * GET-DATE and GET-TIME
    01  DATE-YYYYMMDD                  PIC X(8).
    01  TIME-HHMMSS                     PIC X(6).

```

## Sample server (COBOL, OS/2 and Windows NT) SAMPSEV.CBL (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page, which is the main routine of SAMPSEV.

1. Set up the lengths of the returned data.

Set DATE-LENGTH (*note 7, page 164*) to length of DATE-YYYYMMDD (*note 8, page 164*).

Set TIME-LENGTH to length (*note 7, page 164*) TIME-HHMMSS (*note 8, page 164*).

2. Issue a SRVINIT call to register this program as a server with LANDP.

The first and third parameters (*note 3, page 164*) are ignored in LANDP for OS/2 and Windows NT, but are included for compatibility with LANDP for DOS

The second parameter (*note 4, page 164*) is required to hold an error code if the call fails.

EHC-RESERVED (defined in EHCDEFVA.CBL, *note 2, page 164*) is a required parameter.

3. Initialize a loop to iterate until the function code in the passed CPRB is "SE", the byte-reversed value of END-OF-SERVICE (*note 5, page 164*). The loop ends at **11**.
4. Issue a Get Request call to obtain the next pending request. EHC-CPRB (defined in EHCDEFVA.CBL, *note 2, page 164*) is the CPRB structure applied to the first parameter of the incoming request. If the call used additional options, EHC-RESERVED would be replaced by EHC-GETREQ-OPTS (also defined in EHCDEFVA.CBL, *note 2, page 164*).
5. Initialize EVALUATE statement, which ends at **9**. When a WHEN phrase is true, the associated statement is executed.
6. When the function code in the incoming request is GET-DATE (*note 5, page 164*), perform PROCESS-GET-DATE (*note 1, page 168*).
7. When the function code in the incoming request is GET-TIME (*note 5, page 164*), perform PROCESS-GET-TIME (*note 2, page 168*).
8. When the function code in the incoming request is neither GET-DATE nor GET-TIME, in the client's CPRB:
  - Set the server-provided parameter list length to zero.
  - Set the server-provided data length to zero.
  - Set the server return code to zero, using UERERROK, defined and initialized in EHCDEFVA.CBL (*note 2, page 164*).This action is taken for any value (including "SE") of EHC-FUNCT other than "DG" or "TG".
9. End EVALUATE statement.
10. Issue a Remote Reply call returning exactly the same parameters as received by the Get Request call, the address of the CPRB and the value of EHC-RESERVED. The contents of the CPRB have been changed.
11. End PERFORM statement.
12. Terminate program.

```

PROCEDURE DIVISION.
*=====
1  * Set up lengths of returned data
    MOVE LENGTH OF DATE-YYYYMMDD TO DATE-LENGTH.
    MOVE LENGTH OF TIME-HHMMSS   TO TIME-LENGTH.

2  * Register the server with LANDP (note that the first and third
    * parameters are ignored under LANDP for OS/2 and Windows NT)
    CALL "SRVINIT" USING BY VALUE TWO-BYTES
                        BY VALUE INIT-ERROR
                        BY VALUE FOUR-BYTES
                        BY VALUE EHC-RESERVED

    END-CALL.

3  * Loop until ES (End of Service) function is received
    PERFORM WITH TEST AFTER
        UNTIL EHCFUNCT IS EQUAL TO END-OF-SERVICE

4  * Get a request
    CALL "GETREQ" USING BY REFERENCE EHC-CPRB
                      BY VALUE      EHC-RESERVED

    END-CALL

5  EVALUATE TRUE
6  * GD (Get Date) received
    WHEN EHCFUNCT IS EQUAL TO GET-DATE
        PERFORM PROCESS-GET-DATE

7  * GT (Get Time) received
    WHEN EHCFUNCT IS EQUAL TO GET-TIME
        PERFORM PROCESS-GET-TIME

8  * Other function received
    WHEN OTHER
        MOVE ZERO      TO EHCREPLDPLEN
        MOVE ZERO      TO EHCREPLDDLEN
        MOVE UERERROK TO EHCSEVRVC

9  END-EVALUATE

10 * Reply to a request
    CALL "RMTRPLY" USING BY REFERENCE EHC-CPRB
                       BY VALUE      EHC-RESERVED

    END-CALL

11 END-PERFORM.

12 STOP RUN.

```

## Sample server (COBOL, OS/2 and Windows NT) SAMPSEV.CBL (continued)

The numbers or letters of the following notes refer to the highlighted characters in the listing on the facing page.

1. PROCESS-GET-DATE sets fields in the client CPRB for an RMTRPLY call to be issued by the main routine. The call is in reply to a GET-DATE function call from the client.
  - a. If the reply data length in the CPRB is equal to the date field length, DATE-LENGTH (*note 8, page 164*), set CPRB as follows, lines **1)** - **5)** :
    - 1) Set DATE-YYYYMMDD (*note 8, page 164 and note 1, page 166*) to apply to the area pointed to by the reply data address.
    - 2) Set DATE-YYYYMMDD (that is, the area pointed to by reply data address) to the value returned by the function CURRENT-DATE. DATE-LENGTH gives the number of characters, starting at position 1.
    - 3) Set server-provided replied parameter list length field to zero.
    - 4) Set server-provided replied data length field to the value of DATE-LENGTH.
    - 5) Set the server return code to zero, using UERERROK (defined and initialized in EHCDEFVA.CBL, *note 2, page 164*).
  - b. If the reply data length in the CPRB is not equal to DATE-LENGTH, set CPRB as follows, lines **1)** - **3)** :
    - 1) Set server-provided replied parameter list length field to zero.
    - 2) Set server-provided replied data length field to zero.
    - 3) Set the server return code to BAD-REPLY-DATA-LENGTH (*note 6, page 164*), indicating that the client passed an incorrect reply data length.
  - c. End of if statement at **1a**.
2. PROCESS-GET-TIME sets fields in the client CPRB for an RMTRPLY call to be issued by the main routine. The call is in reply to a GET-TIME function call from the client.
  - a. If the reply data length in the CPRB is equal to the time field length, TIME-LENGTH (*note 8, page 164*), set CPRB as follows, lines **1)** - **5)** :
    - 1) Set TIME-HHMMSS (*note 8, page 164 and note 1, page 166*) to apply to the area pointed to by the reply data address.
    - 2) Set TIME-HHMMSS (that is, the area pointed to by reply data address) to the value returned by the function CURRENT-DATE (TIME-LENGTH gives the number of characters, starting at position 9).
    - 3) Set server-provided replied parameter list length field to zero.
    - 4) Set server-provided replied data length field to the value of TIME-LENGTH.
    - 5) Set the server return code to zero, using UERERROK (defined and initialized in EHCDEFVA.CBL, (*note 2, page 164*).
  - b. If the reply data length in the CPRB is not equal to TIME-LENGTH, set CPRB as follows, lines **1)** - **3)** :
    - 1) Set server-provided replied parameter list length field to zero.
    - 2) Set server-provided replied data length field to zero.
    - 3) Set the server return code to BAD-REPLY-DATA-LENGTH (*note 6, page 164*), indicating that the client passed an incorrect reply data length.
  - c. End of if statement at **2a**.

3. This line is documentation.

```
* If the reply data length is correct copy the data into the
* client's storage and set up the replied parameter and data
* lengths and a good server return code. If the reply data length
* is incorrect set up the replied parameter and data lengths and a
* bad server return code.
```

**1** PROCESS-GET-DATE.

**a** IF EHCRDATAL IS EQUAL TO DATE-LENGTH

**1)** SET ADDRESS OF DATE-YYYYMMDD TO EHCRDATAAD

```

2) MOVE FUNCTION CURRENT-DATE (1:DATE-LENGTH)

```

TO DATE-YYYYMMDD

```
3) MOVE ZERO TO EHCREPLDPLEN
```

```

3)  MOVE EERS                                TO EHCREFDPLEN
4)  MOVE DATE-LENGTH                          TO EHCREFLDDLEN

```

```
5) MOVE UERERROK TO EHCSERVRC
```

**b** ELSE

1) MOVE ZERO TO EHCREPLDPLEN

```
2) MOVE ZERO TO EHCREPLDDLEN
```

```
3) MOVE BAD-REPLY-DATA-LENGTH TO EHCSERVRC
```

**C**      END-IF.

```
* If the reply data length is correct copy the time into the
* client's storage and set up the replied parameter and data
* lengths and a good server return code. If the reply data length
* is incorrect set up the replied parameter and data lengths and a
* bad server return code.
```

## 2 PROCESS-GET-TIME.

**a** IF EHCRDATAL IS EQUAL TO TIME-LENGTH

```
1) SET ADDRESS OF TIME-HHMMSS TO EHCRDATAAD
```

```

2) MOVE FUNCTION CURRENT-DATE (9:TIME-LENGTH)

```

TO TIME-HHMMSS

3) MOVE ZERO TO FIVE THIRDS TO EHCPLDPLEN

```

3)  MOVE ZERO          TO ENCRYPTED_LEN
4)  MOVE TIME-LENGTH   TO FHCRCPLDLEN

```

```

5) MOVE UERBRBOK TO EHCSERVBC

```

**b** ELSE

```
1) MOVE ZERO TO EHCREPLDPLEN
```

```
2) MOVE ZERO TO EHCRCPLDDLEN
```

```
3) MOVE BAD-REPLY-DATA-LENGTH TO EHCSERVRC
```

**C** END-IF.

```
3 END PROGRAM "SAMPSEV.CBL".
```

---

### Sample client application (COBOL), DOS and OS/2



The sample program SAMPLECB.CBL shows how the LANDP Common Application Programming Interface (CAPI) can be used to develop LANDP client applications. This sample is portable between LANDP for DOS and LANDP for OS/2. To highlight the interface to LANDP, the application is kept deliberately simple. The most complex applications use the same interface. The code of this sample is supplied in the directories EHCD500\SAMPLES\CLIENT AND EHCO500\SAMPLES\CLIENT.

**Note:** Because the sample was tested on an Intel machine, function codes and event codes are byte-reversed. See “Function, return, and event codes” on page xv.

This application:

1. Issue a LANDP initialize function
2. Repeat the following steps:
  - a. Issue **Start Timer** function with a one second interval.
  - b. Issue **Wait Multiple** function to wait for any asynchronous event (one of them is **Timer expired**).
  - c. Determine why the Wait Multiple returned.
  - d. If the reason was **Timer expired**, display the timeout counter and continue.
  - e. If the reason was **Keyboard pressed**, display the reason and terminate.
  - f. If the reason was any other, display a suitable message and continue.

### Sample client application program SAMPLECB.CBL

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page.

1. These lines are the mandatory identification division and environment division. The only non-keywords are `_MAIN`, the name that identifies the object program to the system, and `ibm-personal-computer`, which is documentation.
2. Define data area for setting up the Timer function call. The minutes and seconds fields each contain three digits initialized to zeros.
3. Define request and reply parameter fields for function calls. Each field contains 26 characters initialized to blanks. The fields can include any valid character.
4. Define the variable `SUPERVISOR` as 8 characters initialized to `SPV` with five trailing blanks. This variable is used to set the CPRB server field.
5. Copy the supplied file `EHCD500\SAMPLES\CLIENT` into working storage. This file defines the CPRB (connectivity programming request block, see page 5).
6. Define the LANDP function and event codes required by the program. The codes are held in byte-reversed mode in two-character fields. In the order defined, the functions are Activate Timer, Wait for Asynchronous Events, Keyboard Event, and Initialize.
7. Define a field to contain the return code from the server (to an activate timer call) if the timer is already set. This field is used at **3**, page 175.

- 1**    identification division.  
      program-id. "\_MAIN".  
      environment division.  
      configuration section.  
      source-computer. ibm-personal-computer.  
      input-output section.  
      data division.  
      working-storage section.
- 2**    \* Definition of the Dataarea for the Supervisor  
      01 DATAAREA.  
          02 MINUTES PIC 9(3) USAGE IS COMP VALUE IS 0.  
          02 SECONDS PIC 9(3) USAGE IS COMP VALUE IS 0.
- 3**    \* Parameter area  
      01 REQPARMAREA PIC X(26) VALUE IS SPACES.  
      01 REPPARMAREA PIC X(26) VALUE IS SPACES.
- 4**    \* The name of the Supervisor is SPV  
      01 SUPERVISOR PIC X(8) VALUE IS 'SPV     '.
- 5**    \* The CPRB definition  
      COPY EHCDEFEB.CBL.
- 6**    \* Some Supervisor functions  
      77 T1 PIC X(2) VALUE IS '1T'.  
      77 WM PIC X(2) VALUE IS 'MW'.  
      77 KB PIC X(2) VALUE IS 'BK'.  
      77 INF PIC X(2) VALUE IS 'NI'.
- 7**    \* The status returned when the Timer is already set  
      77 P3 PIC X(4) VALUE IS X"33500001".

### Sample client application program SAMPLECB.CBL (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page.

1. Set CPRB fields for the IN call as follows:
  - Set server field to the required value SPV using the variable SUPERVISOR (defined in note 4, page 170).
  - Set function code to the required value NI using the variable INF (defined in note 6, page 170).
  - Set request data length to zero.
  - Set request parameter length to zero.
  - Set reply parameter length to zero.
  - Issue the standard LANDP request call, passing:
    - The value of the LANDP-supplied keyword EHC-RESERVED
    - The address in this program's storage of the CPRB (EHC-CPRB is the name of the CPRB structure in the supplied copybook EHCDEFEB.CBL (note 5, page 170)). This enables the supervisor to change the contents of the CPRB, in this example by inserting the return codes.
2. If the return code from the IN call is zero (successful completion), skip the nested if then else clause and go to **5** to start the application loop.
3. If the router return code is not zero, display a message and terminate program.
4. If the server return code is not zero, display a message and terminate program.
5. The label forever identifies the start of a loop. Control returns here when a timer event or unrecognizable asynchronous event is detected (notes 5 and 6, page 176).
6. Set CPRB fields for the Activate Timer (T1) call as follows:
  - Set server field to the required value SPV using the variable SUPERVISOR (defined in note 4, page 170).
  - Set function code to the required value 1T using the variable T1 (defined in note 6, page 170).
  - Set MINUTES to zero. Set SECONDS to 1. These are the minutes and seconds parts of the timer interval held in the non-CPRB field DATAREA (defined in note 2, page 170). DATAREA defines the format of the request data area required by an Activate Timer function call. For a description of the activator and deactivator timer functions, see *LANDP Programming Reference*, chapter entitled "Supervisor local functions". Here, the requested timer interval is one second.
  - Set the CPRB request data length to 4 bytes, the length of DATAREA.
  - Set the CPRB request data address to the address of DATAREA.
  - Set request parameter length to 26 bytes.
  - Set reply parameter length to 26 bytes.
  - Set request parameter address to the address of REQPARAMAREA (note 3, page 170).
  - Set reply parameter address to the address of REPPARAMAREA (note 3, page 170).
7. Issue the standard LANDP request call, passing:
  - The value of the LANDP-supplied keyword EHC-RESERVED
  - The address in this program's storage of the CPRB (EHC-CPRB is the name of the CPRB structure in the LANDP-supplied copybook EHCDEFEB.CBL



(note 5, page 170)). This enables the supervisor to change the contents of the CPRB, in this example by inserting the return codes.

---

```

procedure division.
1  * Issue function IN.
    move SUPERVISOR to EHCSERVER.
    move INF to EHCFUNCT.
    move 0 to EHCQDATAL.
    move 0 to EHCQPAPML.
    move 0 to EHCRPAPML.
    call "__RMTREQ" using
        by value EHC-RESERVED
        by reference EHC-CPRB
    end-call.
2  * Parse return codes from the Supervisor
3  if (return-code not = 0) then
4      if (EHCRETCODE not = UERERROK) then
        display 'Router error issuing function IN'
        stop run
      else
        if (EHCSERVRC not = UERERROK) then
            display 'Server error issuing function IN'
            stop run
        end-if
      end-if.
5  * Beginning of the eternal loop
    forever.
6  * Issue Activate Timer function (T1) to the Supervisor
    move SUPERVISOR to EHCSERVER.
    move T1 to EHCFUNCT.
    move 0 to MINUTES.
    move 1 to SECONDS.
    move 4 to EHCQDATAL.
    set EHCQDATAAD to address of DATAREA.
    move 26 to EHCQPAPML.
    move 26 to EHCRPAPML.
    set EHCQPAPMAD to address of REQPARMAREA.
    set EHCRPAPMAD to address of REPPARMAREA.
7  call "__RMTREQ" using
        by value EHC-RESERVED
        by reference EHC-CPRB
    end-call.

```

### Sample client application program SAMPLECB.CBL (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page.

1. If the return code from the T1 call is zero (successful completion), skip the nested `if then else` clause and go to **4** to issue a Wait Multiple function call to wait for an asynchronous event.
2. If the router return code is not zero, display an appropriate message and terminate the program.
3. If the server return code is not zero and the timer was not already set, display an appropriate message and terminate the program. The variable P3 (*defined in note 7, page 170*) contains the value returned by the supervisor if the timer is already set.
4. Set CPRB fields for the WM call as follows:
  - Set server field to the required value SPV using the variable SUPERVISOR (*defined in note 4, page 170*).
  - Set function code to the required value MW using WM (*defined in note 6, page 170*).
  - Set request data length to zero.
  - Set request data address to the address of DATAREA (*note 2, page 170*).
  - Set request parameter length to 26 bytes.
  - Set reply parameter length to 26 bytes.
  - Set request parameter address to the address of REQPARMAREA (*note 3, page 170*).
  - Set reply parameter address to the address of REPPARMAREA (*note 3, page 170*).

Issue the standard LANDP request call, passing:

- The value of the LANDP-supplied keyword EHC-RESERVED
- The address in this program's storage of the CPRB (EHC-CPRB is the name of the CPRB structure in the LANDP-supplied copybook EHCDEFEB.CBL (*note 5, page 170*)). This enables the supervisor to change the contents of the CPRB, in this example by inserting the return codes.

