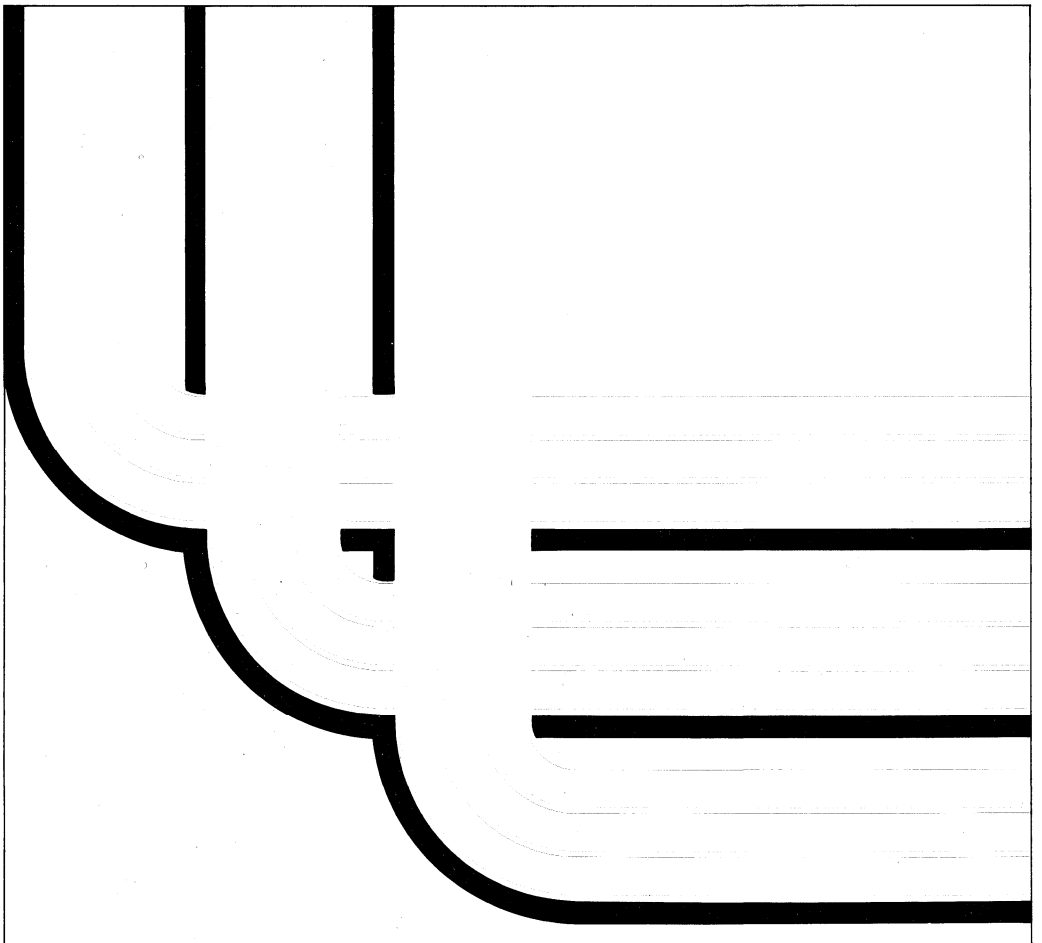
Application System/400

SC41-8254-02

PC Support/400: Application Program Interface Reference

Version 2



Application System/400

SC41-8254-02

PC Support/400: Application Program Interface Reference

Version 2

Take Note!

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

Third Edition (November 1993)

This edition applies to the licensed programs IBM PC Support/400 (Program 5738-PC1), Version 2 Release 3 Modification 0, and IBM Operating System/400 (Program 5738-SS1), Version 2 Release 3 Modification 0, and to all subsequent releases and modifications until otherwise indicated in new editions. This major revision makes obsolete SC41-8254-01. Make sure you are using the proper edition for the level of the product.

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

A Customer Satisfaction Feedback form for readers' comments is provided at the back of this publication. If the form has been removed, you can mail your comments to:

Attn Department 245
IBM Corporation
3605 Highway 52 N
Rochester, MN 55901-7899 USA

or you can fax your comments to:

United States and Canada: 800+937-3430
Other countries: (+1)+507+253-5192

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 or restricting your use of it.

Contents

Part 3. PC Support Windows Application Program Interfaces

Figures

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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 of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577, U.S.A.

This publication could contain technical inaccuracies or typographical errors.

This publication may refer to products that are announced but not currently available in your country. This publication may also refer to products that have not been announced in your country. IBM makes no commitment to make available any unannounced products referred to herein. The final decision to announce any product is based on IBM's business and technical judgment.

Changes or additions to the text are indicated by a vertical line (|) to the left of the change or addition.

Refer to the "Summary of Changes" on page xxiii for a summary of changes made to the PC Support/400 APIs and how they are described in this publication.

This publication contains small programs that are furnished by IBM as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. All programs contained herein are provided to you "AS IS". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

Programming Interface Information

This *PC Support/400 API Reference* is intended to help experienced programmers create application programs. This *PC Support/400 API Reference* documents General-Use Programming Interface and Associated Guidance Information provided by OS/400.

General-Use programming interfaces allow the customer to write programs that obtain the services of OS/400.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

Application System/400	Presentation Manager
AS/400	PS/2
C/2	RPG/400
IBM	SAA
Macro Assembler/2	SQL/400
OS/2	System/360
OS/400	System/370
Operating System/400	System Application Architecture
Personal System/2	400

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies:

Intel	Intel Corporation
Microsoft	Microsoft Corporation
Microsoft Windows	Microsoft Corporation
RUMBA	Wall Data Incorporated
TOPSPEED	Jensen & Partners International, Inc.
Windows	Microsoft Corporation

About This Manual

This manual describes the application program interfaces (APIs) provided by the PC Support/400 product.

This entire manual contains General-Use Programming Interface and Associated Guidance Information.

This manual details the characteristics of the APIs provided for certain PC Support functions. Using an API, you can write programs on a personal computer or PS/2 that can send data to, and receive data from, programs that run on an AS/400 system. The first section describes the interfaces provided for the PC Support router. The second section describes the interfaces to other PC Support functions that provide APIs. The third section describes high-level interfaces supplied by IBM. These interfaces can be called from application programs written by the user.

You may need to refer to other IBM manuals for more specific information about a particular topic. The *Publications Guide*, GC41-9678, provides information on all the manuals in the AS/400 library.

For a list of related PC Support publications, see "PC Support/400 Information." For a list of related publications, see the "Bibliography" on page H-1.

Who Should Use This Manual

This manual is intended for application and systems programmers who are familiar with PC Support/400, APPC concepts, and programs written for data communications.

To use the APIs described in this manual, you need PC Support/400 Version 2 Release 2.

PC Support/400 Information

The following is a summary of the documentation available for PC Support/400. For a complete overview of the AS/400 system documentation, see the *Publications Guide*, GC41-9678.

Table 0-1. PC Support/400 Tasks and Publications

Tasks	Environment	Look in
Planning, Installation, Administration, Problem Analysis, and Customization	DOS	*PC Support/400: DOS Installation and Administration Guide*, SC41-0006
	DOS (DBCS)	*PC Support/400: DOS Installation and Administration Guide (PS/55)*, SC41-0008
	OS/2	*PC Support/400: OS/2 Installation and Administration Guide*, SC41-0007
	OS/2 (DBCS)	*PC Support/400: OS/2 Installation and Administration Guide (PS/55)*, SC41-0009
Using PC Support Functions	DOS	*PC Support/400: DOS User's Guide*, SC41-8199
	DOS (DBCS)	*PC Support/400: DOS User's Guide (PS/55)*, SC41-2414
	OS/2	*PC Support/400: OS/2 User's Guide*, SC41-8200
	OS/2 (DBCS)	*PC Support/400: OS/2 User's Guide (PS/55)*, SC41-2415
Education	All	• Tutorial System Support[1] • PC Support Introduction (PCSINTRO)[2]
Problem Analysis	All	• Online message help (PCSHELP) and extended help[2] • The PC Support error log (PCSLOG)[2] • The PC Support Installation and Administration Guide for your environment
Technical Information and Programming	All	• *PC Support/400: DOS and OS/2 Technical Reference*, SC41-8091 • *PC Support/400: Application Program Interface Reference*, SC41-8254
Using RUMBA/400 and Programming	DOS Extenders Windows (SBCS) DOS Extenders Windows (DBCS) OS/2 2.0 (SBCS) OS/2 2.0 (DBCS)	• *PC Support/400: RUMBA/400 Guide and Reference*, SC41-0135 • *PC Support/400: RUMBA/400 Programmer's Tools Reference*, SC41-0136

Notes:

1. To start online education, enter STREDU at the AS/400 command line.
2. For information about using these PC Support help features, see the chapter on "Getting Help When You Need It" in the *PC Support User's Guide*.

Summary of Changes

The PC Support/400 Transfer Function and Remote Structured Query Language application programming interfaces have additional support for National Language Sequence Support tables. For more information see "Transfer Function Overview" on page 5-1 and "Application Development Using the Remote SQL API" on page 15-3.

The Windows APIs are now available for single-byte character set and double-byte character set environments. See Chapter 19, "PC Support/400 Windows Application Program Interface Overview" on page 19-1 for additional information.

The Data Queues APIs now support a non-destructive read (peek) as well as a destructive read (receive). For additional information, review the QSETMODE and EHNDQ_SETMODE APIs for Data Queues.

The Virtual Print APIs have been expanded to cover DOS and OS/2 operating systems in both SBCS and DBCS environments. An additional formfeed control character now can be added to ASCII transparent print jobs.

Many performance improvements have been made to the Remote SQL APIs. New APIs were added that allow SQL statements to be stored in an SQL package on the host AS/400 system for later execution. The EHNRQUPCUR and EHNRQ_UPCUR APIs can be called in a new way that allows the application to take advantage of improvements in SQL on the host AS/400 system. Both the EHNRQFETCH and EHNRQGETF APIs allow improved performance with no modifications to existing programs. For additional information see Chapter 15, "Remote SQL Function High-Level Application Program Interface" and Chapter 25, "Remote SQL Windows Application Program Interface."

Part 1. PC Support Low-Level Application Program Interfaces

PC Support Low-Level API Overview

PC Support Low-Level API Overview

This section describes the low-level application program interfaces (APIs) for the PC Support/400 functions. The interfaces described in this section must be accessed using a programming language that can manipulate registers and generate software interrupts.

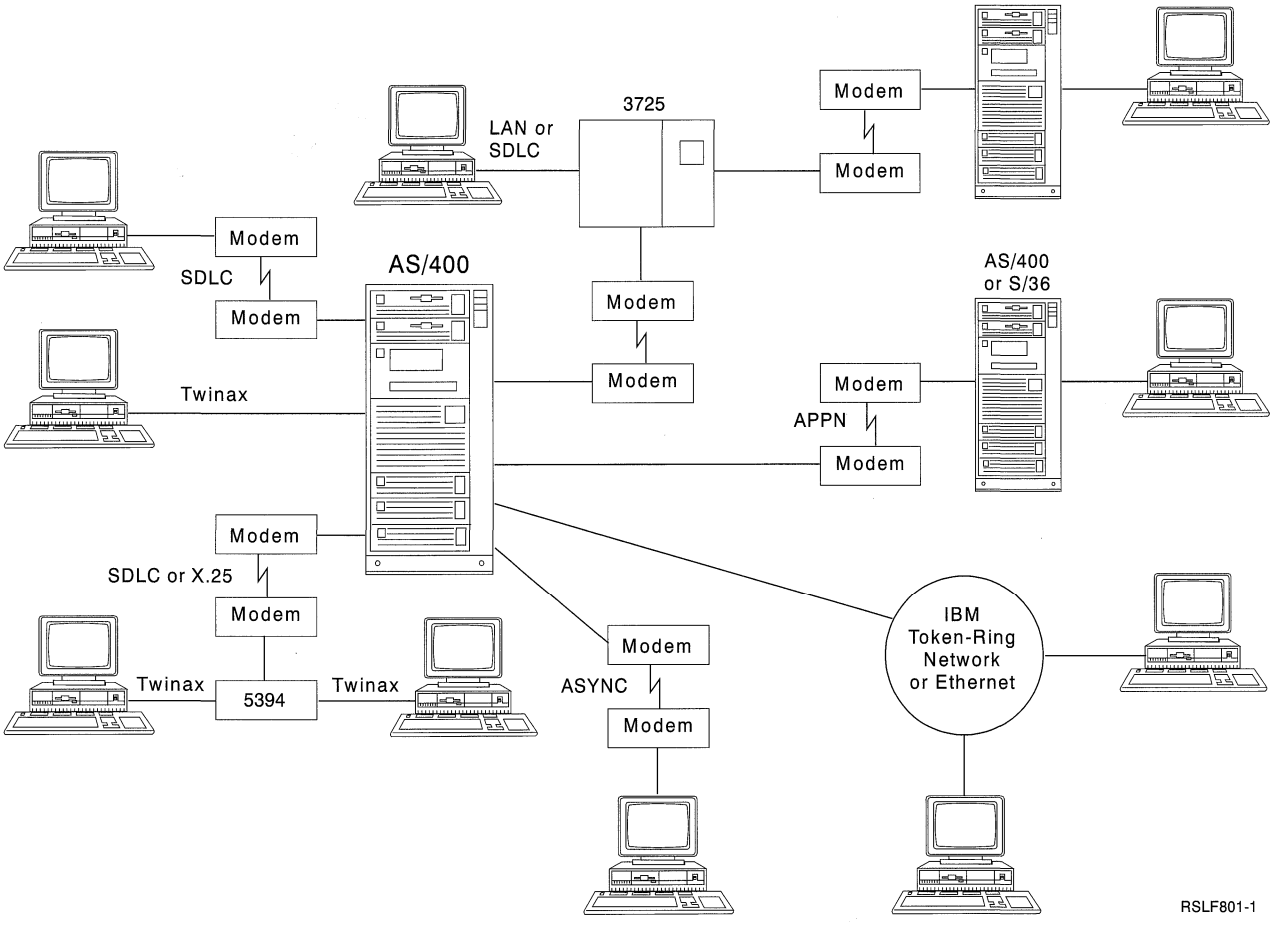
For information on the high-level APIs for DOS and OS/2* that provide access to the PC Support/400 functions using IBM C/2*, IBM Pascal/2, and IBM Macro Assembler/2*, refer to Part 2, "PC Support/400 High-Level Application Program Interfaces for OS/2 and DOS" on page 8-21.

Chapter 1. Introduction to the DOS Router APPC Interface

The PC Support/400 router permits communication between DOS-based cooperative processing programs and programs at Systems Network Architecture (SNA) Logical Unit (LU) 6.2 nodes in an advanced peer-to-peer networking (APPN) network. The following connections to the SNA nodes can be used:

- IBM Token-Ring Network
- Synchronous data link control (SDLC)
- Twinaxial
- Ethernet
- Asynchronous

The following figure shows the types of connections used with the PC Support router in an APPN network.



Router Components, Support, and Elements

LU 6.2 Components Implemented by the Router

The router supports the following functions of LU 6.2:

- Basic conversation verbs
- Mapped conversations using basic conversation verbs
- Synchronization level of confirm
- Conversation level security (user ID and password)
- Parallel sessions

SNA Node Support

The router supports direct links to physical unit (PU) 2.1 nodes (specifically System/36, AS/400*, or a network control program in a SNA subarea network). In all connections, the router functions only as a secondary data link control (DLC) node.

Router Elements

The router consists of three elements or parts:

- Resident element
- Starting and ending elements
- Transaction programs

Resident Element

The resident part of the router must be loaded before any router calls can be made. The router performs the designed APPC functions and provides an application program interface (API) for service and conversation requests.

The router runs as a resident extension of DOS. This means that when the router is loaded, it sets up its own interrupt vector, and then ends processing while remaining resident. This allows transaction programs to pass requests using an interrupt instruction.

Starting and Ending Elements

Starting and ending commands establish and control connections to other systems and control the loading and unloading of the resident part of the router. There are three starting and ending commands:

- STARTRTR is used with a configuration file to load the router, to establish connections to other systems, and to control the connections.
- STOPRTR ends all connections to all systems.
- RMVPCS can remove the resident part of the router from memory.

  These three commands and the configuration file (CONFIG.PCS) are described in *PC Support/400: DOS Installation and Administration Guide*, SC41-0006.

Transaction Programs

Transaction programs perform user application processing in cooperation with transaction programs at other APPC nodes. This is accomplished by conversations between the cooperating programs.

Configuration

A configuration file must be created before you can use the router for communications. The configuration file contains entries describing things like the type of connection, the network name of your personal computer, and the name of the other systems that you want to connect.

A configuration file is created for you when PC Support is installed. CONFIG.PCS is the name of the configuration file. See the *PC Support/400 DOS Installation and Administration Guide* for information on configuring PC Support.

Application Programming

Application programming is the task of writing programs to perform user application processing. Application programs are distributed by communicating with other programs over APPC conversations.

This manual describes how to use APPC verbs to:

- Create conversations
- Send data
- Receive data
- End conversations

User Tasks

The following tasks are performed by the user. See the *PC Support/400 DOS Installation and Administration Guide* for more information about the PC Support commands used to perform the tasks.

- Starting the router

  The STARTPCS or STARTRTR commands can be used to start the router.

- Starting and ending transaction programs

  These tasks depend on the method you provide for your users to start and end your program.

- Stopping the router

  The STOPRTR command is used to stop the router.

- Removing the router from memory

  The RMVPCS command is used to remove the router from PC memory when you no longer need to communicate with the host system.

Data Link Control Considerations

The router provides a high degree of transparency for the underlying data link control (DLC) to the application and to the end user. Some operational considerations may be apparent, such as dialing required on an SDLC switched line or the time it takes to open the token-ring adapter.

Performance is highly dependent on the type of connection that is used. Timing dependencies should be avoided in transaction programs that are used on different types of connections.

Router Packaging

The PC Support router is shipped with the PC Support product. It is included on the PC Support installation diskette and also in the following PC Support folders on the AS/400 system:

- QIWSFLR (for DOS)
- QIWSOS2 (for OS/2)
- QIWSFL2 (for extended DOS)
- QIWSFLRD (for DOS on PS/55)
- QIWSOS2D (for OS/2 on PS/55)
- QIWSFL2D (for extended DOS on PS/55)

By using the PC Support update program (PCSUPDT) and the appropriate PC Support folder on the AS/400 system, you can make sure that your users have the latest level of the router programs. You can also use the PCSUPDT program to distribute your own transaction programs and updates.

Chapter 2. Development of a Router Transaction Program

A transaction program (TP) is part of an application program that uses APPC communications functions. Using these functions, an application program can communicate with application programs on other systems that support APPC. A single application program may contain many transaction programs to one system or to many different systems. These transaction programs may be concurrent (active at the same time), serial (one after another), or any combination.

To request the router to take an action, your application program must issue a verb. A verb is a formatted request that the router runs. Programs use APPC verb sequences to communicate with programs at other locations (network nodes). You can view the set of APPC verbs as a programming language in which you can write conversations. APPC verbs are coded as records; each verb having a precisely defined syntax. Your programs gain access to the router services by providing verb records to the router application program interface.

When your local transaction program exchanges information with a remote transaction program, the programs are called partner transaction programs.

The verbs can be issued in any programming language that supports access to the registers for the Intel** processors and allows you to issue software interrupts. Language interfaces for C, Modula-2, and Macro Assembler are described in "Program Examples" on page 4-60.

Understanding Conversation States

Transaction programs use conversations to communicate. A send-receive relationship exists between transaction programs using a conversation. One transaction program issues verbs to send information and the other (partner) transaction program issues verbs to receive information.

When it finishes sending information, the sending transaction program can transfer send control of the conversation to the receiving transaction program.

Understanding Available Router Services

The router provides the following services that are important to your transaction program:

- Default system

  The router maintains a default system name that is used when you start a conversation. The default system name is used if no system name is supplied on the allocate verb when starting a conversation.

- Mode name

  Mode names are defined on the AS/400 system. They are used in APPC to group sessions together so those sessions all have similar attributes. The PC Support router only supports the mode name QPCSUPP.

The session limits should be set up so that all of the sessions are controlled by the personal computer (the number of locally controlled sessions should be set to 0). No sessions should be started automatically.

- User ID and password values

  When the PC Support router is first started, the user is prompted for a user ID and password for each system in the configuration file (CONFIG.PCS). When your transaction program allocates a conversation, the router sends the user ID and password along with the allocate request to the host system. Your program does not supply the user ID or password.

Choosing Basic or Mapped Conversations

The router only supports basic conversation verbs. However, a transaction program that provides its own mapping layer can conduct a mapped conversation using the basic conversation verbs. To do this, the transaction program using the router application program interface is responsible for the generalized data stream required for mapped conversations.

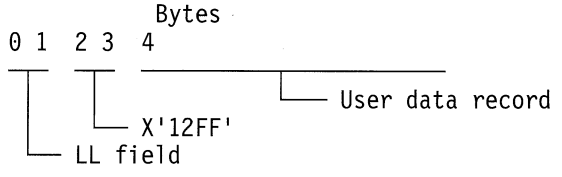
Basic Conversations

In a basic conversation, transaction programs exchange logical records that contain a 2-byte length prefix called the LL field. The LL field is formatted by the transaction program. Your transaction program must supply the LL field at the head of each block of data. The LL field specifies the sum of the length of the data plus its own length, which is 2 bytes. For example, to send 12 bytes of data, set the field to 14 (12+2).

Mapped Conversations

When a program uses mapped conversation verbs, it does not supply the two-byte prefix for logical records. The LU 6.2 support supplies this field. Because the router does not support mapped conversation verbs, the transaction programs that use the router API must always supply the value for the LL field.

Mapped conversations can be used to allow partner transaction programs to use mapped conversation verbs. To use a mapped conversation with the basic conversation verbs, your transaction must specify the conversation as mapped on the allocate verb. Each record sent must have the following format:



Bytes 0 and 1 contain the sum of the length of user data plus 4. Bytes 2 and 3 contain the value X'12FF', which is the general data stream identifier (GDSID). X'12FF' indicates that the data stream contains a mapped conversation record.

All data received on the mapped conversation must also be in this format. Your transaction program must look for user data starting in byte 4 of the record.

Selecting Transaction Program Names

Transaction program names identify the partner transaction program and are supplied on the allocate request. Transaction program names are from 1 to 64 characters long and must be specified in EBCDIC characters. AS/400 transaction program names must be formatted with the program name first, followed by a period, followed by the library name as follows:

```
PGMNAME.LIBNAME
```

If the library name is not specified, do not specify a period after the program name. If no library name is sent, the library list is searched.

Using Security

The router supports user verification. This enables the host system to verify the identity of the user before granting access to the transaction program and its resources. The user ID and password are entered by the user when the connection is first made to a remote system. If the security values are incorrect or the user is not authorized to a transaction program, the allocate request is denied.

Converting Between ASCII and EBCDIC

Whether a transaction program needs to translate data depends on the design of the application and the support at the remote system.

In most cases, character parameters that are passed to the router must be in EBCDIC format. The remote system name field and the transaction program name must be specified as EBCDIC characters.

The router maintains translation tables for converting between ASCII and EBCDIC. Your transaction program can access these tables by using the appropriate service verbs.

Chapter 3. Introduction to Router Verb Types

This chapter describes the verbs used with the PC Support DOS router.

Verb Types

The router uses the following verb types:

- Service verbs
- Basic conversation verbs
- APPC verbs in a transaction program

Service Verbs

Service verbs are used to get access to information maintained internally by the router. Four service verbs are supported by the router application program interface:

Get_ASCII_TO_EBCDIC_TABLE
Provides a pointer to the ASCII-to-EBCDIC translation table being used by PC Support.

Get_EBCDIC_TO_ASCII_TABLE
Provides a pointer to the EBCDIC-to-ASCII translation table being used by PC Support.

Get_System_List
Returns a list of system names. The router has an active connection to each of these systems.

Get_Fully_Qualified_System_List
Returns a list of fully qualified system names. The router has an active connection to each of these systems.

Query_Router_Capabilities
Returns a list of the router's capabilities.

Basic Conversation Verbs

Conversation verbs are issued by transaction programs to send information to, and receive information from, partner transaction programs. The router only supports basic conversation verbs. The following list shows the basic conversation verbs supported by the router:

Verb	Description
ALLOCATE	Requests a conversation with a remote transaction program.
CONFIRM	Sends a confirmation request to a partner transaction program and waits for a reply. This verb enables the local and partner programs to synchronize their processing.
CONFIRMED	Sends a confirmation reply to the partner transaction program after a confirmation request has been received.

DEALLOCATE Sent to indicate that a transaction program is done with a conversation. All resources associated with the conversation are released and made available to other transaction programs.

FLUSH Clears the send buffer of the router by sending all buffered data to the partner transaction program.

GET_ATTRIBUTES Returns information about the specified conversation.

PREPARE_TO_RECEIVE Changes the conversation state from send to receive state and sends any buffered data.

RECEIVE_AND_WAIT Receives information already available or waits for information to arrive on the specified conversation and then receives the information.

RECEIVE_IMMEDIATE Receives any currently available information from the specified conversation but does not wait for information.

REQUEST_TO_SEND Notifies the partner transaction program that the local transaction program is requesting send control of the conversation.

SEND_DATA Sends data to the partner transaction program.

SEND_ERROR Informs the partner transaction program that the local transaction program has detected an error.

A Minimal Starter Set

There are three phases in any conversation: its startup, the exchange of data, and its take-down. In the following example, the local transaction program initiates the conversation, sends data to the partner transaction program, and ends the conversation.

Starting a Conversation

ALLOCATE Allocates a conversation by specifying:

- The remote system to establish a session with.
- The partner transaction program that should be the partner in the conversation (specified in the tp_name parameter).

Exchanging Data

The following two verbs are used to illustrate a simple data exchange. Additional verbs, described later, give transaction programs a great deal of control over the sending and receiving of data and other conversation indicators.

SEND_DATA Directs the router to send a block of data to the partner transaction program.

RECEIVE_AND_WAIT Directs the router to wait for a block of data from the partner transaction program and to receive it into a local program buffer when it arrives.

Taking Down the Conversation

DEALLOCATE Releases the resources used for the conversation.

Figure 3-1 illustrates a pair of complete transaction programs using these verbs in a conversation.

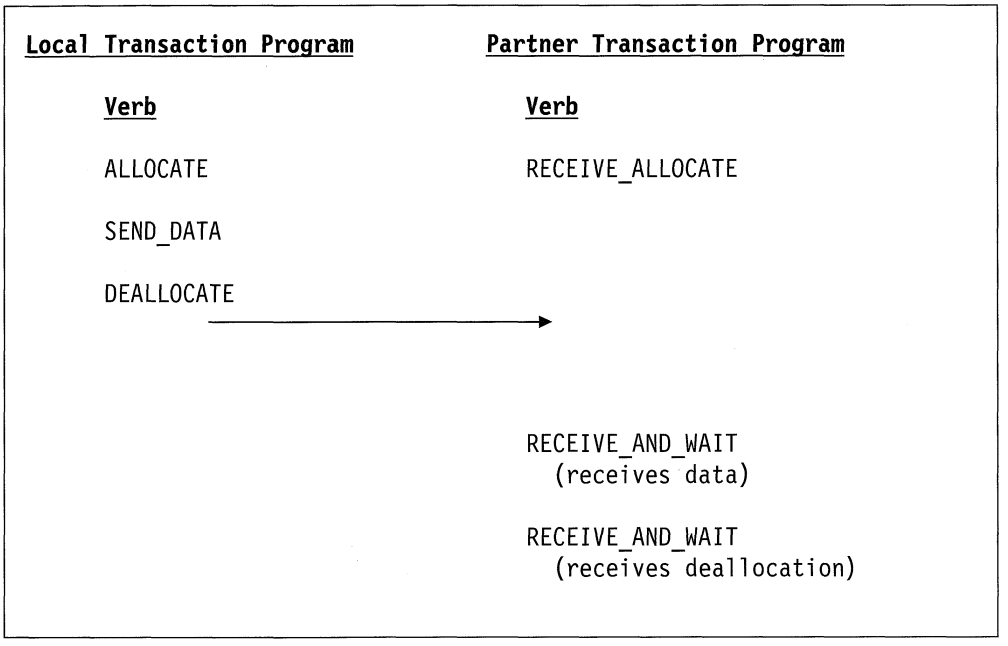


Figure 3-1. The Local Transaction Program Sending One Block of Data to the Partner Transaction Program

In this example, the local transaction program allocates the conversation, sends the data and deallocates the conversation. If the data is not larger than the send buffer size, nothing is sent until the DEALLOCATE verb is issued. This minimizes the number of buffers actually transmitted.