```

* Parse return codes from the Supervisor
1   if (return-code not = 0) then
2       if (EHCRETCODE not = UERERROK) then
           display 'Router error issuing function T1'
           stop run
       else
3           if (EHCSERVRC not = UERERROK) and
               (EHCSERVRC not = P3) then
               display 'Server error issuing function T1'
               stop run
           end-if
       end-if
   end-if.

* Now, for the Wait Multiple function to wait for asynchronous event

4   * Issue WM function
       move SUPERVISOR to EHCSERVER.
       move WM to EHCFUNCT.
       move 0 to EHCQDATAL.
       set EHCQDATAAD to address of DATAREA.
       move 26 to EHCQPARML.
       move 26 to EHCRPARML.
       set EHCQPARMAD to address of REQPARMAREA.
       set EHCRPARMAD to address of REPPARMAREA.

       call "__RMTREQ" using
           by value EHC-RESERVED
           by reference EHC-CPRB

       end-call.

```

### Sample client application program SAMPLECB.CBL (continued)

The numbers of the following notes refer to the highlighted numbers in the listing on the facing page.

1. If the return code from the WM call is zero (successful completion), skip the nested if then else clause and go to **4** to check the parameter returned by the call.
2. If the router return code is not zero, display an appropriate message and terminate the program.
3. If the server return code is not zero, display an appropriate message and terminate the program.
4. If the reply parameter list contains 'BKSPV', the user has pressed a keyboard key, which is interpreted as a request for program termination. Display an appropriate message and terminate the program.

In the reply parameter list, 'BKSPV', the first two bytes are KB (byte-reversed) indicating a keyboard event. The last three bytes are SPV, the name of the server, which in this case is the supervisor. See "Function, return, and event codes" on page xv.

5. If the reply parameter list contains '1TSPV', the timeout specified in the timer call has expired. Display an asterisk and return to the start of the loop (*note 5, page 172*).

In the reply parameter list contains '1TSPV', the first two bytes are T1 (byte-reversed) representing a timer event. 1T explicitly identifies the timer call that requested the timeout that has expired (*note 6, page 172*). The last three bytes are SPV, the name of the server, which in this case is the supervisor. See "Function, return, and event codes" on page xv.

6. Control reaches here only if:

The supervisor return code is zero

and

The reply PARMLIST is not 'BKSPV' or '1TSPV'.

Display an appropriate message and the reply parameter list. Return to the start of the loop (*note 5, page 172*).

This program loops until one of the following occurs:

- The user presses a keyboard key.
- The router returns a non-zero return code.
- The server returns a non-zero return code.

```

* Parse return codes from the Supervisor
1   if (return-code not = 0) then
2       if (EHCRETCODE not = UERERROK) then
           display 'Router error issuing function WM'
           stop run
       else
3           if (EHCSERVRC not = UERERROK) then
               display 'Server error issuing function WM'
               stop run
           end-if
       end-if
   end-if.

* Parse reason to know why we have been dispatched
4   if REPPARMAREA = 'BKSPV' then
       display 'The user requested termination'
       stop run
   end-if.
5   if REPPARMAREA = 'ITSPV' then
       display ' * '
       go to forever
   end-if.

6   * If we reach here, there has been another reason
       display 'Unrecognized function from WM'.
       display REPPARMAREA.
       go to forever.

       stop run.

```

---

### Building sample applications

This section describes how to build the sample applications documented in this chapter. The main headings are:

- “Sample client and server, C, Windows NT”
- “Sample client and server, COBOL, OS/2 and WINDOWS NT” on page 179
- “Sample client application, COBOL, DOS and OS/2” on page 179
- “Running the sample programs” on page 180

### Sample client and server, C, Windows NT

#### Building a client program

1. Ensure that the LANDP C include file (EHCDEF.C) is in the defined INCLUDE search path.
2. Ensure that the correct .LIB file is in the defined LIBRARY search path.

For details, see “Compiling and linking your application program” on page 13.

#### Building the sample server

1. This sample uses service.h and service.obj to handle making an NT service. Compile with the following options:  
`/Gz /MT /DWIN_32 /D_DLL /D_MT`
2. Link with ehcwinnt.lib and advapi32.lib..
3. Ensure that the LANDP C include file (EHCDEF.C) is in the defined INCLUDE search path.
4. Ensure that the appropriate .LIB files are in the defined LIBRARY search path. See “Compiling and linking your application program” on page 13.

To handle making an NT service, the sample uses the header file service.h and the object file service.obj.

#### Running the application

1. Run autofbss.bat.
2. Load sample server, loader ldpsmain.exe, unless it is included in autofbss.bat.
3. Run the client, ldpcmain.exe.
4. To end the program, enter ehcfree ldpsmain.exe from the command line or stop LANDP with ehcfree spv.

### Sample client and server, COBOL, OS/2 and WINDOWS NT

Follow the initial steps in the *VisualAge for COBOL Getting Started* manual (which is supplied with VisualAge for COBOL) in the chapter "Build Your First VisualAge for COBOL Application". When the COBOL editor displays, you can either start writing your own application or use one of the sample programs supplied with LANDP. The following steps apply if you choose to use a sample program:

- Select **Get File** from the File menu
- Select a sample program (SAMPSEV.CBL or SAMP-CLI.CBL) which you will find in the samples sub-directory or EHC0500 or EHCN500 under the LANDP directory (for example, C:\EHC)
- Select **Save as ...** from the File menu and select the directory you chose for your project's files in an earlier step
- Close the COBOL editor

Press F5 in the window that shows the icon view of your project to refresh the view. The sample program's name and icon will then appear.

You must do the following before you compile and link the program:

- Select **Compile** from the **Options** menu
- On the first page of the notebook (Syntactical), select the option called **Process COPY, BASIS, and REPLACE statements**
- On the same page, select the **As-is** option under the title **Resolve program names:**
- Under Windows NT on the page with the tab **System**, in the **Call Interface Convention** field select **CDECL**
- On the page with the tab **Link**, fill in the field called **Enter library/object file name(s):** with the path and name of the LANDP for OS/2 32-bit library file (for example, C:\EHC\EHCO500\EHCOS232.LIB) or the LANDP for Windows NT 32-bit library file (for example, C:\EHC\EHCN500\EHCWINNT.LIB)
- On the page with the tab **Other**, fill in the field called **Enter copy file search path:** with the path containing the sample include file (for example, C:\EHC\EHCO500 or C:\EHC\EHCN500)
- Click the **OK** button to save your changes

Click on the **Build::Build normal** icon to compile and link the program.

### Sample client application, COBOL, DOS and OS/2

To build a 16-bit sample client, the IBM COBOL/2 compiler (or equivalent) is needed. Ensure that the LANDP include file EHCDEF.CBL is accessible by the compiler. For linking, the LANDP library file EHC.DOS.LIB is required for DOS and EHC.OS2.LIB for OS/2. Ensure that these libraries are in the defined library search path.

To compile the sample server file:

## sample client program, COBOL, DOS and OS/2

```
COBOL samplecb /vsc2;
```

To link for DOS:

```
LINK samplecb,,,ehcdos;
```

To link for OS/2:

```
LINK samplecb,,,pcobol + os2 + ehcos2 /NOP;
```

If you compile using the /LITLINK switch, you can take out the two underscores in '\_\_\_RMTREQ' when calling LANDP. The compiler then generates external references for all call literal statements. Depending on your needs, this may be desirable. For more details, look at the compiler documentation.

### Running the sample programs

See the information on user server definitions in the *LANDP Installation and Customization* book for details of setting up workgroups containing user servers.

If you intend to run your program on a machine which does not have VisualAge for COBOL installed you must create a package containing your application and the run-time files it uses. Please see the VisualAge for COBOL on-line documentation for details of the Package process.



---

## Appendix A. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:**

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM documentation or non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those documents or Web sites. The materials for those documents or Web sites are

not part of the materials for this IBM product and use of those documents or Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,  
Mail Point 151,  
Hursley Park,  
Winchester,  
Hampshire,  
England  
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

---

## Trademarks and service marks

The following terms are trademarks of the IBM Corporation in the United States or other countries, or both:

ACF/VTAM	MQSeries
AIX/6000	MVS/XA
AnyNet	OfficeVision
Application System/400	OfficeVision/MVS
AS/400	OfficeVision/VM
AT	Operating System/2
BookManager	Operating System/400
C/2	OS/2
CICS	OS/390
CICS OS/2	Presentation Manager
CICS/ESA	RETAIN
COBOL/2	RISC System/6000
Common User Access	RS/6000
CUA	S/390
DB2 Universal Database	SecureWay
Distributed Database	SP
Connection Services/2	ThinkPad
Distributed Relational	VisualAge
Database Architecture	VisualGen
DRDA	VM/ESA
Extended Services	VSE/ESA
IBM	VTAM
IBMLink	WIN-OS/2
IMS/ESA	Workplace Shell
LAN Distance	WebSphere
LANDP	Xstation Manager
Macro Assembler/2	XT
Micro Channel	

Lotus is a registered trademark of Lotus Development Corporation in the United States and other countries.

Tivoli and NetView are registered trademarks of Tivoli Systems Inc. in the United States and other countries. In Denmark, Tivoli is a trademark licensed from Kjøbenhavn Sommer - Tivoli A/S.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc in the United States and/or other countries.

Microsoft, Windows, Windows NT, Windows 2000, and the Windows logo are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Intel is a registered trademark of Intel.

PC/TCP and NetManage are registered trade marks of NetManage Inc..

Other company, product, and service names may be trademarks or service marks of others.

---

## Glossary

This glossary includes abbreviations, terms, and definitions used in the IBM LANDP Licensed Programs Family publications. It does not include all terms previously established for IBM networks, programs, operating systems, or other IBM products.

If you do not find the term you are looking for, refer to the *IBM Dictionary of Computing*.

This glossary includes terms and definitions from the following sources:

- The *IBM Dictionary of Computing*, New York: McGraw-Hill, copyright 1994 by International Business Machines Corporation. Copies may be purchased from McGraw-Hill or in bookstores.
- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- The ANSI/EIA Standard—440-A, *Fiber Optic Terminology*. Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006. Definitions are identified by the symbol (E) after the definition.
- The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Definitions that are specific to IBM products are so labeled, for example, “In LANDP,” or “In SNA.”

## A

**abend.** Abnormal end of task.

**abnormal end of task (abend).** Termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

**abstract class.** A class that provides common information for subclasses, and that therefore cannot be instantiated. Abstract classes provide at least one abstract method.

**abstract method.** A method with a signature, but no implementation. You provide the implementation of the method in the subclass of the abstract class that contains the abstract method.

**account.** In the AIX operating system, the log-in directory and other information that gives a user access to the system.

**ACF.** Advanced Communications Function.

**ACF/NCP.** Advanced Communications Function for the Network Control Program.

**activate logical unit request (ACTLU).** A request, sent by the host to the LANDP SNA server, to establish a logical session. The LANDP SNA server sends a positive response if the logical unit has been defined for this workstation.

**activate physical unit request (ACTPU).** A request, sent by the host to the LANDP SNA server, to establish a physical session.

**active.** In an XLR environment, the server (and, by implication, the workstation) that handles client requests and sends logging data to the backup.

**ACTLU.** Activate logical unit request.

**ACTPU.** Activate physical unit request.

**adapter.** (1) A part that electrically or physically connects a device to a computer or to another device.  
(2) A printed circuit board that modifies the system unit to allow it to operate in a particular way.

**address.** The unique code assigned to each device or workstation connected to a network. A standard Internet address is a 32-bit address field. This field can be broken into two parts. The first part contains the network address; the second part contains the host number.

**Advanced Communications Function (ACF).** (1) A group of IBM licensed programs, principally VTAM programs, TCAM, NCP, and SSP, that use the concepts of Systems Network Architecture (SNA), including distribution of function and resource sharing. (2) See also Network Control Program (NCP).

**Advanced Communications Function for the Network Control Program (ACF/NCP).** (1) An IBM program product that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. (2) See also Advanced Communications Function (ACF) and Network Control Program (NCP).

**advanced program-to-program communication (APPC).** The general facility characterizing the LU 6.2 architecture and its various implementations in products.

**AID.** Attention identifier.

**AIX (Advanced Interactive Executive).** IBM's licensed version of the UNIX operating system.

**alert.** (1) A message sent to a management services focal point in a network to identify a problem or an impending problem. (2) In the NetView program, a high-priority event that warrants immediate attention. A database record is generated for certain event types that are defined by user-constructed filters.

**alert condition.** A problem or impending problem for which information is collected and possibly forwarded for problem determination, diagnosis, or resolution.

**alert description.** Information in an alert table that defines the contents of a Systems Network Architecture (SNA) alert for a particular message ID.

**alert focal point.** The system in a network that receives and processes (logs, displays, and optionally forwards) alerts. An alert focal point is a subset of a problem management focal point.

**alert ID number.** A value created from specific fields in the alert using a cyclic redundancy check. A focal point uses this value to refer to a particular alert, for example, to filter out duplicate alerts.

**alert type.** A value in an alert that indicates the problem being reported.

**American National Standards Institute (ANSI).** An organization consisting of producers, consumers, and general interest groups, that establishes the procedures by which accredited organizations create and maintain voluntary industry standards in the United States. (A)

**ANSI.** American National Standards Institute.

**APAR.** Authorized program analysis report.

**API.** Application program interface.

**APPC.** Advanced program-to-program communication.

**applet.** A Java program designed to run within a Web browser. Contrast with application.

**application.** (1) In LANDP, a program using IBM LANDP for DOS, IBM LANDP for OS/2, IBM LANDP for Windows NT, IBM LANDP for AIX, IBM FBSS/2, IBM PC/Integrator, or IBM PC Integrator/2, tailored to the needs of the workstation user. (2) The use to which an information processing system is put; for example, a payroll application, an airline reservation application, a network application. (3) A collection of software components used to perform specific types of user-oriented work on a computer. (4) In Java programming, a self-contained, stand-alone Java program that includes a static main method. It does not require an applet viewer. Contrast with applet.

**application program.** (1) A program that is specific to the solution of an application problem. Synonymous with application software. (T) (2) A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll. (3) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

**application program interface (API).** (1) In LANDP, the common interface by which server functions are requested. Requests are expressed by issuing a call to the supervisor. (2) A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program. (3) The interface through which an application program interacts with an access method.

**application software.** (1) Software that is specific to the solution of an application problem. (T) Synonymous with application program. (2) Software coded by or for an end user that performs a service or relates to the user's work. (3) Software products such as games, spreadsheets, and word processing programs designed for use on a personal computer.

**argument.** (1) An independent variable. (I) (A) (2) Any value of an independent variable; for example, a search key; a number identifying the location of an item in a table. (I) (A) (3) A parameter passed between a calling program and a called program.

**arrival sequence.** An order in which records are retrieved that is based on the order in which records are stored in a physical file.

**AS/400®.** IBM Application System/400®.

**ASCII (American National Standard Code for Information Interchange).** The standard code, using a coded character set consisting of 7-bit coded characters (8-bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

**Note:** IBM has defined an extension to ASCII code (characters 128-255).

**ASCIIZ format.** A string of ASCII characters ending with a null character (X'00').

**ASYNC.** Asynchronous.

**asynchronous (ASYNC).** (1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (T) (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions.

**attention identifier (AID).** (1) A code in the inbound 3270 data stream that identifies the source or type of data that follow. (2) A character in a data stream indicating that the user has pressed a key, such as the Enter key, that requests an action by the system.

**authorization.** (1) In computer security, the right granted to a user to communicate with or make use of a computer system. (T) (2) An access right. (3) The process of granting a user either complete or restricted access to an object, resource, or function.

**authorized program analysis report (APAR).** A report of a problem caused by a suspected defect in a current unaltered release of a program.

## B

**back-out.** To restore a file to a previous condition by removing changes in the inverse chronological order from which the changes were originally made.

**backup.** In an XLR environment, the server (and, by implication, the workstation) that accepts logging data from the active and maintains a mirror set of databases (at a transaction level).

**BASIC.** (1) Beginner's all-purpose symbolic instruction code. A procedural algebraic language originally designed for ease of learning with a small instruction repertoire. (A) (2) A high-level programming language with a small number of statements and a simple syntax that is designed to be easily learned and used and that is widely used for interactive applications on microcomputers.

**Basic Input/Output System (BIOS).** (1) Code that controls basic hardware operations, such as interactions with diskette drives, hard disk drives, and the keyboard. (2) See also NetBIOS.

**BAT, bat.** (1) A DOS batch file extension (.BAT). (2) A batch file that contains a series of commands to be processed sequentially.

**BB.** Begin bracket.

**begin bracket (BB).** (1) An SNA bracket protocol term issued by the LANDP SNA server when bracket protocol is requested in the bind session. (2) Contrast with end bracket.

**BID.** In SNA, a request to start a bracket.

**bind.** To associate a variable with an absolute address, identifier, or virtual address, or with a symbolic address or label in a program.

**BIND.** (1) In SNA, a request to start a session between two logical units. (2) Contrast with UNBIND.

**binding.** (1) In programming, an association between a variable and a value for that variable that holds within a defined scope. The scope may be that of a rule, a function call, or a procedure invocation. (T) (2) In the AIX operating system, a temporary association between

a client and both an object and a server that exports an interface to the object. A binding is meaningful only to the program that sets it and is represented by a bound handle.

**BIOS.** Basic Input/Output System.

**block.** (1) The smallest complete unit of data that can be transmitted between units in a communication network. The maximum size of a block depends on the characteristics of the sending or receiving unit. (2) A group of contiguous characters recorded as a unit. (3) See also connectivity programming request block, program control block.

**browser.** An Internet-based tool that lets users browse web sites.

**buffer.** (1) A routine or storage used to compensate for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another. (A) (2) A portion of storage used to hold input or output data temporarily.

## C

**C language.** A language used to develop software applications in compact, efficient code that can be run on different types of computers with minimal change.

**call.** In LANDP, the invocation of one of the LANDP API routines, RMTREQ, GETRPLY and RMTAREQ (client calls) and GETREQ, RMTRPLY, and SRVINIT (server calls). A LANDP client uses the RMTREQ call to request a LANDP function. Calls use the connectivity programming request block (CPRB) to pass and receive information. The syntax of a call varies with the programming language. The following examples are for COBOL and C respectively