The partner transaction program issues RECEIVE_AND_WAIT twice. The first occurrence causes the data to be returned to the partner transaction program. The second occurrence returns an indicator telling the partner transaction program that the conversation has been deallocated.

Confirmation Processing and Error Reporting

Your transaction program can request that the partner transaction program confirm that all the data sent so far has been received and processed successfully. This is called **confirmation processing** and is started by specifying CONFIRM for the sync_level parameter on the ALLOCATE verb.

The CONFIRM verb requests that the partner transaction program confirm that all the data has been received and processed successfully. The partner transaction program either issues a response confirming the request (CONFIRMED) or reports an error condition (SEND_ERROR).

The type(SYNC_LEVEL) parameter on the DEALLOCATE verb performs the same function as the CONFIRM verb whenever the conversation has been allocated with sync_level(CONFIRM).

The transaction program issuing the CONFIRM verb (or DEALLOCATE with type(SYNC_LEVEL)) waits until a confirmation is received before processing continues.

Figure 3-2 illustrates a pair of complete transaction programs using confirmation processing.

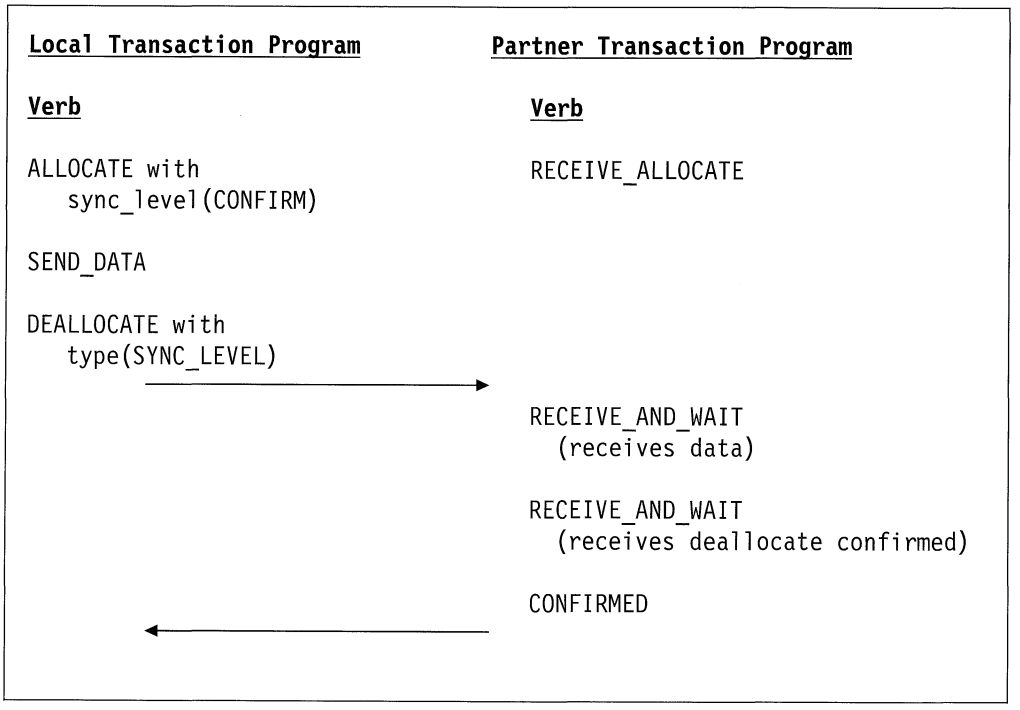


Figure 3-2. A Simple Send-Receive between Local and Partner Transaction Programs with Confirmation

This example is similar to Figure 3-1 on page 3-3, but with confirmation processing. The second RECEIVE_AND_WAIT verb gets an indicator showing the conversation has been deallocated and that the local transaction program has asked for a confirmation that all data has been received and processed. The partner transaction program can use a CONFIRMED verb for a positive reply or a SEND_ERROR verb for a negative reply. When the partner transaction program issues the CONFIRMED verb, the local transaction program's DEALLOCATE verb is satisfied and the conversation ends.

Forcing Data in the Internal Buffers to Be Sent

Data is buffered internally by the router before it is sent. A transaction process can force data in an internal buffer to be sent. The FLUSH verb forces the local transaction program to send to the partner transaction program all the data it is holding in its internal buffer. You do not normally have to use this service because the local transaction program automatically sends the data stored in an internal buffer when the data in the buffer exceeds its predetermined limit, or when a verb is issued that requires the buffer to be sent.

Issuing any of the following verbs also causes the buffers to be flushed:

- DEALLOCATE
- CONFIRM
- PREPARE_TO_RECEIVE

- RECEIVE_AND_WAIT

Figure 3-3 illustrates a pair of complete transaction programs using the FLUSH verb in a conversation.

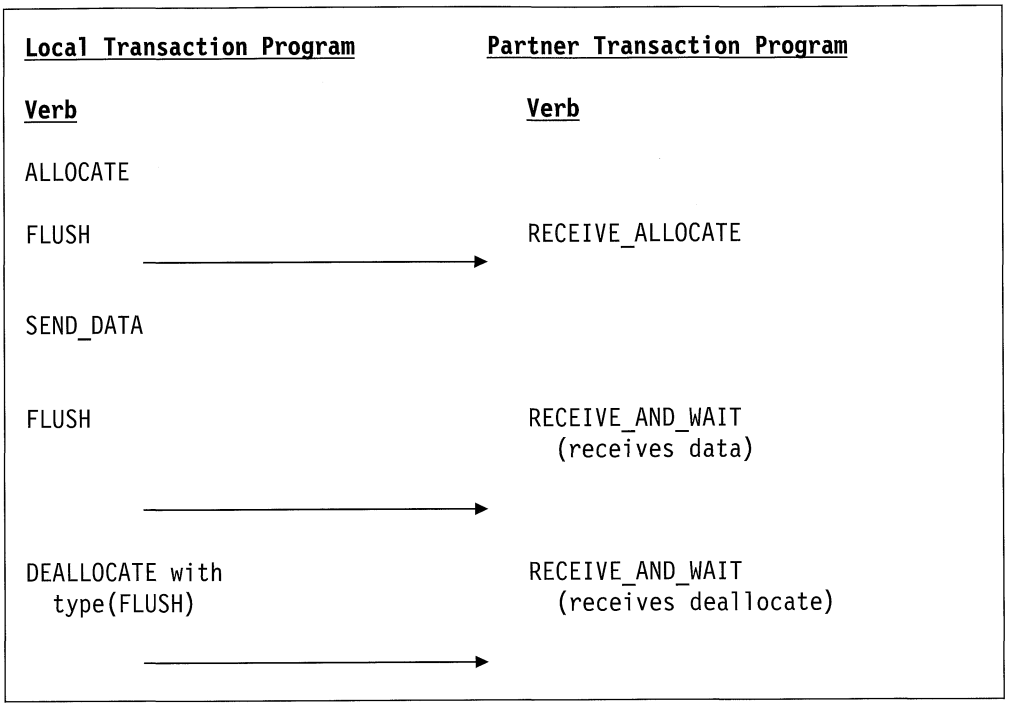


Figure 3-3. The Local Transaction Program Flushing Its Buffers after Each Send Verb

In this example, the local transaction program allocates the conversation, sends the data, and deallocates the conversation. Because of the first FLUSH verb, the allocation request is sent immediately to the partner transaction program. The second FLUSH sends the data immediately to the partner transaction program. The DEALLOCATE verb is issued with the type parameter equal to FLUSH, so deallocation takes place immediately with no confirmation required.

The FLUSH verb causes additional line flows and communications processing. It is not needed if the next verb causes the buffer to be sent. The FLUSH verb helps performance when data in the internal buffer is needed by the partner for immediate processing, but the transaction program might not issue another verb that would cause the buffer to be sent.

Changing Conversation States

The verbs that a transaction program can issue for a particular conversation depend on the state of the conversation. As the program issues verbs, the state of the conversation can change. The change in the state of the conversation is a result of the function of the verb, a result of a verb issued by the partner program, or a result of network errors.

The router defines rules that your program must follow when using a conversation. To enforce these rules, the router manages the state of your conversation and allows your program to perform certain operations only when the conversation is in the correct state.

The state of a conversation is defined in terms of the local program's view of the conversation. The states of other conversations allocated to the program can be different. For example, one conversation can be in Receive state and another in Send state concurrently. The following conversation states are defined at the conversation protocol boundary:

Reset — This is the state of a conversation prior to allocation and subsequent to deallocation; that is, the conversation does not exist.

Send — The program can send data or request a confirmation.

Receive — The program can receive information from the partner program.

Confirm — The program can reply to a synchronization request.

If a verb is issued in a state that is wrong, a state-check condition occurs and the STATE_CHECK return code is returned to the program.

Table 3-1 shows each verb and the conversation states in which it can be issued.

Table 3-1. Correlation between Conversation Verbs and Conversation States

	Conversation States			
Verb	**Reset**	**Send**	**Receive**	**Confirm**
ALLOCATE	Yes	N/A	N/A	N/A
CONFIRM	N/A	Yes*	No	No
CONFIRMED	N/A	No	No	Yes
DEALLOCATE with type(FLUSH) or type(SYNC_LEVEL)	N/A	Yes*	No	No
DEALLOCATE with type(ABEND_PROG)	N/A	Yes	Yes	Yes
FLUSH	N/A	Yes	No	No
GET_ATTRIBUTES	N/A	Yes	Yes	Yes
PREPARE_TO_RECEIVE	N/A	Yes*	No	No
RECEIVE_AND_WAIT	N/A	Yes*	Yes	No
RECEIVE_IMMEDIATE	N/A	No	Yes	No
REQUEST_TO_SEND	N/A	No	Yes	Yes
SEND_DATA	N/A	Yes	No	No
SEND_ERROR	N/A	Yes	Yes	Yes

Use the following information to interpret Table 3-1:

Yes — The program is allowed to issue the verb when the conversation is in that state.

Yes* — *Yes* only if the verb is issued at a logical record boundary; otherwise, *no*. See "Basic Router Conversation Verbs" on page 4-7 for a further discussion of logical record boundary.

No — The program cannot issue the verb because it is not allowed in that state.

N/A — The state is not applicable because it cannot exist at the time the verb is issued or because it is not relevant to the verb.

After issuing an ALLOCATE verb successfully, your initial conversation state is Send. Figure 3-4 on page 3-7 illustrates a pair of complete transaction programs

using confirmation processing and indicates the state of the conversation before each verb is issued.

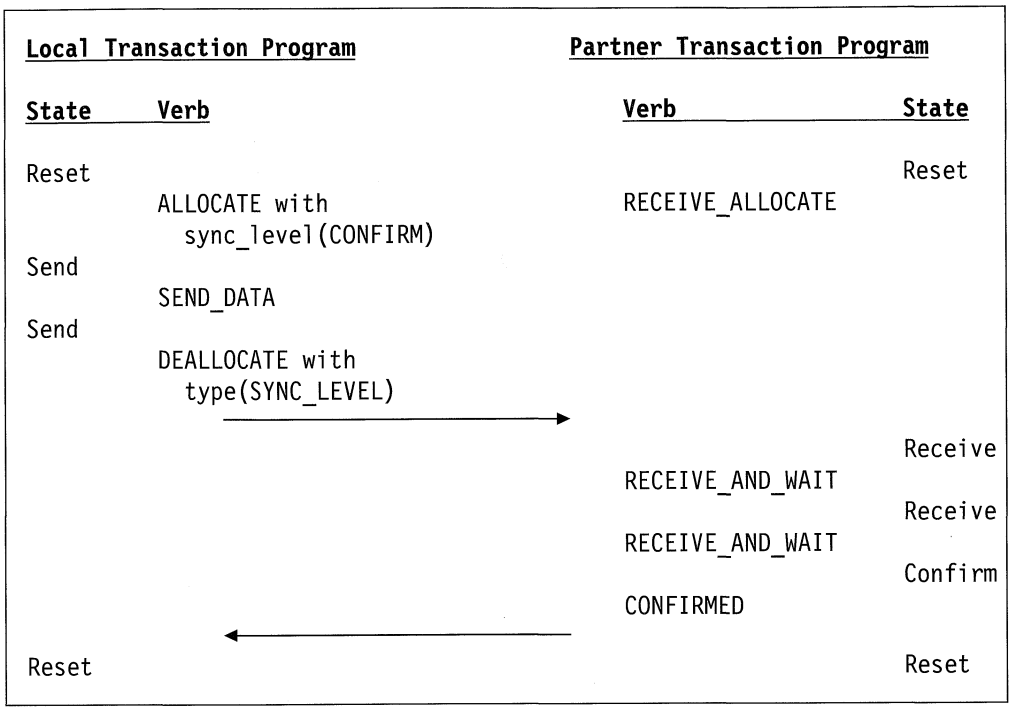


Figure 3-4. The Local Transaction Program Showing the State before Each Verb Is Issued

This example is identical to Figure 3-2 on page 3-4, but with the conversation states indicated before each verb is issued.

Managing Changes in the Conversation States

Two router verbs are specifically designed for managing changes in the conversation state: REQUEST_TO_SEND and PREPARE_TO_RECEIVE.

The (REQUEST_TO_SEND) verb requests permission from the partner transaction program to send data. For any conversation, only one transaction program at a time can send data. If the transaction program Receive state needs to send data, it can request permission to send by issuing the REQUEST_TO_SEND verb. The program might receive a send indicator from the partner transaction program. This changes the direction of data flow on the conversation.

The PREPARE_TO_RECEIVE verb gives the partner program permission to send data.

The request_to_send_received indicator is returned in the Receive verbs and in the Send verbs. It indicates whether the partner transaction program issued a REQUEST_TO_SEND verb (the values are YES or NO). The REQUEST_TO_SEND verb issued by the partner requests the local program to enter Receive state and to place the partner in Send state. However, this request is not binding; the local transaction program can check the indicator, see that its value is YES, yet continue to send data to the partner. See the individual verb descriptions for more detail on the request_to_send_received indicator.

Figure 3-5 on page 3-8 illustrates a typical sequence of verbs.

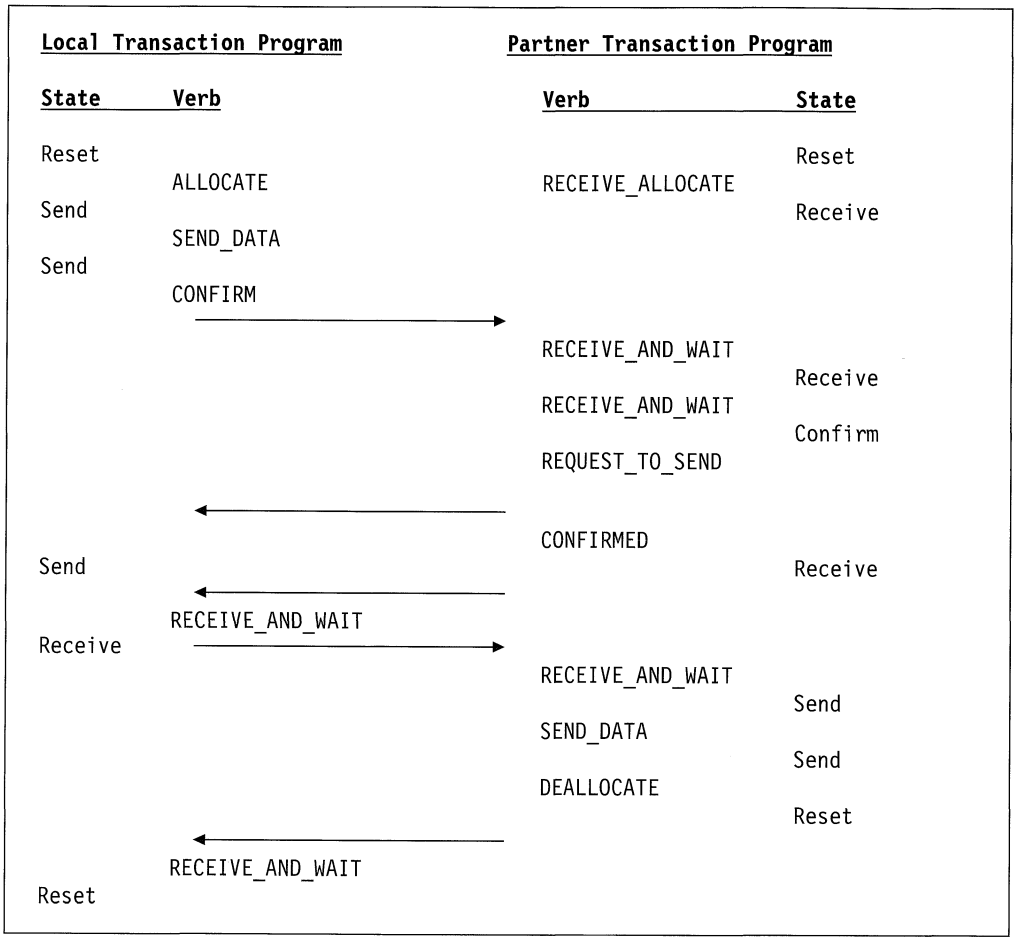


Figure 3-5. A Conversation between Local and Partner Transaction Programs Illustrating the State-Change Verbs

In this example, the local transaction program allocates the conversation, sends data, and then asks for a confirmation. The CONFIRM verb causes the information to be transmitted. The partner transaction program receives the data. This transaction program then receives the indicator requesting confirmation that all data has been received and processed.

Based on the data, the partner transaction program can determine if it needs to send data back. The partner transaction program issues a REQUEST_TO_SEND verb followed by a CONFIRMED verb. The local transaction program gets indicators that the confirmation has been received and that the partner transaction program wants to send data. By issuing the RECEIVE_AND_WAIT verb, the conversation is turned around so that the partner transaction program can send data. The partner transaction program must issue a RECEIVE_AND_WAIT verb to get the indicator that it can now send data.

The two Receive verbs issued by the local transaction program get the data and the indicator that the conversation was deallocated. The REQUEST_TO_SEND verb is only a request. It does not cause an automatic change from Receive state to Send state. It is the responsibility of the transaction program in Send state to issue a Receive verb or a PREPARE_TO_RECEIVE verb.

Monitoring State Changes

The router returns a 1-byte what_received indicator on the two Receive verbs (see "Using the Different Receive Verbs"). On each of the Receive verbs, two different kinds of information can be received:

- Data from the partner.

  DATA, DATA_COMPLETE, or DATA_INCOMPLETE.

- Status of the conversation.

  CONFIRM, CONFIRM_DEALLOCATE, CONFIRM_SEND, or SEND.

See Figure 3-2 on page 3-4 for an example of two RECEIVE_AND_WAIT verbs being issued; the first verb receives the conversation status and the second receives a block of data.

A transaction program should examine the what_received indicator only if the verb was successful (that is, its return code was OK). See the individual verb descriptions for more details about the what_received indicator.

Using the Different Receive Verbs

Two different conversation verbs can be used to receive data:

- RECEIVE_AND_WAIT
- RECEIVE_IMMEDIATE

Using the RECEIVE_AND_WAIT verb causes your transaction program to wait indefinitely until it receives data or a state-change indicator from its partner. The RECEIVE_AND_WAIT verb makes the transaction program design simpler. You may want to design your transaction program so that it can do other functions while waiting for the partner.

The RECEIVE_IMMEDIATE verb returns to its caller immediately. If there is data or an indicator available in the router internal buffers, it is returned. Otherwise, the router return code indicates that there is nothing to receive at this time. The transaction program can perform some other functions and then check again later to see if data or an indicator has been received. You can use the RECEIVE_IMMEDIATE verb to do a form of polling where your program never waits to receive data, but periodically asks whether data or an indicator has been received. If your program is running in an environment where two or more tasks can be performed, such as Microsoft** Windows**, you should use the RECEIVE_IMMEDIATE verb rather than the RECEIVE_AND_WAIT verb. The RECEIVE_IMMEDIATE verb allows other tasks to run while your program is waiting for data from the host system.

The RECEIVE_IMMEDIATE verb cannot be issued in the send state. Use the PREPARE_TO_RECEIVE verb in the send state before using the RECEIVE_IMMEDIATE verb.

Verb Descriptions

Each verb description contains the following:

- Name
- Purpose
- Syntax
- Supplied parameters
- Returned parameters (including return codes)
- State change
- Additional information.

Name

This is the name of the verb.

Purpose

This section gives a brief explanation of the verb function. Supplementary information is included in the "Additional Information" section for each verb description.

Syntax

Each verb description contains two flow-type diagrams:

- The first diagram shows the relationships among the supplied parameters and between each parameter and its options (choices).
- The second diagram shows the relationships among the returned parameters, including return codes resulting from processing the verb and parameters with information that the transaction program needs.

Supplied-Parameter Diagrams: As you read the diagrams for the supplied parameters, keep in mind the following guidelines:

- Read the diagram from left to right, beginning with the first line. A continuation arrow (→) tells you to continue to the next line.
- The verb name is at the left side of the first line in uppercase letters.
- Supplied parameters are in lowercase letters.
- Variable information about a parameter follows in italics. (As the programmer, you supply this information.)
- If the choices are on branched lines, follow one branch.

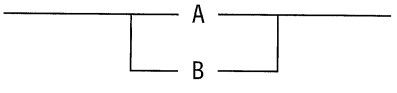


Figure 3-6. A Branched Option to Choose for a Parameter

- Choices you can omit are shown like this.

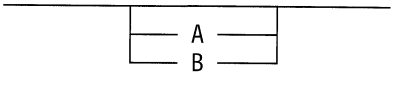


Figure 3-7. A Branched Option to Choose or Omit for a Parameter

- If a single parameter or option is chosen more than once, it is encased in a repetition arrow.

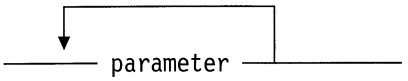


Figure 3-8. Repeatable Choices for a Parameter

- Choices encased in a repetition arrow means that you can pick one or more of the items.

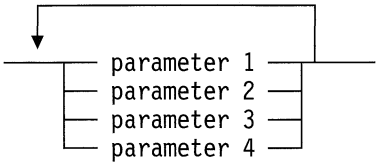


Figure 3-9. Select One or More from Multiple Choices

- If there is no text at the top of the arrow, you can use as many items as you want.
- If there is text at the top of the arrow, follow the directions in the text.

  For example, you may be given a maximum number of choices or be instructed to use all the enclosed parameters and supply an option for each.

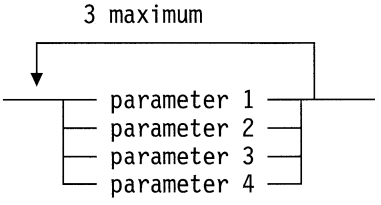


Figure 3-10. Multiple Choices with Instructions

- You have reached the end of the diagram when you arrive at an end symbol:

  ⋈

Returned-Parameters Diagrams: To read the diagrams for the returned parameters:

- Start at the branched lines at the upper left and read from left to right until you arrive at an end symbol.
- Return codes are listed first on the branches. The return code:
  - OK means that the verb ran successfully and that additional information may follow.
  - UNSUCCESSFUL means that the verb ran, but conditions at the time were such that it could not perform the desired function.
  - Return_code with error information means that the verb was not run.

    The error-related return codes are not diagrammed. To see a list of possible error-related return codes for a specific verb, refer to each verb description in the verb chapters. In some cases, such as parameter and state checks, the secondary return codes provide more detailed information.

- If the return code is OK, the additional information requested by the transaction program follows. Read this information from left to right on each line until you reach an end symbol.

The returned parameters for the DISPLAY verb are unique. The DISPLAY verb returns virtually all the information associated with SNA operations. To make the information easier to find, the returned parameters are divided into a number of sections. The sections are identified with reverse-image numbers.

Supplied Parameters

Each parameter that the transaction program supplies to the router is described. When used within text, parameters are shown in lowercase.

If you must select an option for the parameter, you can find the possible options in uppercase letters listed after the parameter.

In some cases, you must supply variable information such as a number (*n*), a name, an address, or a value. These variables are in italics.

An explanation of each of these variables follows:

- *n* (number) represents an integer that is 1, 2, or 4 bytes long. The integer is stored in Intel byte-reversed format.
- *name* represents a character string of a predetermined length that probably corresponds with a configured profile or field name. The name can be in ASCII or EBCDIC and is usually padded on the right with blanks.
- *address* represents a 4-byte DOS memory address. The address is stored in Intel byte-reversed format.
- *value* represents an arbitrary string of hexadecimal bytes of a predetermined length.

Returned Parameters

The router returns parameters to the transaction program after processing the verb. Some returned parameters provide information that the transaction program needs. Other returned parameters, called return codes, indicate that the verb ran either successfully or unsuccessfully. Specific return codes provide specialized error information.

Some return codes are used by many of the verbs. For this reason, they are called *common **return codes***. In the descriptions of the individual verbs, these common return codes are listed with a cross-reference to the "Verb Return Codes" on page 4-45. The return codes that are specific to individual verbs are discussed in the verb chapters.

In some cases, variable information, such as a number (*n*), a name, an address, or a value, is included in the returned parameters and shown in italics. The meaning of the italicized terms is explained in the previous section.

For more detailed information, including the cause of each return code and the action that must be taken, refer to "Verb Return Codes" on page 4-45.

State Change

This section of the verb description discusses the state of the conversation. The state of a conversation is the current status or condition. See Appendix B, "Table of Router Conversation States," for a list of the conversation states.

Additional Information

This section discusses verb functions and provides supplementary information. The following pages include an example of a verb description.

Router Verb Example

Purpose

Provides a general description of the verb's purpose.

Syntax

Syntax is a graphic representation of the correct format of the verb in relationship to its parameters and options.

The following is a graphic presentation of the verb and its supplied parameters.

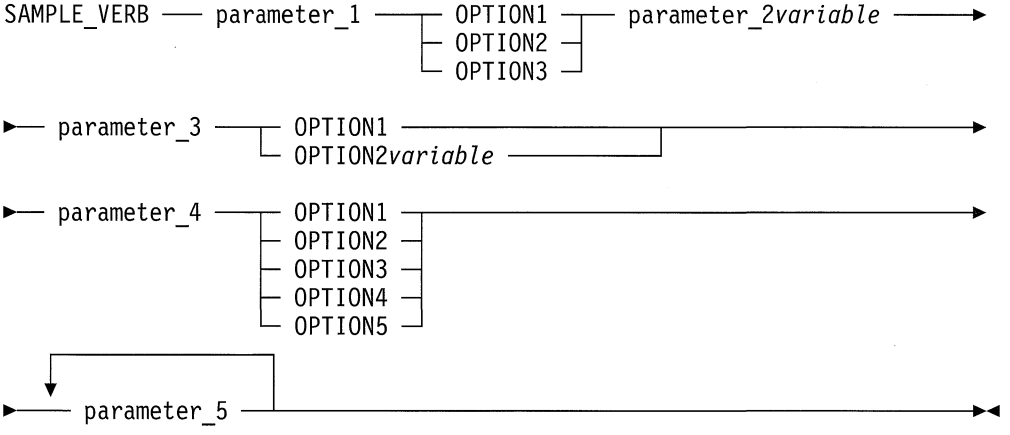


Figure 3-11. Sample Syntax of a Verb and Supplied Parameters

The following is a graphic representation of the returned parameters.