```
CALL "RMTREQ" USING BY REFERENCE EHC-CPRB
                     BY VALUE      EHC-RESERVED
```

```
retcode = GETREQ(&mycprb, EHC_RESERVED);
```

**CCITT.** Comité Consultatif International Télégraphique et Téléphonique. The International Telegraph and Telephone Consultative Committee.

**CD.** Compact disc.

**CD-ROM.** Compact disc-read-only memory.

**CICS®.** Customer Information Control System.

**CID.** Configuration, Installation, and Distribution. An IBM standard methodology for installing and distributing products under DOS, OS/2, and Windows 3.1.

**ciphertext.** (1) In computer security, text produced by encryption. (2) Synonym for enciphered data.

**cleartext.** (1) Nonencrypted data. (2) Synonym for plaintext.

**class.** An encapsulated collection of data and methods to operate on data. A class can be instantiated to produce an object that is an instance of the class.

**CLASSPATH.** In your deployment environment, the environment variable keyword that specifies the directories in which to look for class and record files.

**client.** (1) A functional unit that receives shared services from a server. (T) (2) A user. (3) See also client/server, client workstation, server, and user.

**client workstation.** (1) In IBM LANDP for DOS, IBM LANDP for OS/2, IBM LANDP for AIX, IBM LANDP for Windows NT, IBM FBSS/2, IBM PC/Integrator, and IBM PC Integrator/2, a workstation in a LAN that uses a service. (2) See also client, client/server, server, and user.

**client/server.** (1) In communications, the model of interaction in distributed data processing in which a program at one site sends a request to a program at another site and awaits a response. The requesting program is called a client; the answering program is called a server. (2) See also client, client workstation, server, and user.

**CLIST, clist.** Command list.

**close.** (1) A LANDP family function used to release a server. (2) To end the processing of a file. (3) A data manipulation function that ends the connection between a file and a program. (4) Contrast with open.

**COBOL.** Common business-oriented language. A high-level programming language, based on English, that is used primarily for business applications.

**code page.** An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, assignment of characters and meanings to 128 code points for a 7-bit code.

**collating sequence.** A specified arrangement used in sequencing. (I) (A)



**COM, com.** A DOS file with the file extension .COM.

**command.** (1) Loosely, a mathematical or logic operator. (A) (2) A request from a terminal for performance of an operation or processing of a program. (3) A character string from a source external to a system that represents a request for system action.

**command list (CLIST, clist).** A list of commands and statements designed to perform a specific function for the user.

**Common User Access™ architecture.** Guidelines for the dialog between a human and a workstation or terminal. One of the three SAA architectural areas.

**communication configuration.** In LANDP, the process of selecting and describing to the LANDP programs the particular arrangement of communication functions about a particular user.

**communication controller.** (1) A device that directs the transmission of data over the data links of a network; its operation may be controlled by a program executed in a processor to which the controller is connected or it may be controlled by a program executed within the device. (T) (2) A type of communication control unit whose operations are controlled by one or more programs stored and executed in the unit. It manages the details of line control and the routing of data through a network.

**communication server.** A server that communicates with a remote computer for various workstations in a local area network.

**Communications Server.** An IBM licensed program that supports the development and use of OS/2 applications involving two or more connected systems or workstations. IBM SecureWay Communications Server for OS/2 Warp provides multiple concurrent connectivities using different protocols for IBM 3270 and 5250 emulation sessions, printer sessions, and file transfers. It supports a range of application programming interfaces (API), which may be called concurrently and are designed for a variety of applications. IBM SecureWay Communications Server for OS/2 Warp includes the necessary interfaces for network management.

**compact disc (CD).** (1) A disc, usually 4.75 inches in diameter, from which data is read optically by means of a laser. (2) A disc with information stored in the form of pits along a spiral track. The information is decoded by

a compact-disc player and interpreted as digital audio data, which most computers can process.

**compact disc-read-only memory (CD-ROM).** A 4.75-inch optical memory storage medium, capable of storing about 550 megabytes of data. The standards for CD-ROM storage are known as the "Yellow Book."

**compaction.** (1) Any method for encoding data to reduce the storage it requires. (2) In SNA, the transformation of data by packing two characters in a byte so as to take advantage of the fact that only a subset of the allowable 256 characters is used; the most frequently sent characters are compacted. (3) See also compression and encode.

**compression.** (1) The process of eliminating gaps, empty fields, redundancies, and unnecessary data to shorten the length of records or blocks. (2) In SNA, the replacement of a string of up to 64 repeated characters by an encoded control byte to reduce the length of the data stream sent to the LU-LU session partner. The encoded control byte is followed by the character that was repeated (unless that character is the prime compression character). (3) Contrast with decompression.

**config.sys.** A file created during the customization process that holds the details about the system configuration. The CONFIG.SYS file is used during system operation.

**configuration.** (1) The manner in which the hardware and software of an information processing system are organized and interconnected. (T) (2) The physical and logical arrangement of devices and programs that make up a data processing system. (3) The devices and programs that make up a system, subsystem, or network.

**connection.** (1) An association established between functional units for conveying information. (2) The path between two protocol modules that provide reliable stream delivery service. On the Internet, a connection extends from a TCP module on one machine to a TCP module on the other.

**connectivity.** The capability to attach a variety of functional units without modifying them.

**connectivity programming request block (CPRB).** The control block used for communication between a server and a client. This control block contains the information that is exchanged between clients and

servers, and the information required for routing the requests and replies.

**constructor.** A method called to set up a new instance of a class.

**control program.** A computer program designed to schedule and supervise the execution of programs of a computer system. (I) (A)

**coprocessor.** (1) A supplementary processor that performs operations in conjunction with another processor. (2) In personal computers, a microprocessor on an expansion board that extends the address range of the processor in the system unit or adds specialized instructions to handle a particular category of operations; for example, an I/O coprocessor, math coprocessor, or networking coprocessor.

**corrective service diskette.** A diskette provided by IBM to registered service coordinators for resolving user-identified problems with previously installed software. This diskette includes program updates designed to resolve problems.

**CPRB.** Connectivity programming request block.

**CRC.** The cyclic redundancy check character. (A)

**critical error handler.** A routine that the operating system calls automatically if an error occurs in an operating system function call. There is a standard error handler or the user can provide one for special functions.

**CRV.** Cryptography verification request.

**cryptography.** (1) The transformation of data to conceal its meaning. (2) In computer security, the principles, means, and methods for encrypting plaintext and decrypting ciphertext.

**cryptography key.** A parameter that determines cryptographic transformations between plaintext and ciphertext.

**cryptography verification (CRV) request.** A request unit sent by the primary logical unit (PLU) to the secondary logical unit (SLU) as part of cryptographic session establishment, to allow the SLU to verify that the PLU is using the correct session cryptography key and initialization vector (IV).

**CTS.** Clear to Send.

**CUA™ architecture.** Common User Access™ architecture.

**cursor.** (1) A movable, visible mark used to show a position of interest on a display surface. (A) (2) In SAA Common User Access architecture, a visual cue that shows a user where keyboard input will appear on the screen.

**Customer Information Control System (CICS®).** An IBM licensed program that allows transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases.

**Customer Information Control System for Virtual Storage (CICS/VS).** An IBM licensed program used in a communications network.

**customization.** The process of designing a data processing installation or network to meet the requirements of particular users.

**customization workstation.** A workstation on which LANDP is installed, and which is used to customize a LANDP workgroup.

**cyclic redundancy check character (CRC).** A character used in a modified cyclic code for error detection and correction. (A)

## D

**DASD.** Direct access storage device.

**data circuit-terminating equipment (DCE).** In a data station, the equipment that provides the signal conversion and coding between the data terminal equipment (DTE) and the line. (I)

### Notes:

1. The DCE may be separate equipment or a part of the DTE or an integral part of the DTE or of the intermediate equipment.
2. A DCE may perform other functions that are usually performed at the network end of the line.

**Data Encryption Standard (DES).** In computer security, the National Institute of Standards and Technology (NIST) Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm.

**data flow control (DFC).** In SNA, a request/response unit (RU) category used for requests and responses exchanged between the data flow control layer in one half-session and the data flow control layer in the session partner. Half duplex, flip-flop is the only LANDP-supported data flow control for both send and receive.

**data link control (DLC).** (1) In SNA, the layer that consists of the link stations that schedule data transfer over a link between two nodes and perform error control for the link. Examples of data link control are SDLC for serial-by-bit link connection and data link control for the System/370™ channel. (2) See also Systems Network Architecture (SNA). (3) In SNA, a set of rules used by two nodes on a data link to accomplish an orderly exchange of information.

**data set.** The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. Sometimes called a file.

**data terminal equipment (DTE).** The part of a data station that serves as a data source, data sink, or both. (I) (A)

**database description (DBD).** (1) In LANDP, the shared-file server descriptor. (2) In IMS/VS, the collection of macro-parameter statements that describes an IMS/VS database. These statements describe the hierarchical structure, IMS/VS organization, device type, segment length, sequence fields, and alternate search fields. The statements are assembled to produce database description blocks.

**datagram.** The basic unit of information that is passed across the Internet. It consists of one or more data packets.

**DBCS.** Double-byte character set.

**DBD.** Database description.

**DBM.** Database manager.

**DCA.** Direct communication adapter.

**DCE.** (1) Data circuit-terminating equipment. (2) Distributed Computing Environment.

**DDE.** Dynamic data exchange.

**DDT.** Diagnostic and debugging tool.

**decipher.** (1) To convert enciphered data in order to restore the original data. (T) (2) In computer security, to convert ciphertext into plaintext by means of a cipher system. (3) To convert enciphered data into clear data. (4) Synonymous with decrypt. (5) Contrast with encipher.

**decompression.** (1) A function that expands data to the length that preceded data compression. (2) Contrast with compression.

**decrypt.** (1) In computer security, to decipher or decode. (2) Synonymous with decipher. (T)

**default.** A value, attribute or option that is assumed when none is explicitly specified.

**delimiter.** (1) A character used to show the beginning and end of a character string. (T) (2) A character that groups or separates words or values in a line of

**deprecation.** An obsolete component that may be deleted from a future release of a product.

**DES.** Data Encryption Standard.

**development workstation.** A workstation which is part of a LANDP workgroup, and which is customized via a customization workstation.

**device driver.** In Advanced DOS, a file that contains the code needed to attach and use a device.

**DFC.** Data flow control.

**DIN.** Deutsches Institut für Normung.

**direct access.** (1) The capability to obtain data from a storage device, or to enter data into a storage device, in a sequence independent from their relative position, by means of addresses indicating the physical position of the data. (T) (2) Contrast with sequential access.

**direct access storage device (DASD).** A device where access time is effectively independent of the location of the data.

**directory.** (1) A table of identifiers and references to the corresponding items of data. (I) (A) (2) A type of file containing the names and controlling information for other files or other directories. (3) An index that is used by a control program to locate one or more blocks of data that are stored in separate areas of a data set in direct access storage. (4) A listing of the files stored on a diskette.

**directory service (DS).** An application service element that translates the symbolic names used by application processes into the complete network addresses used in an OSI environment. (T)

**disk.** (1) A round, flat data medium that is rotated to read or write data. (T) (2) Loosely, a magnetic disk unit.

**disk operating system.** An operating system for computer systems that use disks and diskettes for auxiliary storage of programs and data.

**diskette.** (1) A thin, flexible magnetic disk and a semi-rigid protective jacket, where the disk is permanently enclosed. (2) Contrast with hard disk.

**Distributed Computing Environment (DCE).** The Open Software Foundation (OSF) specification (or a product derived from this specification) that assists in networking. DCE provides such functions as authentication, directory service (DS), and remote procedure call (RPC).

**distributed system.** A data processing system where processing, storage, and control functions, and also input and output operations, are distributed among remote locations.

**distribution diskette.** A diskette on which IBM sends programs and documentation to a customer.

**DLC.** Data link control.

**DLL.** Dynamic link library.

**DMA.** Direct memory access.

**domain.** (1) The part of a computer network where the data processing resources are under common control. (T) (2) In a database, all the possible values of an attribute or a data element. (3) In SNA, a system services control point (SSCP) and the physical units (PUs), logical units (LUs), links, link stations, and all associated resources that the SSCP could control with activation requests and deactivation requests.

**DOS.** Disk Operating System.

**double-byte character set (DBCS).** (1) A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2

bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. (2) Contrast with single-byte character set (SBCS).

**DS.** Directory service.

**DSR.** Data Set Ready.

**DTE.** Data terminal equipment. (A)

**DTE/DCE interface.** The physical interface and link access procedures between a data terminal equipment (DTE) and a data circuit-terminating equipment (DCE).

**dynamic data exchange (DDE).** The exchange of data between programs or between a program and a data-file object. Any change made to information in one program or session is applied to the identical data created by the other program.

**dynamic link library (DLL).** A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

## E

**EB.** End bracket.

**EBCDIC.** Extended binary-coded decimal interchange code.

**EGA.** Enhanced graphics adapter.

**EID.** End-of-message (EOM) identification.

**EMM.** Expanded memory manager.

**emulation.** The use of a data processing system to imitate another data processing system, so that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated system. Emulation is usually achieved with hardware or firm-ware. (T)

**encipher.** (1) To scramble data or to convert data to a secret code that masks the meaning of the data to any unauthorized recipient. Synonymous with encrypt. (T) (2) In computer security, to convert plaintext into an unintelligible form by means of a cipher system. Synonymous with cipher. (3) Contrast with decipher. See also encode.

**enciphered data.** (1) Data whose meaning is concealed from unauthorized users or observers. (2) Synonymous with encode.

**encode.** (1) To convert data by the use of a code in such a manner that reconversion to the original form is possible. (T) (2) In computer security, to convert plaintext into an unintelligible form by means of a code system. (3) See also plaintext.

**encrypt.** (1) In computer security, to encode or encipher. (2) Synonym for encipher. (T)

**end bracket (EB).** (1) An SNA bracket protocol term used when the bind session specifies the end bracket call. If specified in the bind session, the personal computer may send both begin bracket and end bracket calls (not-response mode protocol). (2) Contrast with begin bracket.

**end-of-message (EOM).** The character or sequence of characters that shows the end of a message or record.

**enhanced graphics adapter (EGA).** An adapter, such as the IBM Enhanced Graphics Adapter, that provides high-resolution graphics, allowing the use of a color display for text processing and also graphics applications.

**environment.** A named collection of logical and physical resources used to support the performance of a function.

**EOM.** End-of-message.

**erase.** To remove data from a data medium. Erasing is usually accomplished by overwriting the data or deleting the references. (T)

**error log.** (1) A data set or file in a product or system where error information is stored for later access. (2) A record of machine checks, device errors, and volume statistical data.

**error message.** An indication that an error has been detected. (A)

**ERRORLEVEL.** A parameter of the IF command used by batch files. It is used in testing for failure of recently loaded programs.

**event.** (1) An occurrence or happening. (2) An occurrence of significance to a task; for example, the completion of an asynchronous operation, such as an input/output operation. (3) A data link control command

and response passed between adjacent nodes that allows the two nodes to exchange identification and other information necessary for operation over the data link. (4) In the NetView program, a record indicating irregularities of operation in physical elements of a network.