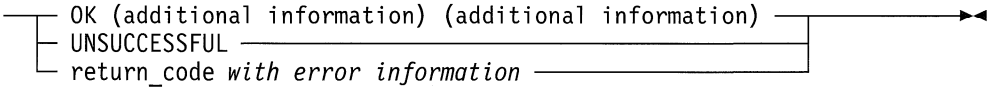


Figure 3-12. Sample Syntax of Returned Parameters

Parameters

parameter_1
: This is an example of a supplied parameter with three OPTIONS.

parameter_2
: This is an example of a supplied parameter with a *variable*.

parameter_3
: This is an example of a supplied parameter with two OPTIONS, one of which has a *variable*.

parameter_4
: This an example of a parameter that has five OPTIONS.

parameter_5
: This is an example of a parameter with no OPTIONS and no *variable*s. This is also an example of a parameter that can be repeated.

Return Codes

OK
: This means that the verb ran successfully. For some verbs, additional information may be provided to the issuing transaction program from the partner transaction program.

UNSUCCESSFUL
: This means that a verb was issued and ran without an error, but the desired result was not achieved.

The following return code gives error information:

return_code with error information
: This means the verb did not run because of an error; a return code with error information was returned to the issuing program.

  For more detailed information, including the cause of each return code and the action you must take, refer to "Verb Return Codes" on page 4-45.

Chapter 4. Router Service Verbs and Basic Conversation Verbs

This chapter describes the service verbs and the basic conversation verbs used with the PC Support DOS router.

Router Service Verbs

The following sections for router service verbs describe in detail:

- Purpose
- Syntax
- Returned parameters and their options
- Verb record formats

Service verbs are used to get information from the router about the operating environment. The router must be loaded before service verbs can be issued.

Get_ASCII_TO_EBCDIC_TABLE

Purpose

Returns a pointer to the ASCII-TO-EBCDIC translation vector maintained by the router. This translation vector allows you to use an ASCII character as an index to retrieve the corresponding EBCDIC code point.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:

```
Get_ASCII_TO_EBCDIC_TABLE ──────────────►◄
  (no supplied parameters)
```

The following is a conceptual scheme of the returned parameters:

```
ASCII_TO_EBCDIC_TABLE_Pointer ──────────►◄
```

Parameters

ASCII_TO_EBCDIC_TABLE_Pointer
: Returns a pointer to the table currently being used by PC Support.

Verb Record Format

Offset	Length	Value	Description
0	1	X'00'	Return code
1	1	X'08'	Get_ASCII_TO_EBCDIC pointer
2	4		Pointer to translation table

Get_EBCDIC_TO_ASCII_TABLE

Purpose

Returns a pointer to the EBCDIC-TO-ASCII translation vector maintained by the router. This translation vector allows you to use an EBCDIC character as an index to retrieve the corresponding ASCII code point.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:

```
Get_EBCDIC_TO_ASCII_TABLE ──────────────────►◄
  (no supplied parameters)
```

The following is a conceptual scheme of the returned parameters:

```
EBCDIC_TO_ASCII_TABLE_Pointer ──────────────►◄
```

Parameters

EBCDIC_TO_ASCII_TABLE_Pointer
: Returns a pointer to the table currently being used by PC Support

Verb Record Format

Offset	Length	Value	Description
0	1	X'00'	Return code
1	1	X'09'	Get_EBCDIC_TO_ASCII pointer
2	4		Pointer to translation table

Get_System_List

Purpose

Returns the list of systems to which the router is currently connected.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:

```
Get_System_List ────────────────────────────►◄
```

The following is a conceptual scheme of the returned parameters:

```
List_Of_Remote_Systems ─────────────────────►◄
```

Parameters

List_Of_Remote_Systems
: The names of all systems connected to the router are returned. Up to 32 systems can be connected to the router.

  - Each name is 8 bytes in EBCDIC. If the system name contains fewer than 8 characters, the rest are filled with blanks.
  - The first name is the default system name.
  - The list ends with a byte of 0.

Note: An empty list is returned if the router is not connected to any system.

Verb Record Format

Offset	Length	Value	Description
0	1	X'00'	Return code
1	1	X'07'	Get location list
2	256		Array of 32 8-byte fields for location names

Get_Fully_Qualified_System_List

Purpose

Returns a list of fully qualified system names. The router has an active connection to each of these systems. Each fully qualified system name includes:

- Alias
- Network ID
- System name

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:

```
Get_Fully_Qualified_System_List ──────────►◄
```

The following is a conceptual scheme of the returned parameters:

```
List_Of_Fully_Qualified_System_Names ──────────►◄
```

Parameters

List_Of_Fully_Qualified_System_Names
: The names of all systems connected to the router are returned. Up to 32 systems can be connected to the router.

  - Each name is 24 bytes in EBCDIC. If a value is shorter than the supplied field, the remainder of the field is filled with blanks.
  - The first name is the default system name.
  - The list ends with a byte of 0.

Note: An empty list is returned if the router is not connected to any system.

Verb Record Format

Offset	Length	Value	Description
0	1	X'00'	Return code
1	1	X'1B'	Get_Fully_Qualified_System_List
2	768		Array of 32 24-byte fields with the following format:

Length	Description
8	Alias
8	Network ID
8	System name

Query_Router_Capabilities

Purpose

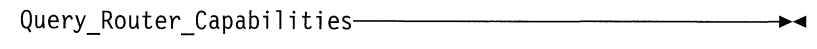
Allows you to query the router to determine the router's capabilities.

To issue this verb, do the following:

1. Set the request code to X'17'.
2. Set the *Length of verb record* field to the size of your verb record. This length includes the return code, request code, length field, and capability list.
3. For each capability in the capability list, the following fields need to be set:
   - Length. The length includes the length of the entire capability: length field, identifier field, return code field, and data field.
   - Identifier.
4. Set the AH register to X'01'.
5. Set the DS:DX to point to the verb record.
6. Call the router.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:

Parameters

Capabilities_List
: Returns a pointer to the list of router capabilities.

Verb Record Format

The Capabilities_List variable points to a list of router capabilities. This list consists of a header followed by one or more capabilities:

HEADER CAPABILITY$_1$ CAPABILITY$_2$ . . . CAPABILITY$_n$

The format of the header is:

Offset	Length	Value	Description
0	1		Return Code. This field contains a binary value.
		X'00'	Information describing each of the capabilities was returned.
		X'01'	One or more of the requested capabilities is not supported by the router.
1	1	X'17'	Request code. This field contains a binary value.
2	2		Length of the entire verb structure. This field is treated as an unsigned integer.

The following table shows the format of each of the capability records. The Offset column indicates the number of bytes from the previous capability record, or from the header if this is the first capability record.

Offset	Length	Value	Description
0	2		Length. This field is treated as an unsigned integer. This length includes this field, plus the length of the following: • Identifier • Return code • Data
2	1		Identifier of the capability. This field contains a binary value that uniquely identifies the capability. The possible values are shown on Table 4-1 on page 4-6.
3	1		Return code. This field contains a binary value.
		X'00'	Information describing the capability was successfully returned.
		X'01'	The router does not understand the query.
4			Data. The length and value of the data depends on the capability queried. See Table 4-1 on page 4-6.

Table 4-1 on page 4-6 shows the capabilities you can query using Query_Router_Capabilities.

Table 4-1. Router Capabilities

Length	Identifier	Value	Description
2	X'02'		EHNAPPC_CAP_OPTIMAL_COM_SIZE The communications buffer multiplier (maximum frame size). This value depends on the router that is in use. The value is a two-byte binary value.
1	X'03'		EHNAPPC_CAP_QUERY_CONV_STATE Determines whether or not you can use the EHNAPPC_QUERYCONVSTATE call.
		X'00'	The capability is supported.
		X'01'	The capability is not supported.
1	X'04'		EHNAPPC_CAP_EXT_ALLOCATE Determines whether or not you can use the EHNAPPC_EXTENDED_ALLOCATE call.
		X'00'	The capability is supported.
		X'01'	The capability is not supported.
1	X'05'		EHNAPPC_QUERY_FULL_SYS Determines whether or not you can use the Get_Fully_Qualified_System_Name service verb.
		X'00'	The capability is supported.
		X'01'	The capability is not supported.

Using the Communications Buffer Multiplier: In general, the larger the communications buffer, the better your performance will be. However, DOS and the architecture limit the amount you can use. To make the most efficient use of DOS memory, you should always make your buffer size a multiple of the communications buffer multiplier.

The following formula shows precisely how to calculate the size of the communications buffer:

```
Optimal_size = communications_buffer_multiplier *
               MIN(63, biggest_buffer_size_you_can_spare /
               communications_buffer_multiplier)
```

This biggest_buffer_you_can_spare value is a two-byte unsigned binary value that you supply; you must decide how much memory you can spare for a buffer. MIN is a function that returns the lesser of two unsigned two-byte integers. "/" is used to indicate integer division. 63 is the SNA architected maximum pacing value.

Querying the Router Capabilities: For the EHNAPPC_CAP_QUERY_CONV_STATE and EHNAPPC_CAP_EXT_ALLOCATE calls, the return codes indicate whether or not the function is supported by the version of the router currently loaded. These capabilities are supported in the V2R1.1 version of the router.

You should query the router capabilities before attempting to use an extended allocate call or a query conversation state call. If you attempt to use an extended allocate call with an unsupported version of the router, the results will be unpredictable.

Basic Router Conversation Verbs

The following sections for basic conversation verbs describe in detail:

- Purpose
- Syntax
- Supplied parameters and their options
- Returned parameters and their options
- Conversation state changes
- Any additional information
- Verb record formats

In a basic conversation, transaction programs exchange logical records that contain a 2-byte length prefix called an *LL field*. The LL field is formatted in hexadecimal by the transaction program. The program does not supply this 2-byte prefix for a mapped conversation.

Each block of data to be sent by a transaction program (for example, each record in a database file) is usually formatted as a single logical record. Your transaction program must provide an LL field at the head of each data block. This length specifies the sum of the length of the data and its own length, which is 2 bytes. For example, to send 12 bytes of data, set the LL field to 14 (2 + 12).

The LL field must be stored in most significant byte first (MSB) format. This field **must not** be in Intel byte-swapped format in the data buffer. A record length of decimal 14 must be placed in the data buffer as X'000E' and not as X'0E00' (which would be interpreted by the partner as 14 * 256 = 3584 bytes). In the C language, for example, a 2-byte numeric field is often defined as *unsigned*; however, using *unsigned* for the LL field results in the bytes getting swapped in memory.

A single data buffer may contain multiple logical records. A single logical record cannot be longer than 32767 bytes. If the last block of received data contained a full logical record and no more, the conversation is said to be on a logical record boundary.

A transaction program cannot use basic and mapped conversation verbs on the same conversation.

Information about the conversation states and the verbs that can be issued during each state follows.

Return Codes for Basic Conversation Verbs

All of the router verbs have return codes that supply information about the success of verb processing or provide error information. These are described in the "Returned Parameters" section for each verb.

The error information is provided in a 2-byte primary code that identifies the error type and a 4-byte secondary code that provides more detailed error information. The return codes are stored with the most significant byte first (not in Intel byte-swapped format). See the error code entries of each verb record for more details.

Some return codes indicate the result of processing by the local LU and are returned in the verb that initiated the processing. Return codes that indicate the

results of processing initiated by the partner LU are returned either in the initiating verb or on a subsequent one. The router returns only one code at a time. The "Returned Parameters" section of the following verb descriptions indicate the conditions under which each return code value is returned.

The "Returned Parameters" sections also provide information about the state changes that result from verb processing. For information about which verbs a program can issue in each state, see Appendix B, "Table of Router Conversation States."

If the return code for a verb is OK, other returned parameter information may follow. This is information that the transaction program requested. If the return code is other than OK, additional parameter information does not follow.

A description of each basic conversation verb follows. An overview of the verb descriptions is given in "Verb Descriptions" on page 3-10. For additional information about reading the diagrams that describe the supplied and returned parameters, see "Syntax" on page 3-10.

ALLOCATE

Purpose

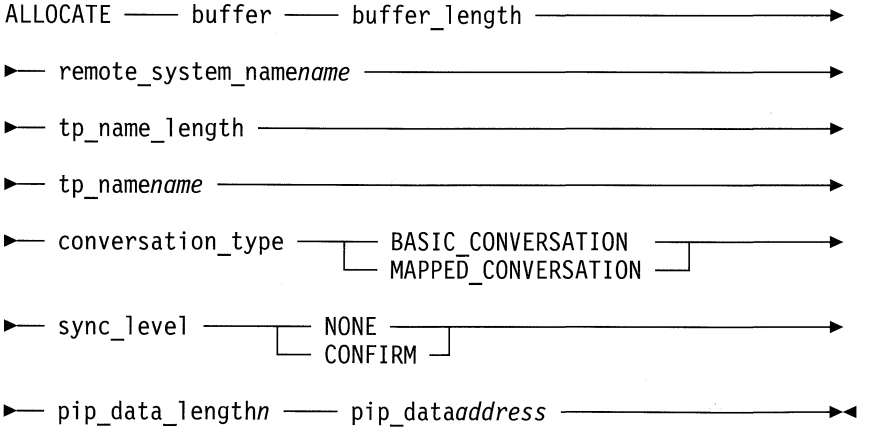
Allocates a session between the local LU and a partner LU, and establishes a conversation in the session between the local transaction program and a partner transaction program.

An ID is assigned to the new conversation between the local transaction program and the new partner transaction program.

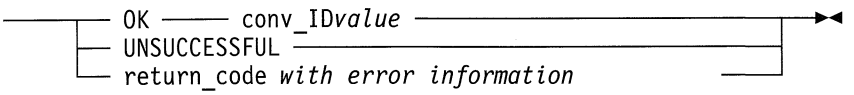
ALLOCATE must be the first conversation verb issued by a local transaction program for a conversation it initiates.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

buffer Specifies the address of a buffer in conventional memory that the router uses to accumulate data for send and receive operations. Each conversation has its own buffer. The buffer must be a minimum of 271 bytes long and it cannot cross a 64KB segment boundary. This means that the offset portion of the address plus the length of the buffer cannot be greater than 64KB.

In general, the larger this buffer is, the better your program's performance will be.

The buffer must be located in conventional memory. It cannot be in extended or expanded memory.

buffer_length
The length of the buffer supplied to the router.

remote_system_name*name*
The name of the remote system where your partner transaction program will run.

If this field is set to X'0', the default remote system is used.

The remote system name is also called the partner location name or the partner LU name.

tp_name_length
The length of the tp_name.

tp_name*name*
The EBCDIC name of the partner transaction program that manages the partner-end of the conversation.

This is a type AE table name, which is an EBCDIC character string consisting of one or more:

- Lowercase letters a through z
- Uppercase letters A through Z
- Numbers 0 through 9
- Four special characters: $, #, @, and the period (.)

conversation_type
The type of conversation the router is to allocate:

BASIC_CONVERSATION
Directs the router to allocate a basic conversation.

MAPPED_CONVERSATION
Directs the router to allocate a mapped conversation.

sync_level
The synchronization level that the local and partner programs can use:

NONE
Transaction programs cannot issue any verbs or recognize returned parameters related to synchronization.

CONFIRM
: Transaction programs can issue verbs and recognize returned parameters related to synchronization.

pip_data_length*n*
: The length of the program initialization parameters (PIPs) for the partner program. Set this parameter to 0 if there is no PIP data. PIP data cannot be longer than 168 bytes.

pip_data*address*
: The address containing the PIP data (consisting of a single LL) that the local program sends to the partner program. The transaction program must format PIP data as follows:

  0–1 Length (4 or n+1) in binary of the PIP variable, including this length field

  2–3 General data stream (GDS) indicator: X'12F5'

  4–n Zero or more PIP subfields; each of which has the following format (shown using zero-origin):

    0–1 Length in binary of PIP subfield, including this length field

    2–3 General data stream (GDS) indicator: X'12E2'

    4–n PIP subfield data

  Program initialization parameters (PIPs) are a means of passing initial parameters to a target transaction program.

  PIP data must be sent to partner transaction programs that require program initialization parameters.

Parameters

The following return code indicates that the verb ran successfully:

OK
: The OK return code for the ALLOCATE verb indicates only that a session with the remote system is now available. This return code does not indicate anything about the tp_name parameter in the ALLOCATE verb. For example, your program does not yet know whether the tp_name will be recognized by the partner or whether the remote TP even can be started.

  The following parameter is returned to the transaction program by the router if the return code is OK:

  conv_id The 4-byte identifier of the new conversation

The following return code indicates the verb ran, but was unsuccessful:

UNSUCCESSFUL
: There were no sessions available, so the router could not allocate a conversation.

The following return codes provide error information:

PARAMETER_CHECK
: The verb did not run because of a parameter error caused by one of the following:

  BAD_SYNC_LEVEL
  : The router does not recognize the specified sync_level.

INVALID_DATA_SEGMENT
The PIP data extends beyond the end of the data segment. Either the pip_data_length is too large or the pip_data*address* is wrong.

PIP_LEN_INCORRECT
The data is longer than 168 bytes.

TP_NAME_LENGTH_ERROR
The length of the tp_name must be greater than 0 and less than 65.

The remaining return codes (listed in Table 4-2) are described in "Verb Return Codes" on page 4-45.

State Change

If the return code indicates OK, the local conversation enters Send state and the partner conversation enters Receive state.

Additional Information

When an ALLOCATE verb establishes a mapped conversation, the local transaction program can issue basic conversation verbs if the program has its own mapping layer.

Note that the DS and DX register should point to 8 bytes past the start of the ALLOCATE verb (byte 0 of the verb record format).

Verb Record Format

Table 4-2 (Page 1 of 2). ALLOCATE Record Format

Offset	Length	Value	Description
-8	02	AB	ASCII characters to identify router allocate record (see note 1)
-6	04		Address of router buffer (see note 1)
-2	02		Length of router buffer (must be at least 271 bytes)
0	12		Not used by the router
12	1	X'01'	Allocate verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Codes**
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0014'	UNSUCCESSFUL indicates a session could not be started because the remote system rejected the BIND request.
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000004'	ALLOCATION_FAILURE_NO_RETRY indicates the maximum number of conversations has been exceeded or the router has been stopped by the STOPRTR command. The condition is permanent and the program should not try the allocation request again.
		X'00000005'	ALLOCATION_FAILURE_RETRY indicates the conversation cannot be allocated because the connection with the host system has been interrupted. The condition is temporary and the program can try the allocation request again.

Table 4-2 (Page 2 of 2). ALLOCATE Record Format

Offset	Length	Value	Description
		X'00000006'	INVALID_DATA_SEGMENT indicates the PIP data address and length crosses a 64K segment boundary.
		X'00000010'	TP_NAME_LENGTH_ERROR
		X'00000011'	BAD_CONV_TYPE
		X'00000012'	BAD_SYNC_LEVEL
		X'00000016'	PIP_LEN_INCORRECT
		X'00000018'	UNKNOWN_PARTNER_SYSTEM
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID (set by the router)
38	1		Conversation type
		X'00'	Basic conversation
		X'01'	Mapped conversation (see note 2)
39	1		Sync level (see note 3)
		X'00'	None
		X'01'	Confirm
40	11	X'00' (11 bytes)	Not used by the router
51	8		Partner Location Name in EBCDIC (see note 5)
59	8	X'00' (8 bytes)	Not used by the router
67	1		Length of target program name
68	64		EBCDIC name of target program
166	2		PIP data length (0 for no PIP, see note 4)
168	4		Address of PIP data

Notes:

1. The fields from -8 to 0 are extensions to the normal verb record format. These fields are used by the application to pass a buffer to the router to be used for building and disassembling LU 6.2 request/response units (RUs). The application points registers DS and DX at offset 0 of the record, but must have set the 8 bytes (-8 through 0) as described.
2. Mapped conversation verbs are not supported by the router. However, the router accepts a conversation type of mapped conversation and sends the attach request with the mapped conversation indicator. This allows an application to use the mapped conversation verbs by sending the correct data streams and interpreting the return codes.
3. The SYNC LEVEL field can only be NONE or CONFIRM.
4. PIP data length is limited to 168 bytes.
5. The partner location name field can be set to hexadecimal zeros, which tells the router to select the default remote system.

CONFIRM

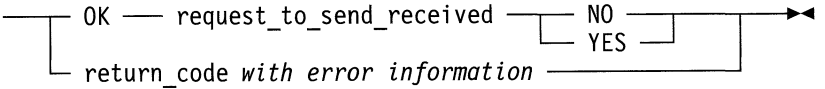
Purpose

Sends a confirmation request to a partner transaction program and waits for a reply. This verb enables the local and partner programs to synchronize their processing with one another. The CONFIRM verb also causes the router to flush its send buffer. To use this verb, the program must have allocated the conversation with a synchronization level of CONFIRM.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:

The following is a conceptual scheme of the returned parameters:



Parameters

conv_id*value*
: The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for the conversation.

Parameters

The following indicates that the verb ran successfully:

OK
: The verb ran successfully.

The following parameters are returned to the transaction program by the router if the return code is OK:

request_to_send_received
: Indicates whether the partner transaction program issued a REQUEST_TO_SEND verb. The router can return either YES or NO.

  YES
  : The router received an indicator that the partner program issued a REQUEST_TO_SEND verb, which requests the local program to enter Receive state and place the partner program in Send state.

  NO
  : The router has not received such an indicator from the partner transaction program.

  The following return codes provide error information:

  PARAMETER_CHECK
  : The verb did not run because of a parameter error caused by one of the following:

    BAD_CONV_ID
    : The router does not recognize the specified conv_id.

    CONFIRM_ON_SYNC_LEVEL_NONE
    : The router does not permit the transaction program to use this verb if the synchronization level of the conversation is NONE.

  STATE_CHECK
  : The verb was not run because it is in the wrong state.

    STATE
    : The conversation is not in Send state.

The remaining return codes (listed in Table 4-3 on page 4-14) are described in "Verb Return Codes" on page 4-45.

State Change

The state of the conversation does not change when the return code indicates OK.

Additional Information

- The transaction program can use this verb for various application-level functions. For example:
  - The transaction program can issue this verb immediately following an ALLOCATE verb to determine whether the allocation of the conversation is successful before sending data.
  - The transaction program can issue this verb to request acknowledgment of data it sent to the remote transaction program. The partner program can respond by issuing a CONFIRMED verb to indicate that it received and processed the data without error or by issuing a SEND_ERROR verb to indicate that it found an error.
- When the request_to_send_received parameter indicates YES, the partner transaction program requests that the local transaction program enter Receive state and place the partner transaction program in Send state. A transaction program enters Receive state by issuing the PREPARE_TO_RECEIVE or the RECEIVE AND WAIT verb. A partner transaction program enters Send state after it issues the RECEIVE_AND_WAIT or the RECEIVE_IMMEDIATE verb and receives a send indicator in the what_received parameter.

Verb Record Format

Table 4-3 (Page 1 of 2). CONFIRM Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'03'	Confirm verb operation code
12	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK (remote program replied CONFIRMED)
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0006'	DEALLOC_ABEND_PROG
		X'0007'	DEALLOC_ABEND_SVC
		X'0008'	DEALLOC_ABEND_TIMER
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0013'	SVC_ERROR_PURGING
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'00000004'	ALLOCATION_FAILURE_NO_RETRY
		X'00000005'	ALLOCATION_FAILURE_RETRY
		X'00000031'	CONFIRM_ON_SYNC_LEVEL_NONE
		X'000000F2'	SEND_DATA_NOT_SEND_STATE
		X'080F6051'	SECURITY_NOT_VALID
		X'084B6031'	TRANS_PGM_NOT_AVAIL_RETRY
		X'084C0000'	TRANS_PGM_NOT_AVAIL_NO_RETRY

Table 4-3 (Page 2 of 2). CONFIRM Record Format

Offset	Length	Value	Description
		X'10086021'	TP_NAME_NOT_RECOGNIZED
		X'10086031'	PIP_NOT_ALLOWED
		X'10086032'	PIP_NOT_SPECIFIED_CORRECTLY
		X'10086034'	CONVERSATION_TYPE_MISMATCH
		X'10086041'	SYNC_LEVEL_NOT_SUPPORTED
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID
38	1		Request to send received
		X'00'	No
		X'01'	Yes

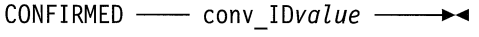
CONFIRMED

Purpose

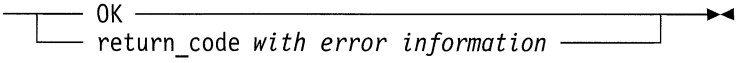
Sends a confirmation reply to the partner transaction program. This verb enables the local and partner programs to synchronize their processing. The local program can issue this verb only when it receives a confirmation request on the what_received parameter of Receive verbs.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*
: The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for the conversation.

Parameters

The following indicates that the verb ran successfully:

OK
: The verb ran successfully.

The following return codes provide error information:

PARAMETER_CHECK
: The verb did not run because of a parameter error.

  BAD_CONV_ID
  : The router does not recognize the specified conv_ID.

STATE_CHECK
The verb was not run because it is in the wrong state.

CONFIRMED_BAD_STATE
The conversation is not in Confirm state.

The remaining return codes (listed in Table 4-4) are described in "Verb Return Codes" on page 4-45.

State Change

The conversation enters Receive state if the transaction program received the CONFIRM value in the what_received parameter of the preceding RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb.

The conversation enters Send state if the transaction program received the CONFIRM_SEND value in the what_received parameter of the preceding RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb.

The conversation enters Reset state if the transaction program received the CONFIRM_DEALLOCATE value in the what_received parameter of the preceding RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb.

Verb Record Format

Table 4-4. CONFIRMED Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'04'	Confirmed verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK (remote program replied CONFIRMED)
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'00000041'	CONFIRMED_BAD_STATE
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID

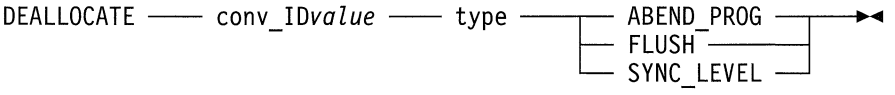
DEALLOCATE

Purpose

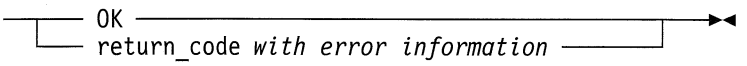
Deallocates the specified conversation. Can perform the function of the FLUSH or the CONFIRM verb before it deallocates the conversation.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*
: The identifier of the conversation in which the verb is being issued. This value is returned in an ALLOCATE verb for the conversation.

type
: The type of deallocation the router is to perform.

  ABEND_PROG
  : The router deallocates the conversation abnormally. If the conversation was in Send state, the router first performs the function of the FLUSH verb. The router can perform logical record truncation of the conversation if it is in Send state. If the conversation is in Receive state, the router can remove data but logical record truncation may occur.

    A transaction program can specify type as ABEND_PROG.

    A transaction program can use type(ABEND_PROG) when it detects an error condition that prevents successful completion of the transaction. The specific use and meaning of ABEND_PROG are left to the designer of the transaction programs.

  FLUSH
  : The router performs the function of the FLUSH verb and then deallocates the conversation.

  SYNC_LEVEL
  : The router deallocates the conversation based on the sync_level allocated to this conversation.

    If the synchronization level of the conversation is NONE, the router performs the function of the FLUSH verb and then deallocates the conversation.

    If the synchronization level of the conversation is CONFIRM, the router performs the function of the CONFIRM verb and deallocates the conversation if OK is returned. Otherwise, the return code determines the state of the conversation.

Parameters

The following indicates that the verb ran successfully:

OK
: The verb ran successfully. The conversation ID (conv_id) is no longer valid.

The following return codes provide error information:

PARAMETER_CHECK
: The verb did not run because of a parameter error caused by the following:

  BAD_CONV_ID
  : The router does not recognize the specified conv_ID.

STATE_CHECK
The verb did not run because it is in the wrong state due to one of the following reasons:

DEALLOC_FLUSH_BAD_STATE
The program specified type(SYNC_LEVEL) for a conversation that has a synchronization level of NONE and is not in Send state. Alternatively, the program may have specified type(FLUSH) when the conversation was not in Send state.