**exception.** An object that has caused some new condition, such as an error. In Java, throwing an error means passing that object to an interested party. A signal indicates what condition has occurred. Catching the condition means receiving the sent object. Handling this exception means dealing with the problem after receiving the object (though it might mean doing nothing, which is bad programming practice).

**exchange identification (XID).** The ID that is exchanged with the remote physical unit when an attachment is first established.

**EXE, exe.** An executable file with the file extension .EXE.

**extended ASCII.** A set of ASCII codes that uses the eighth (most significant) bit to define 127 additional codes. Standard ASCII uses 7 bits and defines 128 codes.

**extended binary-coded decimal interchange code (EBCDIC).** A coded character set of 256 8-bit characters.

**external logging replicator (XLR).** Shared-file mode of operation in which fault-tolerant data replication is achieved by logging database updates to an external server.

## F

**facility.** (1) An operational capability, or the means for providing such a capability. (T) (2) A service provided by an operating system for a particular purpose; for example, the checkpoint/restart facility.

**FBSI.** Financial Branch Systems Integrator.

**FBSS (DOS).** IBM Financial Branch Systems Service (DOS). The predecessor to LANDP.

**FBSS/2.** Financial Branch Systems Service/2.

**FCB.** File control block.

**FIC.** First-in-chain.

**file.** (1) A named set of records stored or processed as a unit. (T) (2) A collection of information treated as a unit. (3) A collection of data that is stored and retrieved by an assigned name.

**file control block (FCB).** A record that contains all of the information about a file, such as its structure, length, and name.

**file index table (FIT).** A table used by WorkSpace On-Demand to redirect file access requests from a client workstation's boot drive to the appropriate location on the boot server.

**file server.** A high-capacity disk storage device or a computer that each computer on a network can use to access and retrieve files that can be shared among the attached computers.

**file transfer.** In remote communications, the transfer of one or more files from one system to another over a communications link.

**first-in-chain (FIC).** A request unit (RU) whose request header (RH) begin chain indicator is on and whose RH end chain indicator is off.

**FIT.** file index table

**fixed disk.** Synonym for hard disk.

**flag.** (1) A variable indicating that a certain condition holds. (T) (2) Any of various types of indicators used for identification; for example, a word mark. (A) (3) A character that signals the occurrence of some condition, such as the end of a word. (A)

**FMH.** Function management header.

**format identification (FID) field.** In SNA, a field in each transmission header (TH) that shows the format of the transmission header; that is, the presence or absence of certain fields.

**forward recovery.** The process of reconstructing a file from a particular point by restoring a saved version of the file and then applying changes to that file in the same order in which they were originally made.

**function.** (1) In IBM LANDP for DOS, IBM LANDP for OS/2, IBM LANDP for Windows NT, IBM FBSS (DOS), IBM FBSS/2, IBM PC/Integrator, and IBM PC Integrator/2 a function is the specification of an activity to be performed by a server. (2) In computer programming, synonym for procedure.

**function management header (FMH).** (1) A special record or part of a record that contains control information for the data that follow. (2) In SNA, one or more headers optionally present in the leading request units (RUs) of an RU chain that allow a half-session in an LU-LU session to: (a) select a destination as session partner and control way where end-user data it sends are handled at the destination, (b) change destination or characteristics of data during session, and (c) send between session partners status or user information about destination; for example, whether it is a program or device.

## G

**gateway.** (1) In LANDP, the workstation that connects the LANDP workgroup to a host computer with the necessary LANDP software and the respective physical attachment. (2) A functional unit that interconnects two computer networks with different network architectures. A gateway connects networks or systems of different architectures. A bridge interconnects networks or systems with the same or similar architectures. (T) (3) A network that connects hosts. (4) Contrast with router.

**generic alert.** A product-independent method of encoding alert data by means of both (a) code points indexing short units of stored text and (b) textual data.

## H

**hard disk.** (1) A rigid magnetic disk such as the internal disks used in the system units of IBM personal computers and in external hard disk drives. (2) Synonym for fixed disk. (3) Contrast with diskette.

**HDLC.** High-level data link control.

**hexadecimal.** Describing a numbering system with base of sixteen; valid numbers use the digits 0 through 9 and characters A through F, where A represents 10 and F represents 15.

**high-level data link control (HDLC).** In data communication, the use of a specified series of bits to control data links under the International Standards for HDLC: ISO 3309 Frame Structure and ISO 4335 Elements of Procedures.

**host, host computer, host processor, or host system.** (1) The primary or controlling computer in a multiple computer installation. (2) A computer used to prepare programs for use on another computer or on

another data processing system; for example, a computer used to compile, link edit, and test programs to be used on another system.

**hot-key.** The key combination used to change from one session to another on the workstation.

**Hypertext Transfer Protocol (HTTP).** The Internet protocol, based on TCP/IP, that is used to fetch hypertext objects from remote hosts.

## I

**I/O.** Input/output.

**IBM Operating System/2® (OS/2).** Pertaining to the IBM licensed program that can be used as the operating system for personal computers. The OS/2 licensed program can perform multiple tasks at the same time.

**ICV.** Initial chaining value.

**ID.** (1) Identifier. (2) Identification.

**identification.** In computer security, the process that allows a system to recognize an entity with personal, equipment, or organizational characteristics or codes.

**identifier.** One or more characters used to identify or name a data element or possibly to show certain properties of that data element. (A)

**IEEE.** Institute of Electrical and Electronics Engineers.

**IMS/VS.** Information Management System/Virtual Storage.

**indexed access.** Pertaining to the organization and accessing of the records of a storage structure through a separate index to the locations of the stored records. (A)

**indexed sequential access.** Pertaining to the organization and accessing of records through an index of the keys that are stored in arbitrarily partitioned sequential files. (A)

**initial chaining value (ICV).** An 8-byte pseudo-random number used to verify that both ends of a session with cryptography have the same session cryptography key. The initial chaining value is also used as input to Data Encryption Standard (DES) algorithm to encipher or decipher data in a session with cryptography.

**initial program load (IPL).** (1) The initialization procedure that causes an operating system to begin

operation. (2) The process by which a configuration image is loaded into storage at the beginning of a work day or after a system malfunction. (3) The process of loading system programs and preparing a system to run jobs.

**initialization.** (1) The operations required for setting a device to a starting state, before the use of a data medium, or before implementation of a process. (T) (2) Preparation of a system, device, or program for operation.

**initiate self.** An SNA command issued by the LANDP SNA server to initiate a host application. The SNA command is issued in response to the receipt of an Open command from the personal computer.

**INITSELF.** Initiate self.

**input/output (I/O).** (1) Describing a device whose parts can perform an input process and an output process at the same time. (I) (2) Describing a functional unit or channel involved in an input process, output process, or both, concurrently or not, and to the data involved in such a process.

**Instruction Pointer (IP).** In System 38, a pointer that provides addressability for a machine interface instruction in a program.

**interface.** A shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics, as appropriate. The concept includes the specification of the connection of two devices having different functions. (T)

**International Organization for Standardization (ISO).** An organization of national standards bodies from various countries established to promote development of standards to simplify international exchange of goods and services, and develop cooperation in intellectual, scientific, technological, and economic activity.

**Internet Protocol (IP).** A protocol used to route data from its source to its destination in an Internet environment.

**interoperability.** (1) The capability to communicate, execute programs, or transfer data among various functional units in a way that requires the user to have little or no knowledge of the unique characteristics of those units. (T) (2) In SAA usage, the ability to link SAA and non-SAA environments and use the combination for distributed processing.

**IP.** (1) Instruction Pointer. (2) Internet Protocol.

**IPL.** Initial program load.

**ISAM.** Indexed sequential access method.

**ISO.** International Organization for Standardization.

## J

**Jar file format.** Java Archive, a platform-independent file format that aggregates many files into one. Multiple Java applets and their requisite components (.class files, images, sounds, and other resource files) can be bundled in a JAR file and subsequently downloaded to a browser in a single HTTP transaction.

**Java.** An object-oriented programming language for portable, interpretive code that supports interaction among remote objects. Java was specified and developed by Sun Microsystems, Incorporated. The Java environment consists of the JavaOS, the Virtual Machines for various platforms, the object-oriented Java programming language, and several class libraries.

**Java Development Kit (JDK).** A set of Java technologies made available to licensed developers by Sun Microsystems. Each release of JDK consists of the Java compiler, Java virtual machine, Java class libraries, Java applet viewer, Java debugger, and other tools.

**JavaDoc.** Sun Microsystems tool for generating HTML documentation of classes by extracting comments from the Java source code files.

**Java Remote Method Invocation (RMI).** Method invocation between peers, or between client and server, when applications at both ends of the invocation are written in Java. Java RMI is included in JDK 1.1.

**Java Virtual Machine.** A software implementation of a central processing unit (CPU) that runs compiled Java code (applets and applications).

**journal.** (1) A chronological record of changes made in a set of data; the record may be used to reconstruct a previous version of the set. (T) (2) A special-purpose data set that provides an audit trail of operator and system actions, or as a means of recovering superseded data.

**JVM.** Java Virtual Machine.

## K

**KB.** Kilobyte; 1024 bytes.

**key.** (1) An identifier within a set of data elements. (T) (2) One or more characters used to identify the record and establish the order of the record within an indexed file.

**keystroke.** Actuation of a key on a keyboard to perform or release a machine function. (T)

**keyword.** A name or symbol that identifies a parameter or an ordered set of parameters.

## L

**LAN.** Local area network.

**LAN configuration.** The process by which the details about the structure of the LAN for a particular user are provided to the LANDP family programs. This includes details about the workstations forming the LAN, the services provided by each workstation, and the workstations that receive the services.

**LAN trace.** A LANDP family trace facility that informs about the LANDP-related LAN and displays the status of the local area network.

**LAN Distributed Platform.** The former name for the LANDP family of products.

**last-in-chain (LIC).** A request unit (RU) whose request header (RH) end chain indicator is on and whose RH begin chain indicator is off.

**LDA.** Logical device address.

**LED.** Light-emitting diode.

**LIC.** Last-in-chain.

**light-emitting diode (LED).** A semiconductor chip that gives off visible or infrared light when operated.

**link connection.** In SNA, the physical equipment providing two-way communication between one link station and one or more other link stations; for example, a telecommunication line and data circuit-terminating equipment (DCE).

**LIP.** LAN Internet Protocol.



**LLAP.** Logical link access path.

**loader.** A routine, commonly a computer program, that reads data into main storage. (A)

**local area network (LAN).** A computer network located on a user's premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. (T)

**local host.** In the Internet, the computer to which a user's terminal is directly connected without using the Internet.

**logging.** The recording of data about specific events.

**logical device address (LDA).** (1) A number used to represent a terminal or terminal component within a workstation. (2) See also physical device address.

**logical link access path (LLAP).** In a multi-system environment, the path between any two systems. One or more logical link paths must be defined for each logical link.

**logical unit (LU).** (1) In SNA, a port through which an end user accesses the SNA network to communicate with another end user and through which the end user accesses the functions provided by the system services control points (SSCPs). An LU can support at least two sessions, one with an SSCP and one with another LU, and may be capable of supporting many sessions with other logical units. (2) A type of network addressable unit that allows end users to communicate with each other and gain access to network resources.

**longitudinal parity check.** A parity check of a row of binary digits that are members of a set forming a matrix; for example, a parity check of the bits of a track in a block on a magnetic stripe. (T)

**longitudinal redundancy check (LRC).** Synonym for longitudinal parity check.

**LRC.** Longitudinal redundancy check.

**LU.** Logical unit.

**LU—LU session type 0.** In SNA, a type of session between two LU—LU half-sessions using SNA-defined protocols for transmission control and data flow control, but using end-user or product-defined protocols to augment or replace FMD services protocols.

**LU—LU session type 1.** In SNA, a type of session between an application program and single- or multiple-device data processing terminals in an interactive, batch data transfer, or distributed processing environment.

**LU—LU session type 2.** In SNA, a type of session between an application program and a single display terminal in an interactive environment, using the SNA 3270 data stream.

**LUSTAT.** An SNA command used to send logical unit status information.

## M

**MAC.** Message authentication code.

**mapper.** A device, such as a piece of code, which performs a mapping function.

**mapping.** (1) A list, usually in a profile, that establishes a correspondence between items in two groups; for example, a keyboard mapping can establish what character is displayed when a certain key is pressed. (2) In a database, the establishing of correspondences between a given logical structure and a given physical structure. (T)

**MB.** Megabyte; 1 048 576 bytes.

**memory.** All of the addressable storage space in a processing unit and other internal storages that is used to execute instructions. (T)

**message.** (1) An assembly of characters and sometimes control codes that is transferred as an entity from an originator to one or more recipients. A message consists of two parts: envelope and content. (T) (2) A communication sent from a person or program to another person or program. (3) A unit of data sent over a telecommunication line. (4) One or more message segments transmitted among terminals, application programs, and systems. (5) In SAA Common User Access architecture, information not requested by a user but displayed by an application in response to an unexpected event, or when something undesirable could occur.

**message authentication code (MAC).** (1) In computer security, a value, part of, or accompanying a message, used to determine that the contents, origin, author, or other attributes of all or part of the message are as they appear to be. (2) In cryptography: (a) a number or

value derived by processing data with an authentication algorithm, (b) the cryptographic result of block cipher operations on text or data using a cipher block chain (CBC) mode of operation, (c) a digital signature code.