DEALLOC_CONFIRM_BAD_STATE
The program specified type(SYNC_LEVEL) for a conversation that has a synchronization level of CONFIRM when the conversation was not in Send state.

If your program specifies type(SYNC_LEVEL) and the synchronization level of this conversation is CONFIRM, the router can report one of the remaining return codes listed in Table 4-5. For detailed information about these return codes, see "Verb Return Codes" on page 4-45.

State Change

The conversation enters Reset state when the return code indicates OK. For information about state changes when the return code indicates other than OK, see Appendix B, "Table of Router Conversation States."

Additional Information

- A transaction program can specify type(SYNC_LEVEL) to deallocate the conversation based on the synchronization level allocated to the conversation.
  - If sync_level is NONE, the router deallocates the conversation unconditionally.
  - If sync_level is CONFIRM, the router deallocates the conversation if the partner transaction program issues a CONFIRMED verb to respond to the confirmation request. The conversation remains allocated if the partner transaction program issues a SEND_ERROR verb to respond to the confirmation request.
- A transaction program can specify type(FLUSH) to deallocate the conversation unconditionally, regardless of its synchronization level.

Verb Record Format

Table 4-5 (Page 1 of 2). DEALLOCATE Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'05'	Deallocate verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0006'	DEALLOC_ABEND_PROG
		X'0007'	DEALLOC_ABEND_SVC
		X'0008'	DEALLOC_ABEND_TIMER
		X'000E'	PROG_ERROR_PURGING

Table 4-5 (Page 2 of 2). DEALLOCATE Record Format

Offset	Length	Value	Description
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0013'	SVC_ERROR_PURGING
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'000000F2'	SEND_DATA_NOT_SEND_STATE
		X'080F6051'	SECURITY_NOT_VALID
		X'084B6031'	TRANS_PGM_NOT_AVAIL_RETRY
		X'084C0000'	TRANS_PGM_NOT_AVAIL_NO_RETRY
		X'10086021'	TP_NAME_NOT_RECOGNIZED
		X'10086031'	PIP_NOT_ALLOWED
		X'10086032'	PIP_NOT_SPECIFIED_CORRECTLY
		X'10086034'	CONVERSATION_TYPE_MISMATCH
		X'10086041'	SYNC_LEVEL_NOT_SUPPORTED
26	8	X'00' (12 bytes)	Not used by the router
34	4		Conversation ID
38	1	X'00'	Not used by the router
39	1		Deallocate type
		X'00'	Sync level
		X'01'	Flush
		X'02'	Abend program

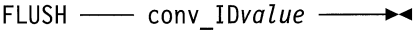
FLUSH

Purpose

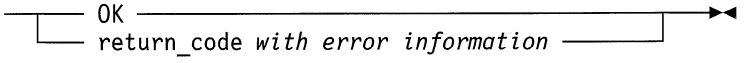
Flushes the send buffer of the router by sending all buffered information to the partner LU. Information buffered by the router can come from the ALLOCATE, DEALLOCATE, or SEND_DATA verbs.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*
: The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for the conversation.

Parameters

The following indicates that the verb ran successfully:

OK The verb ran successfully.

The following return codes provide error information:

PARAMETER_CHECK
: The verb did not run because of a parameter error caused by the following:

  BAD_CONV_ID
  : The router does not recognize the specified conv_id.

STATE_CHECK
: The verb did not run because it is in the wrong state due to the following reason:

  SEND_DATA_NOT_SEND_STATE
  : The conversation is not in Send state.

The remaining return codes (listed in Table 4-6) are described in "Verb Return Codes" on page 4-45.

Additional Information

Use this verb to reduce the waiting time for data between the local and partner transaction programs. The router normally buffers the data from consecutive SEND_DATA verbs until it has a sufficient amount for transmission. However, the local transaction program can issue the FLUSH verb to force the router to send the buffered data.

Verb Record Format

Table 4-6. FLUSH Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'06'	Flush verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'000000F2'	SEND_DATA_NOT_SEND_STATE
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID

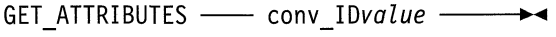
GET_ATTRIBUTES

Purpose

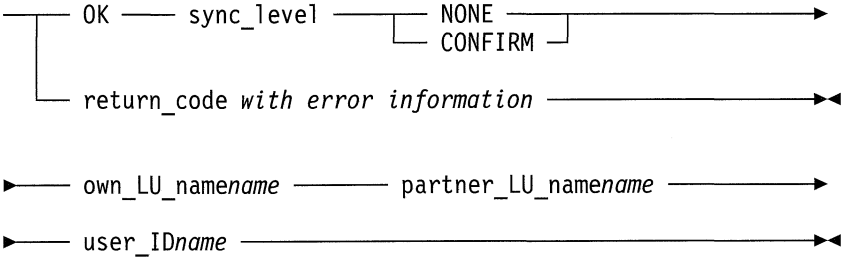
Returns attributes of the specified conversation, including the LU names of the local and partner transaction programs, the level of processing synchronization, and any user ID provided for security.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*
: The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for the conversation.

Parameters

The following indicates that the verb ran successfully:

OK
: The verb ran successfully.

The following parameters are returned to the transaction program by the router if the return code is OK:

own_LU_name*name*
: The EBCDIC name of the LU of the local transaction program. This LU name was configured in the RTLN value of a PC Support configuration file (CONFIG.PCS).

  This is a type A table name, which is an EBCDIC character string consisting of one or more:

  - Uppercase letters A through Z
  - Numbers 0 through 9
  - Three special characters: $, #, and @

partner_LU_name*name*
: The EBCDIC name of the partner.

  This is a type AE table name, which is an EBCDIC character string consisting of one or more:

  - Lowercase letters a through z
  - Uppercase letters A through Z
  - Numbers 0 through 9
  - Four special characters: $, #, @, and the period (.)

sync_level
: The level of synchronization processing specified for the conversation. For information about a conversation's level of synchronization, see the description of the sync_level parameter of the ALLOCATE verb. The synchronization levels are:

  NONE
  : This conversation cannot issue any verbs or recognize returned parameters related to synchronization.

  CONFIRM
  : This conversation can issue verbs and recognize returned parameters related to synchronization.

user_ID*name*
: The EBCDIC user ID that was used to establish this connection.

  This is a type AE table name, which is an EBCDIC character string consisting of one or more:

  - Lowercase letters a through z
  - Uppercase letters A through Z
  - Numbers 0 through 9
  - Four special characters: $, #, @, and the period (.)

  If this name is less than 10 characters long, the router pads it on the right with EBCDIC blanks.

The following return code provides error information:

PARAMETER_CHECK
: The verb did not run because of the following parameter error:

  BAD_CONV_ID
  : The router does not recognize the specified conv_id.

The remaining return codes (listed in Table 4-7 on page 4-23) are described in "Verb Return Codes" on page 4-45.

Additional Information

The following parameters are returned in EBCDIC:

LU_name
partner_LU_name
user_ID

Verb Record Format

Table 4-7. GET_ATTRIBUTES Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'07'	Get attributes verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK
		X'0001'	PARAMETER_CHECK
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID
38	9	X'00' (9 bytes)	Not used by the router
47	1		Sync level of conversation
48	16	X'00' (16 bytes)	Not used by the router
64	8		Local system name in EBCDIC
72	8		Remote system name in EBCDIC
80	19	X'00' (19 bytes)	Not used by the router
99	1		Length of user ID (binary)
100	10		User ID in EBCDIC

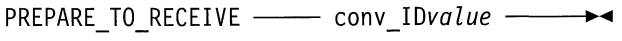
PREPARE_TO_RECEIVE

Purpose

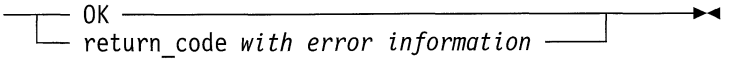
Changes a basic conversation from Send to Receive state in preparation to receive data.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*

The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for a locally initiated conversation or in a RECEIVE_ALLOCATE verb for a remotely initiated conversation.

Return Codes

The following indicates that the verb ran successfully:

OK The verb ran successfully.

The following return codes provide error information:

PARAMETER_CHECK
: The verb did not run because of the following parameter error:

  BAD_CONV_ID
  : The router does not recognize the specified conv_ID.

STATE_CHECK
: The verb did not run because it is in the wrong state.

  SEND_DATA_NOT_SEND_STATE
  : The conversation is not in Send state.

The remaining return codes (listed in Table 4-8 on page 4-25) are described in "Verb Return Codes" on page 4-45.

State Change

If the return code indicates OK, the local conversation enters Receive state. For information about state changes when the return code indicates an error condition, see Appendix B, "Table of Router Conversation States."

Additional Information

The conversation at the partner transaction program enters Send state when the partner transaction program issues a Receive verb and gets a send indicator in the what_received parameter. The partner transaction program can then send data to the local transaction program.

Verb Record Format

Table 4-8. PREPARE_TO_RECEIVE Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'0A'	Prepare to receive verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK (remote program replied CONFIRMED)
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0006'	DEALLOC_ABEND_PROG
		X'0007'	DEALLOC_ABEND_SVC
		X'0008'	DEALLOC_ABEND_TIMER
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'000000F2'	SEND_DATA_NOT_SEND_STATE
		X'080F6051'	SECURITY_NOT_VALID
		X'084B6031'	TRANS_PGM_NOT_AVAIL_RETRY
		X'084C0000'	TRANS_PGM_NOT_AVAIL_NO_RETRY
		X'10086021'	TP_NAME_NOT_RECOGNIZED
		X'10086031'	PIP_NOT_ALLOWED
		X'10086032'	PIP_NOT_SPECIFIED_CORRECTLY
		X'10086034'	CONVERSATION_TYPE_MISMATCH
		X'10086041'	SYNC_LEVEL_NOT_SUPPORTED
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID

RECEIVE_AND_WAIT

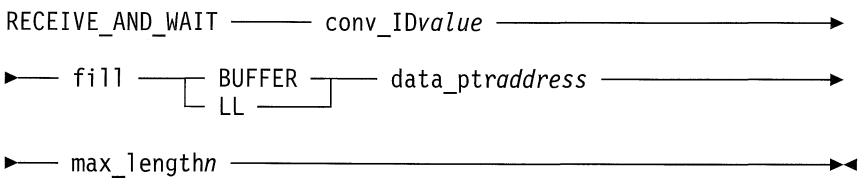
Purpose

Waits for information to arrive on the specified conversation and then receives the information or receives information already available. The information can be data, conversation status, a request for confirmation, or a combination of these. The router returns control to the transaction program and indicates the type of information.

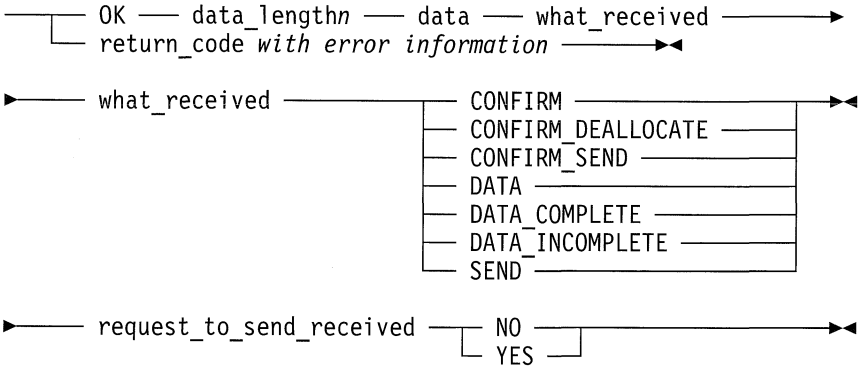
The transaction program can issue this verb when the conversation is in either Receive or Send state. If the conversation is in Send state, the router first flushes its send buffer, sending all buffered information and the send indicator to the partner transaction program. The router then waits for information to arrive from the partner transaction program. The partner transaction program sends data to the local transaction program after it receives the send indicator.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*
: The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for a locally initiated conversation or in a RECEIVE_ALLOCATE verb for a remotely initiated conversation.

fill
: The form in which the program is to receive data, by number of bytes or by logical record length.

  BUFFER
  : An amount of data equal to that specified by max_length, unless the data contains an end-of-data indicator before max_length is reached. The amount of data required for receipt by the local program is independent of the logical record format.

    The end of data is indicated by a value of SEND, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE returned for the status portion of the what_received parameter or by any return code value other than OK.

  LL
  : A complete or truncated logical record, unless an amount of data equal to max_length is received first.

data_ptr*address*
: The address of the buffer that is to contain the received data.

  The data buffer must be entirely within the data segment; therefore, the offset portion of the address plus the length of the data buffer must not exceed the segment size.

max_length*n*
: The maximum number of data bytes (from 0 to 65 535) that the transaction program can receive. The sum of this value and the offset portion of data_ptr must not exceed the size of the data segment.

Return Codes

The following indicates that the verb ran successfully:

OK
: The verb ran successfully.

DEALLOCATE_NORMAL
: The verb ran successfully.

The following parameters are returned to the transaction program by the router if the return code is OK:

data_length*n*
: The amount of the data the transaction program receives, up to the value of max_length. If no data is received by the transaction program, the value is 0.

data
: Indicates that data is received in the buffer specified by data_ptr.

what_received
: Indicates the conversation status and whether data was received. The program examines this variable only if the return code is OK.

  CONFIRM
  : The partner transaction program issued a CONFIRM verb, so the local program can respond by issuing CONFIRMED or SEND_ERROR verbs.

  CONFIRM_DEALLOCATE
  : The partner transaction program issued a DEALLOCATE verb with type specified as SYNC_LEVEL and the synchronization level of the conversation specified as CONFIRM. The local transaction program can respond by issuing CONFIRMED or SEND_ERROR verbs.

  CONFIRM_SEND
  : The partner transaction program issued the PREPARE_TO_RECEIVE verb with type specified as SYNC_LEVEL and the synchronization level of the conversation specified as CONFIRM. The local transaction program can respond by issuing CONFIRMED or SEND_ERROR verbs.

  DATA
  : The transaction program received a max_length amount of data or the end of data (independent of its logical record format) after it specified fill(BUFFER).

  DATA_COMPLETE
  : A complete logical record or the last part of a logical record.