**method.** A fragment of Java code within a class that can be invoked and passed a set of parameters to perform a specific task.

**MIC.** Middle-in-chain.

**MICR.** Magnetic ink character recognition.

**microcode.** (1) One or more microinstructions. (2) A code, representing the instructions of an instruction set, that is done in a part of storage that is not program-addressable. (3) To design, write, and also to test one or more microinstructions.

**middle-in-chain (MIC).** A request unit (RU) whose request header (RH) begin chain indicator and RH end chain indicator are both off.

**mnemonic.** A symbol chosen to help the user remember the significance of the symbol.

**mode.** A method of operation.

**mode switching.** Operator switching between a concurrently running personal computer application and 3270 emulation or other internal application.

**MSR, MSR/E.** Magnetic stripe reader; Magnetic stripe reader/encoder.

**multi-tasking.** A mode of operation that provides for concurrent performance, or interleaved execution of two or more tasks. (I) (A)

**MVDM.** Multiple Virtual DOS Machine.

## N

**name server.** (1) The server that stores resource records about hosts. (2) In the AIX operating system, a host that provides name resolution for a network. Name servers translate symbolic names assigned to networks and hosts into the Internet addresses used by machines. (3) In TCP/IP, synonym for domain name server.

**NAU.** Network addressable unit.

**NCP.** Network Control Program.

**NDIS.** Network Driver Interface Specification

**NetBIOS.** (1) Network Basic Input/Output System. A standard interface to networks, IBM personal computers (PCs), and compatible PCs, that is used on LANs to provide message, print-server, and file-server functions. Application programs that use NetBIOS do not need to handle the details of LAN data link control (DLC) protocols. (2) See also BIOS.

**NetView program.** An IBM licensed program used to monitor and manage a network and to diagnose network problems.

**network.** (1) An arrangement of nodes and connecting branches. (T) (2) A configuration of data processing devices and software connected for information interchange.

**network addressable unit (NAU).** (1) In SNA, a logical unit, a physical unit, or a system services control point. The NAU is the origin or the destination of information transmitted by the path control network. (2) See also logical unit, physical unit, system services control point (SSCP).

**Network Control Program (NCP).** (1) An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. (2) See also Advanced Communications Function (ACF).

**network management vector transport (NMVT).** A management services request/response unit (RU) that flows over an active session between physical unit management services and control point management services (SSCP-PU session).

**network resource.** In ACF/VTAM®, a network component such as a local network control program, an SDLC data link, or a peripheral node.

**network services procedure error (NSPE).** A request unit that is sent by a system services control point (SSCP) to a logical unit (LU) when a procedure requested by that LU has failed.

**NLS.** National language support.

**NMVT.** Network management vector transport.

**node.** (1) In a network, a point at which one or more functional units connect channels or data circuits. (I) (2) In network topology, the point at an end of a branch. (T)

**NPSI.** X.25 NCP Packet Switching Interface.

**NSPE.** Network services procedure error.

## O

**object.** The principal building block of object-oriented programs. Objects are software programming modules. Each object is a programming unit consisting of related data and methods.

**object-oriented programming (OOP).** A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates on the data objects that constitute the problem and how they are manipulated, not on how something is accomplished.

**ODBC.** Open Database Connectivity is a standardized set of API function calls that can be used to access data stored in both relational and non-relational DBMSs.

**OIA.** Operator information area.

**OIC.** Only-in-chain.

**only-in-chain (OIC).** A request unit (RU) for which the request header (RH) begin chain indicator and RH end chain indicator are both on.

**OOP.** object-oriented programming

**open.** (1) The function that connects a file to a program for processing. (2) Contrast with close.

**open system.** A system with specified standards, and that therefore can be readily connected to other systems that comply with the same standards.

**operating system.** Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible. (T)

**operator information area (OIA).** In the 3270 Information Display System, the area near the bottom of the display area where terminal or system status information is displayed.

**option.** A specification in a statement that may be used to influence the processing of the statement.

**OS/2 operating system.** IBM Operating System/2.

## P

**pacing.** A technique by which a receiving station controls the rate of transmission of a sending station to prevent overrun.

**package.** A program element that contains classes and interfaces.

**packet.** A sequence of binary digits, including data and control signals, that is transmitted and switched as a composite entity.

**panel.** A formatted display of information that appears on a display screen.

**parallel port.** (1) On a personal computer system, a port used to attach devices such as dot matrix printers and input/output units; it transmits data one byte at a time. (2) See also serial port.

**parameter.** (1) A variable that is given a constant value for a specified application and that may denote the application. (I) (A) (2) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted. (3) Data passed between programs or procedures.

**Pascal.** A high-level, general purpose programming language, related to ALGOL. Programs written in Pascal are block structured, consisting of independent routines. They can run on different computers with little or no modification.

**path.** In a personal computer system, the logical relationship between directories.

**PBM.** Personal banking machine.

**PC.** Personal computer.

**PC-ID.** Workstation identifier.

**PCB.** Program control block.

**PC/TCP.** FTP Software's implementation of TCP/IP for systems running DOS and Windows. Now called PC/TCP Network Software version 5.0 and available from NetManage Inc..

**PDA.** Physical device address.

**PDP.** Problem determination procedure.

**personal computer system.** IBM Personal System/2 and also the various IBM Personal Computer system units, unless otherwise described.

**Personal Identification Number (PIN) pad.** A pad with twelve keys in a specific arrangement that display alphabetic and numeric characters that may be entered onto a financial transaction terminal. (T) (A)

**physical device address (PDA).** An address or set of addresses that identifies a particular device.

**physical unit (PU).** In SNA, the component that manages and monitors the resources, such as attached links and adjacent link stations, associated with a node, as requested by an SSCP via an SSCP-PU session. An SSCP starts a session with the physical unit to indirectly manage, through the PU, resources of the node such as attached links. This term applies to type 2.0, type 4, and type 5 nodes only.

**PIN.** Personal identification number.

**plaintext.** (1) Nonencrypted data. Synonymous with cleartext. (2) Synonym for clear data.

**PLU.** Primary logical unit.

**PM.** Presentation Manager® (in OS/2).

**pointing device port.** The IBM PS/2 port that allows attachment of various devices including pointing devices.

**port.** (1) An access point for data entry or exit. (2) A connector on a device to which cables for other devices such as display stations and printers are attached. (3) A specific communications end point within a host. A port is identified by a port number.

**Post Telephone and Telegraph Administration (PTT).** An organization, usually a government department, that provides communication common carrier services in countries other than the USA and Canada. Examples of PTTs are the Bundespost in Germany, and the Nippon Telephone and Telegraph Public Corporation in Japan.

**PPC.** Program to program communications.

**Presentation Manager.** A component of OS/2 that provides a complete graphics-based user interface, with pull-down windows, action bars, and layered menus.

**primary logical unit (PLU).** (1) In SNA, the logical unit (LU) that contains the primary half-session for a particular LU—LU session. (2) Contrast with secondary logical unit (SLU). (3) See also logical unit (LU).

**problem determination procedure (PDP).** A prescribed sequence of steps taken to identify the source of a problem.

**process.** (1) A unique, finite course of events defined by its purpose or by its effect, achieved under defined conditions. (2) Any operation or combination of operations on data. (3) A function being performed or waiting to be performed. (4) A program in operation.

**processor.** (1) In a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic and logic unit. (T) (2) The functional unit that interprets and processes instructions.

**profile.** (1) In computer security, a description of the characteristics of an entity to which access is controlled. (2) Data that describes the significant characteristics of a user, a group of users, or one or more computer resources.

**program.** A sequence of instructions suitable for processing by a computer. Processing may include the use of an assembler, a compiler, an interpreter, or a translator to prepare the program for execution, and also to execute it. (I)

**program control block (PCB).** LANDP family shared-file server pointer related to a specific DBD.

**Program temporary fix (PTF).** A temporary solution or by-pass of a problem diagnosed by IBM as resulting from a defect in a current unaltered release of the program.

**protocol.** In SNA, the meanings of and the sequencing rules for requests and responses used for managing the network, transferring data, and synchronizing the states of network components.

**PS/2.** Personal System/2.

**PTF.** Program temporary fix.

**PTT.** Post Telephone and Telegraph Administration.

**PU.** Physical unit.

## Q

**QLLC.** Qualified logical link control.

**qualified logical link control (QLLC).** An X.25 protocol that allows the transfer of data link control information between two adjoining systems network architecture (SNA) nodes that are connected through an X.25 packet-switching data network. The QLLC provides the qualifier "Q" bit in X.25 data packets to identify packets that carry logical link protocol information.

**query.** (1) A request for information from a file relying on specific conditions. (2) In the AS/400 system, the query management object that is used to define queries against relational data.

**quiescing.** The process of bringing a device or a system to a stop by rejection of new requests for work. (A)

## R

**RAM.** Random access memory. (A)

**random access memory (RAM).** A storage device where data can be written and read.

**RC.** Return code.

**RCMS.** Remote change management services.

**RDBMS.** Relational database management system. A generic name for any relational database system such as DB2.

**re-synchronization.** Restarting the transmission of a function at the point where it was interrupted.

**read-only memory (ROM).** (1) A storage device where data, under normal conditions, can only be read. (T) (2) See also read-only storage (ROS).

**read-only storage (ROS).** (1) A storage device whose contents cannot be modified, except by a particular user, or when operating under particular conditions. (2) See also read-only memory (ROM).

**record.** (1) In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. In some programming languages, records are called structures. (I) (2) A set of data treated as a unit. (T)

(3) A set of one or more related data items grouped for processing.

**remote attachment.** A method of connecting two devices over a telecommunication line.

**remote initial program load (remote IPL).** A feature that permits a computer to receive its initial program from another computer, rather than from its own internal disk or diskette storage.

**remote method invocation.** A specific instance of the more general term RPC (remote procedure call). Remote method invocation (RMI) allows objects to be distributed over a network, that is, a Java program running on one computer can call the methods of an object running on another computer. RMI and java.net are the only 100% pure Java APIs for controlling Java objects in remote systems.

**remote procedure call (RPC).** A facility that a client uses to request the execution of a procedure call from a server. This facility includes a library of procedures and an external data representation.

**REMS.** Reader/encoder magnetic stripe.

**request/response header (RH).** In systems network architecture (SNA), control information preceding a request/response unit (RU) that specifies the type of RU and contains control information associated with the RU.

**request/response unit (RU).** In systems network architecture (SNA), a generic term for a request unit or a response unit.

**resource.** (1) Any of the data processing system elements needed to perform required operations, including storage, input/output units, one or more processing units, data, files, and programs. (T) (2) See also network resource.

**retry.** To resend data a prescribed number of times or until the data is received correctly.

**return code (RC).** (1) A code used to influence the execution of succeeding instructions. (A) (2) A value returned to a program to indicate the results of an operation requested by that program.

**RH.** Request/response header.

**roll back.** To remove changes that were made to database files under commitment control since the last commitment boundary.

**RMI.** Remote Method Invocation.

**rollback.** (1) A programmed return to a prior checkpoint. (A) (2) The process of restoring data changed by an application program or user to the state of its last commitment boundary. (3) In SQL, the process of restoring data changed by an application program or user to the state of its last commit point.

**ROM.** Read-only memory. (A)

**ROS.** Read-only storage.

**router.** (1) A computer that determines the path of network traffic flow. The path selection is made from several paths based on information obtained from specific protocols, algorithms that attempt to identify the shortest or best path, and other criteria such as metrics or protocol-specific destination addresses. (2) An attaching device that connects two LAN segments, which use similar or different architectures, at the reference model network layer. Contrast with bridge, gateway. (3) In OSI terminology, a function that determines a path by which an entity can be reached.

**RPC.** Remote procedure call.

**RTR.** Ready to Receive.

**RU.** Request/response unit.

## S

**SAM.** Service availability manager.

**SAP.** Service access point.

**SBCS.** Single-byte character set.

**scan code.** A code generated by a keyboard.

**SCS.** Systems network architecture character string.

**SDLC.** Synchronous data link control.

**secondary logical unit (SLU).** (1) In systems network architecture (SNA), the logical unit (LU) that contains the secondary half-session for a particular LU-LU session. (2) Contrast with primary logical unit (PLU). (3) See also logical unit (LU).

**SEQ.** Sequential file.

**sequential access.** (1) The capability to enter data into a storage device or a data medium in the same

sequence as the data is ordered, or to obtain data in the same order as it has been entered. (T) (2) An access method in which records are read from, written to, or removed from a file based on the logical order of the records in the file. (3) Contrast with direct access.

**serial port.** (1) On personal computer systems, a port used to attach devices such as display devices, letter-quality printers, modems, plotters, and pointing devices such as light pens and mice; it transmits data one bit at a time. (2) See also parallel port.

**serialization.** Turning an object into a stream and back again.

**server.** (1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. (T) (2) In LANDP, a functional area that provides functions to LANDP workstations in a LANDP workgroup. (3) The computer that hosts the Web page that contains an applet. The .class files that make up the applet, and the HTML files that reference the applet reside on the server. When someone on the Internet connects to a web page that contains an applet, the server delivers the .class files over the Internet to the client that made the request. The server is also known as the originating host. (4) See also client, client workstation, and user. (5) In LANDP, a function provided by a server.

**service access point (SAP).** A logical point made available by a token-ring adapter where information can be received and transmitted.

**service availability manager (SAM).** Facility used by the shared-file server to provide fault-tolerant data access in an XLR environment.

**servlet.** Server-side program that executes on and adds function to a Web server. Java servlets allow for the creation of complicated, high-performance, cross-platform Web applications. They are highly extensible and flexible, making it easy to expand from client or single-server applications to multi-tier applications.

**session.** (1) In systems network architecture (SNA), a logical connection between two network addressable units (NAU) that can be started, tailored to provide various protocols, and deactivated, as requested. (2) The time during which programs or devices can communicate with each other.

**single-byte character set (SBCS).** (1) A character set in which each character is represented by a one-byte

code. (2) Contrast with double-byte character set (DBCS).

**SLU.** Secondary logical unit.

**SNA.** Systems network architecture.

**SNUF.** Systems network architecture up-line facility.

**socket.** (1) An end-point for communication between processes or applications. (2) A pair consisting of TCP port and IP address.

**SOM.** Start-of-message code.

**SPC, spc.** Specification file.

**specification file (SPC, spc).** In LANDP, a file with the file extension .SPC. This file can be edited. It contains information for customization purposes.

**SQL.** Structured query language.

**SSCP.** System services control point.

**start-of-message code (SOM).** A character or group of characters transmitted by the polled terminal and indicating to other stations on the line that what follows are addresses of stations to receive the answering message.

**storage.** A functional unit into which data can be placed, where it can be retained, and from which it can be retrieved. (T)

**stream.** A continuous sequence of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format.

**structured query language (SQL).** An established set of statements used to manage information stored in a database. By using these statements, users can add, delete, or update information in a table, request information through a query, and display the results in a report.

**subdirectory.** A directory contained within another directory in a file system hierarchy.

**synchronous.** (1) About two or more processes that depend on the occurrence of a specific event such as common signal timing. (2) Occurring with a regular or predictable time relationship. (3) See also asynchronous.

**synchronous data link control (SDLC).** A discipline conforming to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute (ANSI) and High-level Data Link Control (HDLC) of the International Organization for Standardization, for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges may be duplex or half-duplex over switched or not-switched links. The configuration of the link connection may be point-to-point, multi-point, or loop. (I)

**system diskette.** (1) The diskette, either real or virtual, that contains your control program. (2) In personal computer systems, the diskette on which you have the operating system.

**system distribution manager.** A system that contains the files and programs required for product installation, and initiates or manages the installation process.

**system services control point (SSCP).** In systems network architecture (SNA), the focal point within an SNA network for managing the configuration, coordinating network operator and problem determination requests, and providing directory support and other session services for end users of the network.

**systems network architecture (SNA).** The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

**systems network architecture character string (SCS).** In systems network architecture (SNA), a character string composed of EBCDIC controls, optionally intermixed with end-user data, that is carried within a request/response unit (RU).

**systems network architecture network (SNA network).** In systems network architecture (SNA), the part of an application program network that conforms to the formats and protocols of SNA. It allows reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network addressable units (NAU), boundary function components, and the path control network.

**systems network architecture up-line facility (SNUF).** The communications support that allows an AS/400 system to communicate with CICS/VS and IMS/VS application programs on a host computer.

# T

**takeover.** In an XLR environment, the process by which a backup server assumes the role of the (failed) active. This involves backing out incomplete transactions, rebuilding indexes, and informing SAM of the new active workstation.

**TCP/IP.** Transmission Control Protocol/Internet Protocol.

**terminal status line.** Synonym for operator information area (OIA).

**TH.** Transmission header.

**thin client.** A client workstation that loads its operating system environment and applications across a network from a server. The degree of local processing power in a thin client can vary considerably depending on the implementation of the thin client concept.

The term thin client usually refers to a system that runs on a resource-constrained machine or that runs on a small operating system. This clients do not require require local system administration, and they execute Java applications delivered over the network.

**Time Sharing Option (TSO).** An operating system option; for the System/370 system, the option provides interactive time sharing from remote terminals.

**token-ring network.** (1) A ring network that allows unidirectional data transmission between data stations by a token passing procedure, so that the transmitted data returns to the transmitting station. (T) (2) A network that uses a ring topology, where tokens are passed in a circuit from node to node. A node that is ready to send can capture the token and insert data for transmission.

**trace.** (1) A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed. (A) (2) The process of recording the sequence in which the statements in a program are executed and, optionally, the values of the program variables used in the statements. (3) To record a series of events as they occur. (4) For data links, a record of the frames and bytes transmitted or received.

**trace file.** A file that contains a record of events that occur in a system.

**trace function.** A function used for problem determination.

**trace log.** A file in which trace events are recorded.

**trace program.** A computer program that performs a check on another computer program by exhibiting the sequence in which the instructions are executed and, usually, the results of executing the instructions. (I) (A)

**trace routine.** A routine that provides an historical record of specified events in the execution of a computer program. (A)

**transaction.** An exchange between a workstation and another device that accomplishes a particular action or result.

**translation.** Conversion of a code or codes to another code or codes according to a set of specifications.

**transmission.** The sending of data from one place for reception elsewhere. (A)

## Notes:

1. Transmission implies only the sending of data; the data may or may not be received.
2. The term transmit is used to describe the sending of data in telecommunication operations. The terms move and transfer are used to describe movement of data in data processing operations.

**transmission control (TC) layer.** The layer within a half-session or session connector that synchronizes and paces session-level data traffic, checks session sequence numbers of requests, and enciphers and deciphers end-user data.

**Transmission Control Protocol (TCP).** A communications protocol used in the Internet and in any network that follows the US Department of Defense standards for inter-network protocol. TCP provides a reliable host-to-host protocol between hosts in packet-switched communications networks and in interconnected systems of such networks. It assumes that the Internet protocol is the underlying protocol.

**Transmission Control Protocol/Internet Protocol (TCP/IP).** A set of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

**transmission header (TH).** In systems network architecture (SNA), control information, optionally followed by a basic information unit (BIU) or a BIU segment, that is created and used by path control to



route message units and to control their flow within the network.

**transmission services (TS) profile.** In systems network architecture (SNA), a specification in a session activation request (and, optionally in the responses) of transmission control (TC) protocols, such as session-level pacing and the usage of session-level requests, to be supported by a particular session. Each defined TS profile is identified by a number.

**trap.** An unprogrammed conditional jump to a specified address that is automatically activated by hardware. A recording is made of the location from which the jump occurred.

**TRDLC.** Token-ring data link control.

**TS.** Transmission services.

**TSO.** Time Sharing Option.

## U

**UDP.** User Datagram Protocol.

**UNBIND.** (1) In systems network architecture (SNA), a request to deactivate a session between two logical units (LU). (2) Contrast with BIND.

**user.** (1) A function that uses the services provided by a server. A host can be a user and a server at the same time. (2) Any person or any thing that may issue or receive commands and messages to or from the information processing system. (T) (3) Any person who requires the services of a computing system. (4) See also client, client/server, client workstation, and server.

**User Datagram Protocol (UDP).** In TCP/IP, a packet-level protocol built directly on the Internet protocol layer. UDP is used for application-to-application programs between TCP/IP host systems.

**user profile.** In computer security, a description of a user that includes such information as user identification (ID), user name, password, access authority, and other attributes obtained at log-on.

**user-written server.** In LANDP, a server not supplied with a LANDP program, but developed by the customer.

**utility program.** (1) A computer program which supports computer processes; for example, a sort program. (T) (2) A program designed to perform an everyday task such as copying data from one storage device to another. (A)

## V

**validation.** The checking of data for correctness, or compliance with applicable standards, rules, and conventions. (A)

**VDM.** Virtual DOS machine.

**vector.** A set of keyword=parameter statements that define configuration items. These items can correspond to both model and real configurations.

**verify.** To determine whether a transcription of data or other operation has been accomplished accurately. (A)

**VFS.** Virtual file system.

**virtual DOS machine (VDM).** A functional simulation of a machine running under DOS.

**virtual file system (VFS).** A remote file system that has been mounted so that it is accessible to the local user.

**virtual machine (VM).** A virtual data processing system that seems to be at the exclusive disposal of a particular user, but whose functions are accomplished by sharing the resources of a real data processing system. (T)

### Virtual Telecommunications Access Method

**(VTAM).** A set of programs that maintain control of the communication between terminals and application programs running under Disk Operating System/Virtual Storage (DOS/VS), OS/VS1, and OS/VS2 operating systems.

**VisualGen®.** A high-level object-oriented programming language.

**VM/CMS.** Virtual machine/conversational monitor system.

**VTAM.** Virtual Telecommunications Access Method.

# W

**WAN.** Wide area network.

**wide area network (WAN).** A network that provides communication services to a geographical area larger than that served by a local area network.

**WebSphere™.** A comprehensive solution to build, deploy, and manage e-business Web sites. WebSphere is the cornerstone of IBM's overall Web strategy. The Websphere product line provides companies with an open, standards-based, Web server deployment platform, together with Web site development and management and management tools to help accelerate the process of moving to e-business.

**window.** A division of a screen where one of several programs being run concurrently can display information.

**workgroup.** In LANDP, the logical connection of LANDP for DOS, LANDP for OS/2, LANDP for Windows NT, and LANDP for AIX workstations through the LANDP client/server mechanism, which is available with each LANDP program.

**Workspace On-Demand.** (1) A set of management utilities that enables OS/2 Warp Server to remotely load a thin client operating system, known as Workspace On-Demand client, into a client workstation across a LAN. (2) The client workstation component of Workspace On-Demand, which is loaded into a client workstation from a server machine running OS/2 Warp Server and Workspace On-Demand Server.

**Workspace On-Demand Server.** A server, running OS/2 Warp Server and Workspace On-Demand, that is used to boot client workstations.

**workstation.** (1) A functional unit at which a user works. (2) In LANDP, personal computer system in a local area network (LAN).

**wrapper.** A language binding.

# X

**X.25.** A CCITT recommendation that defines the physical level (physical layer), link level (data link layer), and packet level (network layer) of the open systems inter-connection (OSI) reference model. An X.25 network is an interface between data terminal equipment (DTE) and data circuit-terminating equipment (DCE) operating in the packet mode, and connected to public data networks by dedicated circuits. X.25 networks use the connection-mode network service.

**X.25 NCP Packet Switching Interface.** An IBM-licensed program that allows systems network architecture (SNA) users to communicate over packet switched data networks that have interfaces complying with Recommendation X.25 (Geneva 1980) of the International Telegraph and Telephone Consultative Committee (CCITT). It allows SNA programs to communicate with SNA equipment or with non-SNA equipment over such networks.

**XID.** Exchange identification.

**XLR.** External logging replicator.

**XOR.** Logical operation exclusive-or.

# Numerics

**4700 Processor.** IBM Finance Communication System 4701 Controller Model 3 and IBM 4702 Branch Automation Processor, unless otherwise described.

---

## Bibliography

This bibliography includes publications cited in this book and other publications on related topics. Where a shortened title is used in the text, the short title is listed after the full title.

---

### IBM LANDP Family

*IBM LANDP Family: Introduction and Planning.*  
GC34-5529.

**Short title:** *LANDP Introduction and Planning.*

*IBM LANDP Family: Installation and Customization.*  
GC34-5530.

**Short title:** *LANDP Installation and Customization.*

*IBM LANDP Family: Programming Guide.*  
SC34-5781.

**Short title:** *LANDP Programming Guide.*

*IBM LANDP Family: Programming Reference.*  
SC34-5531.

**Short title:** *LANDP Programming Reference.*

*IBM LANDP Family: Servers and System Management.* SC34-5532.

**Short title:** *LANDP Servers and System Management.*

*IBM LANDP Family: Problem Determination.*  
GC34-5533.

**Short title:** *LANDP Problem Determination.*

---

### IBM Financial Branch System Services Licensed Programs

*IBM FBSS Licensed Programs Family General Information.* GC19-5172.

*IBM FBSS Licensed Programs Family Installation and Customization.* SC19-5173.

*IBM FBSS Licensed Programs Family Program Description.* SC19-5176.

*IBM FBSS Licensed Programs Family Version 2 Programmer's Reference.* GA19-5450.

*IBM FBSS Licensed Programs Family Version 2 Application Programming.* SC19-5174.

---

### IBM Financial Branch System Integrator Licensed Programs

*Financial Branch System Integrator and Financial Branch System Integrator/2 General Information.*  
GC19-5187.

*Financial Branch System Integrator Programmer's Reference Manual.* GA19-5452.

*Financial Branch System Integrator/2 Programmer's Reference Manual.* SC19-5188.

---

### IBM Transaction Security System

*IBM Transaction Security System: General Information Manual and Planning Guide.*  
GA34-2137.

*IBM Transaction Security System: Programming Guide and Reference.* SC31-2934.

---

### Banking Self-Service

*IBM 4721 Self-Service Document Printer Programmer's Reference.* GA19-5342.

*IBM 4731/38/39 Personal Banking Machines P-Models Software Customization and Programming Reference.* GA19-5462.

*IBM 4733 Teller Assist Unit Programmer's Reference.* GA19-5425.

*IBM 4737 Self-Service Transaction Station Programmer's Reference.* GA19-5408.

*IBM Financial Application Development Toolkit Version 2 Program Description and Operation.*  
SB11-8461.

---

## IBM workstations

*PC DOS 7 Technical Update.* GG24-4459.

*PC DOS 7 User Guide.* S83G-9260.

*PC DOS 7 Command Reference.* S83G-9309.

*PC DOS 7 Keyboard and Code Pages.* S83G-9310.

*IBM TCP/IP Version 2.1.1 for DOS: Installation and Administration.* SC31-7047.

*IBM TCP/IP Version 2.1.1 for DOS: User's Guide.* SC31-7045.

*IBM TCP/IP Version 2.1.1 for DOS: Programmer's Reference.* SC31-7046.

*OS/2 Warp Version 4 Up and Running!.* S84H-3098.

*OS/2 Warp Server for e-Business.* SG24-5393.

*OS/2 Warp, PM Programming Reference Vol I.* G25H-7190.

*OS/2 Warp PM Programming Reference Vol II.* G25H-7191.

*OS/2 2.0 Application Design Guide.* S10G-6260.

*OS/2 2.0 Virtual Device Driver Reference.* S10G-6310.

*DB2/2 Guide.* S62G-3663.

*OS/2 LAN Server Network Administration Reference Volume 1: Planning, Installation and Configuration.* S10H-9680.

*OS/2 LAN Server Network Administrator Reference Volume 2: Performance Tuning.* S10H-9681.

*OS/2 LAN Server Network Administrator Reference Volume 3: Network Administrator Tasks.* S10H-9682.

*IBM Systems Application Architecture Common Programming Interface Dialog Reference.* SC26-4356.

*IBM Systems Application Architecture Common Programming Interface Presentation Reference.* SC26-4359.

*IBM OS/2 Programming Tools and Information V1.3 Programming Guide.* S91F-9259.

*TCP/IP for OS/2 Warp Programming Reference,* SC31-8407.

*IBM Network SignON Coordinator/2 Getting Started.* S96F-8629.

---

## IBM RISC System/6000®

*AIX SNA Server/6000: User's Guide.* SC31-7002.

*AIX SNA Server/6000: Transaction Program Reference.* SC31-7003.

*AIX SNA Server/6000: Configuration Reference.* SC31-7014.

*IBM AIX V3.2 Commands Reference for RISC System/6000, Volume 1.* GC23-2376.

*IBM AIX V3.2 Commands Reference for RISC System/6000, Volume 2.* GC23-2366.

*IBM AIX V3.2 Commands Reference for RISC System/6000, Volume 3.* GC23-2367.

*IBM AIX V3.2 Commands Reference for RISC/6000, Volume 4.* GC23-2393.

*SNA Transaction Programmer's Reference for LU Type 6.2.* GC30-3084.

*Assembler Language Reference for IBM AIX Version 3 for RISC System/6000.* SC23-2197.

*General Programming Concepts for IBM RISC System/6000.* SC23-2205.

*IBM AIX V3.2 User Interface Programming Concepts, Volume 1.* SC23-2404.

*IBM AIX NetBIOS on Token-Ring/6000.* SC23-2336.

*Managing Application Software with the Resource Management System.* SC33-9110.

*IBM AIX Windows Programming Guide.* GG24-3382.

*Writing a Device Driver for IBM AIX V4.1.* SC23-2593.

*IBM AIX Calls and Subroutines Reference for RISC System/6000.* SC23-2198.

*IBM AIX Communications Programming Concepts for RISC System/6000.* SC23-2206.

*IBM AIX for RISC System/6000 Performance Monitoring and Tuning Guide.* SC23-2365.

---

<sup>1</sup> This information is available in multiple languages. Contact your IBM representative for ordering information.

*IBM AIX Files Reference for RISC System/6000.* SC23-2512.

*AIX V3.2 Topic Index and Glossary.* GC23-2201.

*IBM RISC System/6000 Planning for Your System Installation V3.2.* GC23-2407.

*IBM RISC System/6000 System Overview V3.2.* GC23-2406.

*IBM RISC System/6000 CD-ROM Hypertext Information Base Library.* SC23-2163.

*AIX V3.2 System Management Guide: Operating System and Devices.* GC23-2486.

*AIX 4777/4778 Programming Guide.* SA34-2358.

---

## IBM Local Area Network

*IBM Token-Ring Network: Introduction and Planning Guide.* GA27-3677.

*IBM Token-Ring Network: Problem Determination Guide.* SX27-3710.

*Local Area Network: Administrator's Guide.* GA27-3748.

*IBM PC Network: Technical Reference.<sup>2</sup>*

*IBM Personal Computer LAN Support Program.<sup>2</sup>*

*IBM Personal Computer Baseband and Broadband.<sup>2</sup>*

*IBM Cabling System Planning and Installation Guide.* GA27-3361.

*Using the IBM Cabling System with Communication Products.* GA27-3620.

*IBM Token-Ring Network Architecture Reference.* SC30-3374.

*IBM Local Area Network Technical Reference.* SC30-3587.

*IBM Local Area Network Support Program User's Guide.* SC21-8288.

---

## IBM 3270

*IBM 3270 Personal Computer Control Program Programming Guide.* SC23-0165.

*IBM 3270 Information Display System Character Set Reference.* GA27-2837.

*IBM PC 3270 Emulation Program, Entry Level V2.0 Programmer's Guide.* S91F-8583.

*IBM 3270 PC High Level Language API Programming Reference.* SC23-2473.

*Personal Communications Version 4.3 Emulator Programming.* SC31-8478.

---

## Wide Area Communications

*SNA Primary Custom Feature Description.* GC31-2509.

*Advanced Function for Communications: System Summary.* GA27-3099.

*System Network Architecture (SNA) Technical Overview.* GC30-3073.

*System Network Architecture (SNA) Format and Protocol Reference Manual.* SC30-3112.

*System Network Architecture (SNA) Formats.* GA27-3136.

*System Network Architecture (SNA) Format and Protocol Reference Manual: Management Services.* SC30-3346.

*System Network Architecture (SNA) Sessions between Logical Units.* GC20-1868.

*CCITT X.25 Recommendations, Interface between Data Terminal Equipment (DTE) and Data Circuit Terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks.* Vol. VIII. Fascicle VIII.5. This document is useful when writing X.25 applications.

*RT PC X.25 Communication Support User's Guide.* SC33-0630.

*X.25 Interface for Attaching SNA Nodes to Packet-Switched Data Network General Information Manual.* GA27-3345.

*RT PC X.25 Communications Support Programmer's Reference.* SC33-0631.

*IBM Cryptographic Subsystem Concepts and Facilities.* GC22-9063.

*IBM X.25 Co-Processor Support Program User's Guide.* X07F-8915.

*IBM X.25 Co-Processor Support Program Programmer's Reference.* X07F-8916.

---

<sup>2</sup> This publication is shipped with the product. Contact your IBM Representative for ordering information.

*IBM X.25 Interface Co-Processor/2 Technical Reference.* S16F-1879.

*SNA Advanced Peer-to-Peer Networking Dependent LU Requester Architecture Reference.* SV40-1010.

*Multiprotocol Transport Networking (MPTN) Architecture: Formats.* GC31-7074.

*Multiprotocol Transport Networking (MPTN) Architecture: Technical Overview.* GC31-7073.

*Telnet Protocol Specification, STD 8, RFC 854, USC/Information Sciences Institute. J. Postel and J. Reynolds .*

To view this book, use the keyword **RFC 854** with an internet search engine.  
Printed copies of RFCs are available for a fee from:

SRI International, Room EJ291  
333 Ravenswood Avenue  
Menlo Park, CA 94025  
(415) 859-3695  
(415) 859-6387  
FAX (415) 859-6028

---

## IBM NetView

*NetView Distribution Manager: General Information V1.6.* GH19-6792.

*NetView Distribution Manager: Planning.* SH19-6589.

*NetView Distribution Manager Release 6: Installation and Customization.* SH19-6794.

*NetView Distribution Manager: Operation.* SH19-6592.

*NetView Distribution Manager: User's Guide V1.6.* SH19-6795.

*NetView Distribution Manager: Diagnosis R5.* LY19-6374.

*NetView Distribution Manager: Messages and Codes V1.6.* SH19-6798.

---

## IBM Financial I/O Devices

*IBM 4009 Operator's manual.* GA19-5650.

*IBM 4009 Service manual/Parts catalogue.* SY19-6392.

*IBM 4009 Quick Reference Card.* GX11-6316.

*IBM 4009 Customer Setup.* GA19-5651.

*IBM 4009 Safety Instructions.* GA19-5651.

*IBM 4009 Product and Programming Description (PPD) DOS.* SH19-4015.

*IBM 4009 Product and Programming Description (PPD) OS/2.* SH19-4038.

*IBM 4700 Finance Communication System Summary.* GC31-2016.

*IBM 4700 Financial I/O Planning Guide.* GC31-3762.

*IBM 4700 Financial I/O Devices Programming Guide.* GC31-3770.

*IBM 4700 Financial I/O Devices Programming Guide for OS/2.* GC31-2661.

*IBM 4700 Finance Communication System, Controller Programming Library, Volume 5, Cryptographic Programming.* GC31-2070.

*IBM 4700 Financial I/O Devices Operating Guide.* SC31-3763.

*IBM 4712 Transaction Printer Models 1, 2, and 3 Reference Card.* SC31-3765.

*IBM 4722 Document Printer Model 3 Programming Addendum.* GC31-2928.

*IBM 4722 Document Printer Models 1, 2, and 3 Reference Card.* SC31-3767.

*IBM 4748 Document Printer Programming Guide.* SA34-2090.

*IBM 4748 Document Printer Operating Guide.* SA34-2068.

*IBM 4748 Document Printer Service Guide.* SA34-2091.

*IBM 4770 Ink Jet Transaction Printer Product Profile.* G571-0276.

*IBM 4772 Universal Financial Printer Model 1 Programming Guide.* SA34-2199.

*IBM 4772 Universal Financial Printer Model 1 and 2 Installation and Operating Guide.* GA34-2192.

*IBM 4772 Universal Financial Printer Model 1 and 2 Reference Card.* GX31-2077.

*IBM 4772 Universal Financial Printer Model 1 and 2 Service Guide.* SA34-2193.

*IBM 4777 Magnetic Stripe Unit Installation and Operating Guide.* GA34-2189.

*IBM 4777 Magnetic Stripe Unit: Programming Guide for OS/2.* SA34-2194.

*IBM 4777 Magnetic Stripe Unit: Programming Guide for DOS.* SA34-2195.

*IBM 4778 PIN-Pad Magnetic Stripe Reader Installation and Operating Guide.* GA34-2190.

*IBM 4778 PIN-Pad Magnetic Stripe Reader: Programming Guide for OS/2.* SA34-2196.

*IBM 4778 PIN-Pad Magnetic Stripe Reader: Programming Guide for DOS.* SA34-2197.

*IBM 4777 Magnetic Stripe Unit and 4778 PIN-Pad Magnetic Stripe Reader AIX Programming Guide.* SA34-2358.

*IBM 9055-001 Document Printer: Planning and Programming Guide.* SA18-7496.

*IBM 9055-002 Document Printer: Planning and Programming Guide.* SA18-7489.

*IBM 9068 Multi-Purpose Passbook Printer Model D01 Planning and Programming Guide.* SA18-7505.

*IBM 9068 Multi-Purpose Passbook Printer Model S01 Planning and Programming Guide.* SA18-7506.

*IBM 9068-S01 Multi-Purpose Passbook Printer Operating Guide.* SA18-7507.

*IBM 9069 Printer Planning and Programming Guide.* SA18-7525.

*IBM 9069 Operating Guide.* SA18-7524.

---

## Distributed Computing Environment

*AIX DCE Overview.* SC23-2477.

*DCE Administration Guide.* SC23-2475.

*Introduction to DCE.* Prentice Hall Inc.

*DCE User's Guide and Reference.* Prentice Hall Inc.

*DCE Administration Reference.* Prentice Hall Inc.

*DCE Application Development Guide.* Prentice Hall Inc.

*DCE Application Development Reference.* Prentice Hall Inc.

*IBM DCE for OS/2: Application Developer's Guide.* S96F-8506.

---

## Encryption and Decryption

*IBM Cryptographic Subsystem Concepts and Facilities.* GC22-9063.

*IBM 4700 Finance Communication System, Controller Programming Library, Volume 5, Cryptographic Programming.* GC31-2070.

*IBM Transaction Security System Workstation Security Services: Installation and Operating Guide.* SA34-2141.

*IBM Transaction Security System Concepts and Programming Guide: Volume 1, Access Controls and DES Cryptography.* GC31-3937.

---

## IBM VisualAge C++

Product web site:

<http://www.ibm.com/software/ad/vacpp/>

---

## IBM VisualAge Generator

Product web site:

<http://www.ibm.com/software/ad/visgen/>

IBM Redbook:

*VisualAge Generator Version 3.1 System Development Guide.* SG24-4230-02

---

## IBM VisualAge Smalltalk

Product web site:

<http://www.ibm.com/software/ad/smalltalk/>

IBM Redbooks:

*VisualAge for Smalltalk Handbook - Volume 1: Fundamentals.* SG24-4828-00

*VisualAge for Smalltalk Handbook - Volume 2: Features.* SG24-2219-00

---

## Java

IBM Java web site:

<http://www.ibm.com/software/java/>

IBM VisualAge for Java product web site:

<http://www.ibm.com/software/ad/vajava/>

IBM developerworks web site:

<http://www.ibm.com/software/developer/java/>

IBM VisualAge Developers Domain web site:

<http://www.ibm.com/software/vadd/>

IBM Redbooks:

*Programming with VisualAge for Java Version 2.* SG24-5624-00

*Application Development with VisualAge for  
Java Enterprise, SG24-5081-00*

---

## IBM Personal Communications

*Personal Communications AS/400 and 3270 for  
OS/2 Up and Running, SC31-8258.*

*Personal Communications AS/400 and 3270 for  
Windows NT Up and Running, GC31-8314.*

*Personal Communications/3270 Programmer's  
Guide for DOS (Entry Level), S20H-1774.*

*Personal Communications/3270 Programmer's  
Guide for OS/2, S85G-8681.*

*Personal Communications/3270 Reference Guide  
for OS/2, S85G-8721.*

*Personal Communications Version 4.3 for Windows  
98 and Windows NT Reference, Volumes 1 and 2 ,  
SC31-8682, SC31-8680.*

*Personal Communications Windows NT Quick  
Beginnings, GC31-8679.*

---

## IBM Communications Server

*IBM SecureWay Communications Server for OS/2  
Warp Version 6 Quick Beginnings, GC31-8189.*

*IBM Communications Server for Windows NT Quick  
Beginnings, GC31-8424.*

*eIBM Network Communications Server for OS/2  
Guide to AnyNet, GC31-8193, GC31-8320*

---

## Workspace On-Demand

*Workspace On-Demand Road Map Release 2.0,  
SG24-5117 (10/98)*

*Workspace On-Demand Handbook Release 2.0,  
SG24-5117 (10/98)*

*Workspace On-Demand Handbook (Release 1),  
SG24-2028 (12/97)*

*Workspace On-Demand Early Customer  
Experiences, SG24-5107 (10-98)*

*IBM Up and Running! OS/2 Warp Server,  
S25H-8004*

*Workspace On-Demand Administrator's Guide.,, on  
the web at  
[www.ibm.com/software/network/workspace/library](http://www.ibm.com/software/network/workspace/library)*

---

## MQSeries

*MQSeries for Windows NT V5.0 Quick Beginnings,  
GC33-1871-00*

*MQSeries for OS/Warp Quick Beginnings,  
GC33-1868-01*

*MQSeries Planning Guide, GC33-1349-05*

*MQSeries Intercommunication, SC33-1872-00*

*MQSeries Clients, GC33-1632-04*

*MQSeries System Administration, SC33-1873-00*

*MQSeries Command Reference, SC33-1369-08*

*MQSeries Programmable System Management,  
SC33-1482-05*

*MQSeries Messages, GC33-1876-00*

*MQSeries Application Programming Guide,  
SC33-0807-07*

*MQSeries Application Programming Reference,  
SC33-1673-03*

*MQSeries Using C++, SC33-1877-00*



## Index

### Special Characters

- \*\* (process disconnection)** function in all servers 61
- \*\* (workstation disconnection)** function in all servers 61
- && (process connection)** function in all servers 63
- && (workstation connection)** function in all servers 62
- #, use of in server names** 11

### Numerics

- 16-bit programs** 15
- 32-bit programs** 16
- 3270 books** 209

## A

- acting as client, server** 2
- activate and deactivate timers (T0–T8)** supervisor local function
  - used in sample application 173
- AIX to host and PC code page conversion** 117
- AIX, use of native system functions** 2
- alias names** 12
- API (application programming interface)**
  - See application programming
- application programming**
  - 32-bit programs 16
  - alias names 12
  - API link routines 4
    - GETREQ 51
    - GETRPLY 32
    - RMTAREQ 55
    - RMTREQ 28
    - RMTREQ NoWait 29
    - RMTRPLY 54
    - SRVINIT 49
  - asynchronous event notification 36
  - C Set/2 9
  - C/6000 9
  - C++ 87
  - client/server interaction 27
  - COBOL programs under VisualAge 103
  - COBOL/2 9
  - COBOL/6000 9

### application programming (*continued*)

- common API 2, 19
  - invocation 27
  - used by clients 4
  - used by servers 5
- compiling and linking 13
- connectivity programming request block (CPRB)
  - fields required for GETRPLY 33
  - fields required for RMTREQ 28
- dynamic link library 14, 17
- EHC\_GETRPLY\_OPTS structure in calls 34
- EHC\_NOWAIT\_PARM structure in calls 32, 34
- EHC\_RMTREQ\_OPTS structure in calls 31
- event ID 37
- event notification
  - support (asynchronous) 36
  - using posted GUI message 39
- expanded memory in LANDP for DOS 68
- GETREQ 4
- GETRPLY 4, 32
- GETRPLY options control block 34
- hints 35
- include file 16, 17
- including options control blocks 9
- including the CPRB 9
- interrupt handling in LANDP for DOS 68
- invoking the common API 27
- keyboard events 41
- LANDP for AIX server structure 73
- LANDP for DOS Windows 3.1 support 43
- LANDP–DCE client 78
- LANDP–DCE server 82
- link-editing 13
- MASM/2 9
- memory management considerations with LANDP for DOS 66
- Micro Focus COBOL 9, 13
- mono-service servers 11
- mouse events under LANDP for OS/2 and LANDP for Windows NT 41
- multiple threads in LANDP for OS/2 69
- multiple threads in LANDP for Windows NT 71
- multiple-service servers 11
- names of system resources 10
- notation conventions xiv
- Pascal/2 9

# Index

## application programming (continued)

- Pascal/6000 9
  - passing
    - data 10
    - parameters 9
  - polling asynchronous event information 39
  - process connection and disconnection events 42
  - pure 16-bit programs 15
  - receiving event notifications 38
  - reply buffer allocation in LANDP for AIX 75
  - requesting services through the common API 27
  - REXX on LANDP for OS/2 113
  - RMTAREQ 4
  - RMTREQ 4, 28
  - RMTREQ NoWait 4
  - RMTREQ options control block 31
  - RMTRPLY 4
  - router return codes 35
  - sample application program
  - sample application programs
    - LANDP for DOS, OS/2, and AIX 36
  - semaphore events under LANDP for OS/2 and LANDP for Windows NT 42
  - server child support in LANDP for AIX 76
  - server events in LANDP for AIX 76
  - server names 10
  - server return codes 35
  - Smalltalk 92
  - special considerations with LANDP for AIX 73
  - special considerations with LANDP for DOS 66
  - special considerations with LANDP for OS/2 69
  - special considerations with LANDP for Windows NT 71
  - SRVINIT 4
  - SRVINIT, calling in LANDP for AIX 75
  - tutorial 127—180
  - types of events 36
    - system events 40
    - timer events 41
  - VisualAge for COBOL 103
  - VisualAge programs 9
  - waiting for multiple events 38
  - WM used by LANDP for OS/2 servers 70
  - WM used by LANDP for Windows NT servers 72
- ## asynchronous events 64
- event notification or cancellation with RMTRPLY 55
  - notation for codes xv
  - notification 36
  - polling 39
  - wait (WM) supervisor local function 38

## B

### banking self-service books 207

### bibliography 207

- 3270 209
- banking self-service 207
- Communications server 212
- Distributed Computing Environment 211
- encryption and decryption 211
- FBSS 207
- Financial Branch System Integrator 207
- Financial I/O Devices 210
- LANDP 207
- Local Area Network 209
- NetView 210
- Personal Communications 212
- Personal Computer 208
- Personal System/2 208
- RISC System/6000 208
- Transaction Security System 207
- VisualAge C++ 211
- VisualAge Generator 211
- wide area communications 209

### binding

- using naming services 78, 82
- using string bindings 79, 83

### bit positions, convention for xv

### books for LANDP xvi, 207

### building sample applications 178

## C

### C language programs 16, 17

- sample client and server programs 127—147

### C Set/2 programs 9

### C/6000 programs 9

### C++, writing application programs using 87

### client, definition of 1

### client/server mechanism 2

- client/server interaction 27
- communication 2
- data and the API 10
- how clients and servers interact 1
- parameters and the API 9
- routing requests 1
- server acting as client 2
- supervisor local functions 2

### COBOL (VisualAge for OS/2 programs) 9

### COBOL programs under VisualAge

- sample client and server programs 149—169

- COBOL programs, under VisualAge 103
- COBOL/2 programs 9
- COBOL/6000 programs 9
- code page conversion by DCZYSVP 117
- COFF (Common Object File Format) 17
- common API
  - See application programming
- Common Object File Format (COFF) 17
- communication
  - books 209
- communication between clients and servers 1, 2
- Communications Server
  - books 212
- compiling application programs 13
- CONFIG.SYS
- connection functions
  - of process (&&) in all servers 63
  - of workstation (&&) in all servers 62
- connectivity programming request block (CPRB)
  - all fields 5
  - definition of 2
  - fields used and set by clients 28
  - fields used and set by servers 48
  - use with GETREQ 52
  - use with GETRPLY 33
  - use with RMTAREQ 56
  - use with RMTREQ 28
  - use with RMTRPLY 54, 55
- CPRB
  - See connectivity programming request block (CPRB)
- cryptography
  - books 211

## D

- data, passing through the API 10
- DB2 for OS/2 xv
- DB2 Universal Database xv
- DCE 77
  - books 211
  - client structure 81
  - DCE client accessing LANDP server 77
  - LANDP client accessing DCE server with LANDP–DCE interface 77
  - LANDP client accessing DCE server with non-LANDP–DCE interface 77
  - programming interface 77
  - server exporting bindings to naming services 82
  - server structure 85
  - server using string bindings 83

- DCZYSVP application test program 118
- DCZYXSVP, Motif version of DCZYSVP 121
- deactivate and activate timers (T0–T8) supervisor local function
  - used in sample application 173
- decryption and encryption books 211
- definitions of terms 185
- disconnection
  - process (\*\*) function in all servers 61
  - process events 42
  - workstation (\*\*) function in all servers 61
- Distributed Computing Environment
  - See DCE
- DLL (dynamic link library) 14, 17
- DOS box (VDM) 43
- DOS, use of native system functions 2
- dynamic link library (DLL) 14, 17

## E

- EHC\_GETREQ\_OPTS structure in calls 52
- EHC\_GETRPLY\_OPTS structure in calls 34
- EHC\_NOWAIT\_PARM structure in calls 32, 34
- EHC\_RESERVED field in calls 20, 29
- EHC\_RMTREQ\_OPTS structure in calls 31
- EHC\_SRVINIT\_OPTS structure in calls 50
- EHCAGENT.LIB library file 14
- EHCCONN SNA connection program 18
- EHCDEFC.H header file
  - compiling and linking 14
  - include for C programs 16
  - include for C++ programs 17
  - including CPRB 9
  - LANDP for OS/2 mixed 32-bit programs 16
  - LANDP for OS/2 pure 32-bit programs 16
  - migrating from FBSS (DOS) 19
  - migrating from FBSS/2 21
- EHCDEFC.INC include file 19
- EHCDEFCB.CBL copy file
  - compiling and linking 13
  - including CPRB 9
  - migrating from FBSS (DOS) 19
- EHCDEFM.INC include file
  - compiling and linking 13
  - including CPRB 9
  - migrating from FBSS (DOS) 19
- EHCDEF.P.INC include file 9, 14
- EHC.DOS.LIB library file 13, 27
- EHC.DOSQ program for SQL tables 25

# Index

**EHCOSXM.LIB** library file 27  
**EHCIN** initialization program 18  
**EHCMGR1** migration program 25  
**EHCMGR2** migration program 25  
**EHCMGROS** migration program 25  
**EHCOS2.DLL** dynamic link library file 14, 21  
**EHCOS2.LIB** library file  
    compiling and linking 13  
    LANDP for OS/2 DLL 14  
    use with FBSS/2 15, 27  
**EHCOS216.LIB** library file  
    compiling and linking 13  
    LANDP for OS/2 DLL 14  
    use with LANDP for OS/2 27  
**EHCOS232.LIB** library file  
    compiling and linking 13  
    LANDP for OS/2 DLL 14  
    LANDP for OS/2 mixed 32-bit programs 16  
    LANDP for OS/2 pure 32-bit programs 16  
    migrating from FBSS/2 21  
    use with LANDP for OS/2 27  
**EHCOS2Q** program for SQL tables 25  
**EHCREL** SNA release program 18  
**EHCSVPUE** exit server 124  
**EHCUSER.CFG** file for server-to-server calls 65  
**EHCWINNT** library file (LANDP for Windows NT) 27  
**EHCWINNT.COF** library file (LANDP for Windows NT) 27  
**EHCWINNT.DLL** dynamic link library file 17  
**encryption and decryption**  
    books 211  
**end of service (ES)**  
    function in all servers 57  
**ES (end of service)** function in all servers 57  
    used in sample application 155  
**event codes, notation for** xv  
**event notification** 36  
    *See also* asynchronous events  
    event codes xv  
    event ID 37  
    multiple events 38  
    polling asynchronous events 39  
    posted GUI messages 39  
    receiving event notifications 38  
    types of events 36  
**event types, system and server** 36  
**event-driven applications** 64  
**events, asynchronous**  
    *See* asynchronous events

## examples

asynchronous event codes xv  
asynchronous events and the PIN pad server 64  
DCE client 81  
DCE server 85  
DOS  
    sample client program (COBOL) 170—180  
function codes xv  
LANDP for AIX server 75  
LANDP for AIX service names 75  
listed and annotated 180  
OS/2  
    LANDP application (COBOL) 149—169  
    sample client program (COBOL) 170—180  
PIN pad server using asynchronous events 64  
RequestFromLandp and LandpServer class use 91  
return codes xv  
sample application programs 127  
tutorial 180  
Windows NT  
    LANDP application (C) 127—147  
    LANDP application (COBOL) 149—169  
**exit to idle status (EX) function in FBSS** 45  
**expanded memory (LANDP for DOS)** 68

## F

### FBSS

books 207  
**function (in a server)** 1  
**function codes** 5  
    notation for xv  
    notation for operating environment xiv

## G

**GETREQ** call 4, 51  
    used in sample application 167  
**GETRPLY** call 4, 32  
    used in sample application 159  
**glossary** 185  
**GUI** programs, VisualAge for COBOL 103

## H

**host to AIX code page conversion** 117

**I****I/O books** 210**II (inquire information)** supervisor local function  
used in sample application 163**import libraries** 13**IN (initialize)** supervisor local function  
used in sample application 163, 173**IN (server recognition)** function in all servers 58**include files**EHCDEF.C for C and C++  
needed by LANDP applications 13  
OS/2 16  
Windows NT 17**including the CPRB and options control block  
structures** 9**information about IBM products** 207**initialize (IN)** supervisor local function  
used in sample application 163, 173**interrupt handling** 68**invoking the common API** 27**J****Java websites and redbooks** 211**Java, LANDP support** 95—102client development 96  
exception handling 100  
LANDP Version 4 classes supported 95  
multiple client in a JVM 97  
VisualAge for Java 95  
writing applets to access LANDP 100  
writing LANDP servers 101  
writing servlets 100**K****keyboard**

events 41

**L****LAN (local area network)** 1**LAN books** 209**LANDP common API**

See application programming

**LANDP family books** xvi**LANDP for AIX programming** 73**LANDP for DOS programming** 66**LANDP for DOS Windows 3.1 support** 43**LANDP for OS/2 programming** 14, 17

REXX 113

REXX application programming interface  
(OS/2) 115

writing servers 69

**LANDP for Windows NT programming**

writing servers 71

**LANDP workgroup** 1**LANDP-DCE application programming interface**

LANDP-DCE client

binding using naming services 78

binding using string bindings 79

DCE client structure 81

obtaining a LANDP context 80

obtaining binding handle for LANDP services 78

releasing a LANDP context 81

requesting LANDP services 80

LANDP-DCE server

binding 82

DCE server structure 85

exporting bindings to naming services 82

providing LANDP context 83

releasing LANDP context 84

services to clients 84

using string bindings 83

landpdce.idl and landpdce.acf files 78

**landpdce.idl and landpdce.acf files** 78**LandpRequest class** 87, 92**LandpServer class** 90**LIBDCZY.A library file** 27**link libraries** 13**link routines** 4**link-editing application programs** 13**local area network (LAN)** 1**local servers** 4**M****MASM/2 programs** 9**memory management considerations (LANDP for  
DOS)** 66**Micro Focus COBOL programs**

call RMTREQ 28

compiling and linking 13

including CPRB 9

using XM interface 27

**migrating applications** 18

16-bit programs to 32-bit mode 21

clients to common API 19

# Index

## migrating applications *(continued)*

- FBSS (DOS) clients to LANDP for OS/2 21
- FBSS (DOS) clients to LANDP for Windows NT 22
- FBSS (DOS) servers to LANDP for OS/2 22
- FBSS (DOS) servers to LANDP for Windows NT 23
- FBSS (DOS) to LANDP for OS/2 19
- FBSS (DOS) to LANDP for Windows NT 19
- from shared-file server to LANDP for OS/2 query server 24
- LANDP for DOS clients to LANDP for OS/2 21
- LANDP for DOS clients to LANDP for Windows NT 22
- LANDP for DOS servers to LANDP for OS/2 22
- LANDP for DOS servers to LANDP for Windows NT 23
- LANDP for DOS to LANDP for OS/2 19
- LANDP for DOS to LANDP for Windows NT 19
- LANDP for OS/2 clients to LANDP for Windows NT 25
- LANDP for OS/2 servers to LANDP for Windows NT 25
- to this release of LANDP 18
- user servers to common API 19, 20

## mixed 32-bit programs 16

## mono-service servers 11

## mouse events under LANDP for OS/2 and LANDP for Windows NT 41

## moving services to LANDP for OS/2 19

## moving services to LANDP for Windows NT 19

## multiple events 38

## multiple threads

- LANDP for OS/2 69

- LANDP for Windows NT 71

## multiple-processing facilities of AIX 74

## multiple-service servers 11

## N

## names of system resources 10

## NetView books 210

## non-GUI programs, VisualAge for COBOL for 103

## notation conventions xiv

## O

## Object Module Format (OMF) 17

## object-oriented programming

- C++

- LandpRequest class 87

- LandpServer class 90

- RequestFromLandp class 89

## object-oriented programming *(continued)*

- example of RequestFromLandp and LandpServer class use 91

- LandpRequest class 87, 92

- LandpServer class 90

- RequestFromLandp class 89

- Smalltalk

- LandpRequest class 92

## OMF (Object Module Format) 17

## operating environment, notation for xiv

## operator panel functions in financial printer server

## options control blocks 9

## OS/2, COBOL programs under VisualAge 103

## OS/2, use of native system functions 2

## P

## parallel printer port functions in printer manager server

## parameters, passing through the API 9

## Pascal/2 programs 9

## Pascal/6000 programs 9

## PC to AIX code page conversion 117

## Personal Communications books 212

## personal computer books 208

## polling asynchronous event information 39

## porting applications

- See migrating applications

## posted GUI messages 39

## posting events supervisor local functions

## PPC server

- See program-to-program communication server

## printer functions

## printer manager server

- server functions

## process connection (&&) function in all servers 63

## process connection and disconnection events 42

## process disconnection (\*\*) function in all servers 61

## process identification 7

## program types (LANDP for OS/2) 15

## program-to-program communication server

- server functions

## pure 16-bit programs 15

## pure 32-bit programs 16

## R

## relationship between clients and servers 1

## release

- LANDP context 81

- in the server manager routines 84

- remote servers** 4
- replied DATA length** 6, 10
- replied PARMLIST length** 6, 9
- reply buffer allocation (LANDP for AIX)** 75
- reply DATA** 2, 35
  - address 6, 10
  - length 6, 10
- reply PARMLIST** 2, 35
  - address 6, 9
  - length 6, 9
- request DATA** 2, 35
  - address 5, 10
  - length 5, 10
- request PARMLIST** 2, 35
  - address 5, 9
  - length 5, 9
- RequestFromLandp class** 89
- reserved names** 12
- resource name** 6
- return codes** 35
  - See *also* router return codes
  - See *also* server return codes
  - notation for xv
- REXX application programming interface (OS/2)** 113
- RISC System/6000 books** 208
- RMTAREQ call** 4, 55
- RMTREQ call** 4, 27
  - detailed definition 28
  - use in sample programs
    - See sample LANDP applications
- RMTREQ NoWait call** 4, 29
- RMTRPLY call** 4, 54
- router return codes** 5, 35
  - notation for xv
- routing requests** 1

## S

- sample application programs**
  - building 178
  - DOS client (COBOL) 170—180
  - for LANDP for AIX 36
  - header files (C language) 138
  - listed and annotated 127
  - OS/2 (COBOL) 149—169
  - OS/2 client (COBOL) 149—163, 170—180
  - OS/2 user-written server (COBOL) 164—169
  - tutorial 127
  - Windows NT (C) 127—147
    - building 178
    - header files (C language) 178
- sample application programs** (*continued*)
  - Windows NT (C) (*continued*)
    - using 178
  - Windows NT (COBOL) 149—169
  - Windows NT client (C) 127—137
  - Windows NT client (COBOL) 149—163
  - Windows NT user-written server (C) 139—147
  - Windows NT user-written server (COBOL) 164—169
- semaphores**
  - events under LANDP for OS/2 and LANDP for Windows NT 42
- send CPRBs, for application testing** 122
- sending asynchronous events** 64
- server programming**
  - # in name 11
  - AIX multiple-processing facilities 74
  - alias names 12
  - asynchronous request (ZN, Z4, Z5)
  - C++ language 87
  - CPRB 48
  - definition of server 1
  - end of service (ES) 57
  - events 36, 76
  - expanded memory 68
  - function of a server 1
  - GETREQ 51
  - GETRPLY 32
  - interrupt handling 68
  - LANDP for AIX server structure 73
  - memory management considerations 66
  - mono-service servers 11
  - multiple threads 69, 71
  - multiple-service servers 11
  - names of servers 10
  - options for LANDP for OS/2 and LANDP for AIX
    - servers
      - GETREQ 52
      - SRVINIT 50
  - process
    - connection (&&) 63
    - disconnection (\*\*) 61
  - receiving system requests 56
  - reply buffer allocation (LANDP for AIX) 75
  - reserved names 12
  - return codes 35
  - RMTAREQ 55
  - RMTRPLY 54
  - sending asynchronous events 64
  - server
    - acting as client 2

# Index

## server programming *(continued)*

- server *(continued)*
  - events 76
  - recognition (IN) 58
  - server child support 76
  - server-to-server calls 65
  - structure 5, 47
- server names 10
  - alias names 12
  - mono-service servers 11
  - multiple-service servers 11
  - reserved names 12
  - use of # 11
- server-to-server calls 65
- Smalltalk language 92
- SRVINIT 49, 75
- structure of a server 5
- timer generated request (TT) 59
- wait multiple (WM) in a server 70, 72
- where servers are located 1, 4
  - CPRB field 6
- workstation
  - connection (&&) 62
  - disconnection (\*\*) 61
- writing servers
  - LANDP for AIX 73
  - LANDP for DOS 66
  - LANDP for OS/2 69
  - LANDP for Windows NT 71
- server recognition (IN) function in all servers 58**
- server return codes 6, 35**
  - notation for xv
- server, definition of 1**
- servers supplied with LANDP**
  - See server programming
  - See user-written servers
- service marks 183**
- 16-bit programs 15**
- Smalltalk, writing application programs using 92**
- SPV (resource name) 48**
- SPVCPRB exit server 124**
- SRVINIT call 4, 75**
  - detailed description 49
  - used in sample application 143, 167
- start posting events (SP), supervisor local function**
  - used in sample application 133
- summary of changes xvii**
- supervisor local functions**
  - see Programming Reference manual i

- SVPCPRB.EXE 117**
- SVPCPRBN.EXE 117**
- SVPCPRBx.EXE programs 121**
- system events 40**
  - keyboard events 41
  - mouse events under LANDP for OS/2 and LANDP for Windows NT 41
  - process connection and disconnection events 42
  - semaphore events under LANDP for OS/2 and LANDP for Windows NT 42
  - timer events 41
- system requests**
  - end of service (ES) 57
  - process connection (&&) 63
  - process disconnection (\*\*) 61
  - server recognition (IN) 58
  - timer generated request (TT) 59
  - workstation connection (&&) 62
  - workstation disconnection (\*\*) 61
- system resources, names of 10**
- system verification programs 117**
  - See also testing an application program

## T

- terminate and stay resident (TSR) program 2**
- terms, definitions of 185**
- testing an application program 117**
  - defining a test 118
  - test panel 118
  - test panel, hexadecimal mode 120
  - using keyboard 120
  - using Windows 121
- testing LANDP applications 117**
- testing programs**
  - DCZYSVP application test program 118
  - DCZYXSVP, Motif version of DCZYSVP 121
- 32-bit programs 16**
- threads**
  - LANDP for OS/2 69
  - LANDP for Windows NT 71
- timeouts 8**
- timer events 41**
- timer-generated request (TT) function in all servers 59**
- trademarks 183**
- TSR (terminate and stay resident) program 2**
- TT (timer-generated request) function in all servers 59**



tutorial programming examples 127—180  
types of programs (LANDP for OS/2) 15

## U

### user-written servers

See *also* servers supplied with LANDP  
asynchronous events  
    sending 64  
client/server mechanism requests 56  
    end of service (ES) 57  
    process connection (&&) 63  
    process disconnection (\*\*) 61  
    timer generated request (TT) 59  
    workstation connection (&&) 62  
    workstation disconnection (\*\*) 61  
invoking the common API  
    GETREQ 51  
    GETRPLY 32  
    RMTAREQ 55  
    RMTRPLY 54  
    SRVINIT 49  
server-to-server calls 65  
structure 47

## V

VDM 43  
virtual DOS machine (VDM) 43  
Visual C++ programs 9  
VisualAge C++ web site 211  
VisualAge C++ programs 9  
VisualAge for COBOL 103  
VisualAge for COBOL for OS/2 call RMTREQ 28  
VisualAge for COBOL for OS/2 programs 9  
VisualAge generator 105—112  
    ASCII/EBCDIC translation 109  
    bit-oriented data 107  
    calling functions within the DLL 107  
    calling LANDP servers 106  
    generating an application 112  
    overview 105  
    return codes 112  
    testing applications 112  
    The LANDP Dynamic Link Library 105  
VisualAge Generator books 211  
VisualAge programs 9  
VisualAge Smalltalk web site 211

## W

wait for asynchronous events (WM) supervisor local  
    function  
        See *also* asynchronous events  
        use of, in a client 38  
        use of, in a server 70, 72  
wide area communication  
    books 209  
Windows 2000 xv  
Windows 3.1 support under LANDP for DOS 43  
Windows NT, COBOL programs under  
    VisualAge 103  
Windows NT, use of native system functions 2  
WM (wait for asynchronous events) supervisor local  
    function  
        See *also* asynchronous events  
        use of, in a client 38  
        use of, in a server 70, 72  
workgroup, definition of 1  
workstation functions in all servers  
    connection (&&) 62  
    disconnection (\*\*) 61  
workstation identification 6  
writing clients  
    See application programming  
writing servers  
    See application programming  
    See user-written servers

## X

X-Windows support under LANDP for AIX 39

## Z

Z5 (asynchronous request—event notification or  
    cancellation) supervisor local function  
    used in sample application 145



---

# **Sending your comments to IBM**

**LANDP® Family**

**Programming Guide**

**SC34-5781-00**

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form (RCF)
- By fax:
  - From outside the U.K., after your international access code use 44 1962 870229
  - From within the U.K., use 01962 870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name/address/telephone number/fax number/network ID.



---

# Readers' Comments

**LANDP® Family**

**Programming Guide**

**SC34-5781-00**

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

---

Name

---

Address

---

Company or Organization

---

Telephone

---

Email



You can send your comments POST FREE on this form from any one of these countries:

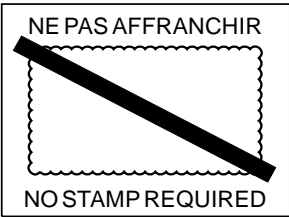
Australia	Finland	Iceland	Netherlands	Singapore	United States
Belgium	France	Israel	New Zealand	Spain	of America
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

2 Fold along this line

By air mail  
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO



REPONSE PAYEE  
GRANDE-BRETAGNE

IBM United Kingdom Laboratories  
Information Development Department (MP095)  
Hursley Park,  
WINCHESTER, Hants  
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name \_\_\_\_\_  
Company or Organization \_\_\_\_\_  
Address \_\_\_\_\_  
\_\_\_\_\_  
EMAIL \_\_\_\_\_  
Telephone \_\_\_\_\_

4 Fasten here with adhesive tape

1 Out along this line

1 Out along this line





Program Number: 5639-I90

Printed in the USA

SC34-5781-00