  DATA_INCOMPLETE
  : Less than a complete logical record. Use one or more of the Receive verbs to receive the rest of the record.

SEND
The partner transaction program is in Receive state, so the local transaction program is now in Send state and can issue a SEND_DATA verb.

request_to_send_received
Indicates whether the partner transaction program issued a REQUEST_TO_SEND verb. The router can return either NO or YES.

NO The router has not received the REQUEST_TO_SEND verb from the partner transaction program.

YES
The router received an indicator that the partner program issued a REQUEST_TO_SEND verb, which requests the local program to both enter Receive state and place the partner program in Send state.

The following return codes provide error information:

PARAMETER_CHECK
The program did not run because of the following parameter error:

BAD_CONV_ID
The router does not recognize the specified conv_id.

STATE_CHECK
The verb did not run because it is in the wrong state for the following reason:

RCV_AND_WAIT_BAD_STATE
The conversation is in the wrong state. See Appendix B, "Table of Router Conversation States," for a list of possible states.

The remaining return codes (listed in Table 4-9 on page 4-30) are described in "Verb Return Codes" on page 4-45.

State Change

If the return code indicates OK, the local conversation enters:

- Receive state if the verb is issued in Send state and the what_received parameter specifies DATA, DATA_COMPLETE, or DATA_INCOMPLETE.
- Send state when the what_received parameter specifies SEND.
- Confirm state when the what_received parameter specifies CONFIRM or CONFIRM_DEALLOCATE.
- No state change if the verb is issued in Receive state and the what_received parameter specifies DATA, DATA_COMPLETE, or DATA_INCOMPLETE.

Additional Information

- DATA_INCOMPLETE cannot be returned with any of the what_received parameters; it is always returned by itself. The next Receive verb (following the one that produced DATA_INCOMPLETE) returns either DATA or a return code other than OK. In this case, ignore the what_received parameter.
- When the transaction program issues the RECEIVE_AND_WAIT verb in the Send state, the LU implicitly runs a PREPARE_TO_RECEIVE verb before it

runs the RECEIVE_AND_WAIT verb. See the description of the PREPARE_TO_RECEIVE verb on page 4-23 for more details.

- The transaction program can issue this verb when the conversation is in either Send or Receive state. If the conversation is in Send state, the LU flushes its send buffer, sending all buffered information and the send indicator to the partner transaction program. The conversation is now in Receive state. The LU then waits for information to arrive from the partner transaction program after the partner transaction program receives the send indicator.
- When the transaction program specifies fill (LL) to receive data in terms of logical records, the what_received parameter can return either the DATA_COMPLETE or the DATA_INCOMPLETE indicator depending on the data received. The two sequences of events leading to these indicators are:
  - The transaction program receives a complete logical record or the last portion of a record. The what_received parameter indicates DATA_COMPLETE.
  - The transaction program receives an incomplete logical record. The logical record is incomplete because:
    - The logical record contains more data than the program specifies in the max_length parameter, so the program receives a portion of the logical record equal to the specified length.
    - Only a portion of the logical record is available because the partner transaction program truncated it. The length of the truncated portion is less than or equal to the length the program specified in the max_length parameter.

    In these two cases, the what_received parameter indicates DATA_INCOMPLETE. The transaction program issues another Receive verb (or more than one) to receive the remainder of the logical record.

    For a definition of complete and incomplete logical records, see the description of the SEND_DATA verb on page 4-38.
- The transaction program specifies fill(BUFFER) to receive data independently of its logical record format. This specification directs the program to receive whatever data is available up to the max_length amount. When the program specifies fill(BUFFER), it must perform its own tracking of the data's logical record format.
- A transaction program can use a RECEIVE_AND_WAIT verb to determine the type of information available without receiving any information. To do this, specify max_length as 0 and examine the return code and the what_received parameters. If fill was specified as LL and data is available, what_received indicates DATA_INCOMPLETE. If fill was specified as BUFFER and data is available, what_received indicates DATA.
- If the partner transaction program truncated a logical record, the local transaction program learns of this by receiving DATA_INCOMPLETE on the first RECEIVE_AND_WAIT verb and SEND or CONFIRM on the next Receive verb.
- The local transaction program usually receives a REQUEST_TO_SEND verb when it is in Send state. The router reports the verb with a SEND_DATA verb or with a SEND_ERROR verb it issues in Send state. However, the program can receive the REQUEST_TO_SEND verb, when its conversation is in Receive state, under the following conditions:

- When the local transaction program enters Receive state and the remote transaction program issues the REQUEST_TO_SEND verb before it enters the Send state.
- When the remote transaction program issues the PREPARE_TO_RECEIVE verb (but not the RECEIVE_AND_WAIT verb) in Send state, the remote transaction program transfers send control to the local transaction program, and then issues the REQUEST_TO_SEND verb before the local transaction program enters Send state. This condition can occur because the REQUEST_TO_SEND verb is an expedited request and this expedited request can arrive ahead of the request carrying the send indication.
- The local transaction program might not be able to distinguish this condition from the preceding case. The remote transaction program can avoid this condition by waiting until it receives information from the local transaction program before it issues the REQUEST_TO_SEND verb.
- When the remote transaction program issues the REQUEST_TO_SEND verb in Send state.

- The offset portion of the data buffer address plus the length must not exceed the segment size; that is, the data buffer must be entirely within the data segment.
- The RECEIVE_AND_WAIT verb waits for data or status to be received. To cancel the verb, send a DEALLOCATE verb from the partner program.

Verb Record Format

Table 4-9 (Page 1 of 2). RECEIVE_AND_WAIT Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'0B'	Receive and wait verb operation code
13	7	X'00'	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK (remote program replied CONFIRMED)
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0006'	DEALLOC_ABEND_PROG
		X'0007'	DEALLOC_ABEND_SVC
		X'0008'	DEALLOC_ABEND_TIMER
		X'0009'	DEALLOC_NORMAL
		X'000C'	PROG_ERROR_NO_TRUNC
		X'000D'	PROG_ERROR_TRUNC
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0011'	SVC_ERROR_NO_TRUNC
		X'0012'	SVC_ERROR_TRUNC
		X'0013'	SVC_ERROR_PURGING
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'00000004'	ALLOCATION_FAILURE_NO_RETRY
		X'00000005'	ALLOCATION_FAILURE_RETRY
		X'00000006'	INVALID_DATA_SEGMENT
		X'000000F2'	SEND_DATA_NOT_SEND_STATE
		X'080F6051'	SECURITY_NOT_VALID

Table 4-9 (Page 2 of 2). RECEIVE_AND_WAIT Record Format

Offset	Length	Value	Description
		X'084B6031'	TRANS_PGM_NOT_AVAIL_RETRY
		X'084C0000'	TRANS_PGM_NOT_AVAIL_NO_RETRY
		X'10086021'	TP_NAME_NOT_RECOGNIZED
		X'10086031'	PIP_NOT_ALLOWED
		X'10086032'	PIP_NOT_SPECIFIED_CORRECTLY
		X'10086034'	CONVERSATION_TYPE_MISMATCH
		X'10086041'	SYNC_LEVEL_NOT_SUPPORTED
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID
38	1		**What received value**
		X'00'	Data
		X'01'	Data complete
		X'02'	Data incomplete
		X'03'	Confirm
		X'04'	Confirm send
		X'05'	Confirm deallocate
		X'06'	Send indicator
39	1		Fill
		X'00'	Buffer
		X'01'	LL
40	1		Request to send received
		X'00'	No
		X'01'	Yes
41	2		Maximum length
43	2		Received data length
45	4		Received data address

RECEIVE_IMMEDIATE

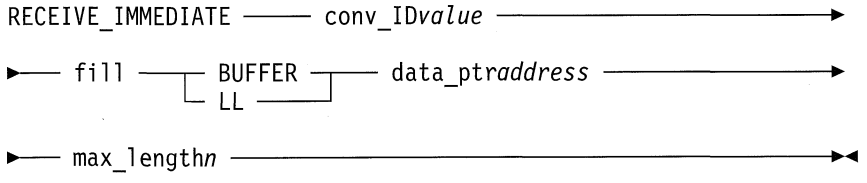
Purpose

Receives any currently available information from the specified conversation, but does not wait for information. The information can be data, conversation status, or a request for confirmation.

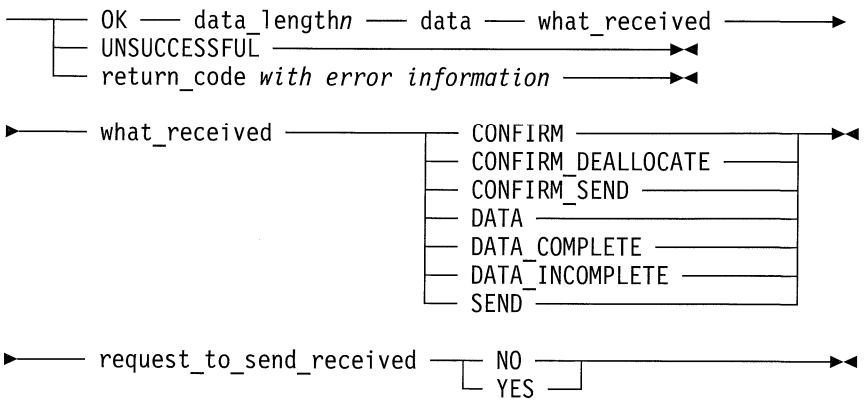
The router returns control to the transaction program and indicates whether the information was received and, if so, the type of information.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*
: The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for the conversation.

fill
: The form, by number of bytes or by logical record length, in which the program is to receive data. The options are:

  BUFFER
  : An amount of data equal to that specified by max_length, unless the data contains an end-of-data indicator before max_length is reached. The amount of data required for receipt by the local program is independent of the logical record format.

    The end of data is indicated by a value of SEND, CONFIRM, CONFIRM_SEND, or CONFIRM_DEALLOCATE returned for the status portion of the what_received parameter or by any return code value other than OK.

  LL
  : A complete or truncated logical record, unless an amount of data equal to max_length is received first.

data_ptr*address*
: The address of the buffer that is to contain the received data.

  The data buffer must be entirely within the data segment; therefore, the offset portion of the address plus the length of the data buffer must not exceed the segment size.

max_length*n*
: The maximum number of data bytes (from 0 to 65 535) that the transaction program can receive. The sum of this value and the offset portion of data_ptr must not exceed the size of the data segment.

Return Codes

The following indicates that the verb ran successfully:

OK
: The verb ran successfully.

DEALLOCATE_NORMAL
: The verb ran successfully.

The following parameters are returned to the transaction program by the router if the return code is OK:

data_length*n*
: The actual amount of data the local transaction program receives, up to the value of max_length. If no data is received by the transaction program, the value is 0.

data
: Data is received in the buffer specified by data_ptr.

what_received
: Indicates the conversation status and whether data was received. The program examines this variable only if the return code is OK.

  CONFIRM
  : The partner transaction program issued a CONFIRM verb, so the local program can respond by issuing CONFIRMED or SEND_ERROR verbs.

  CONFIRM_DEALLOCATE
  : The partner transaction program issued a DEALLOCATE verb with type specified as SYNC_LEVEL and the synchronization level of the conversation specified as CONFIRM. The local transaction program can respond by issuing CONFIRMED or SEND_ERROR verbs.

  CONFIRM_SEND
  : The partner transaction program issued the PREPARE_TO_RECEIVE verb with type specified as SYNC_LEVEL and the synchronization level of the conversation specified as CONFIRM. The local transaction program can respond by issuing CONFIRMED or SEND_ERROR verbs.

  DATA
  : The transaction program received a max_length amount of data or the end of data (independent of its logical record format) after it specified fill(BUFFER).

  DATA_COMPLETE
  : A complete logical record or the last part of a record.

  DATA_INCOMPLETE
  : Less than a complete logical record. Use one or more of the Receive verbs to receive the rest of the record.

  SEND
  : The partner transaction program is in Receive state, so the local transaction program is now in Send state and can issue a SEND_DATA verb.

request_to_send_received
: Indicates whether the partner transaction program issued a REQUEST_TO_SEND verb. The router can return either NO or YES.

  NO
  : The router has not received the REQUEST_TO_SEND verb from the partner transaction program.

  YES
  : The router received an indicator that the partner program issued a REQUEST_TO_SEND verb, which requests the local program to both enter Receive state and place the partner program in Send state.

UNSUCCESSFUL
No data is available for receipt.

The following return codes provide error information:

PARAMETER_CHECK
The program did not run because of a parameter error for the following reason:

BAD_CONV_ID
The router does not recognize the specified conv_ID.

STATE_CHECK
The verb was not run because it is in the wrong state for the following reason:

RCV_IMMD_BAD_STATE
The conversation is in the wrong state.

The remaining return codes (listed in Table 4-10 on page 4-36) are described in "Verb Return Codes" on page 4-45.

State Change

If the return code indicates OK, the local conversation enters:

- Receive state if the verb is issued in Send state and the what_received parameter specifies DATA, DATA_COMPLETE, or DATA_INCOMPLETE.
- Send state when the what_received parameter specifies SEND.
- Confirm state when the what_received parameter specifies CONFIRM or CONFIRM_DEALLOCATE.
- No state change when the what_received parameter specifies DATA, DATA_COMPLETE, or DATA_INCOMPLETE.

Additional Information

- DATA_INCOMPLETE cannot be returned with any conversation status indicators of the what_received parameter; it is always returned by itself. The next Receive verb following the one on which data_incomplete was indicated either returns more DATA or sets a return code other than OK (in which case, the what_received parameter should be ignored).
- When the transaction program specifies fill(LL) to receive data in terms of logical records, the what_received parameter can return either the DATA_COMPLETE (optionally combined with status indicators) or the DATA_INCOMPLETE indicator depending on the data received. The two sequences of events leading to these indicators are:
  - The transaction program receives a complete logical record or the last portion of a record. The length of the record or portion of the record is equal to or less than the length specified in the max_length parameter. The what_received parameter indicates DATA_COMPLETE (optionally combined with status indicators).
  - The transaction program receives an incomplete logical record. The logical record is incomplete because:

- The logical record contains more data than the program specifies in the max_length parameter, so the program receives a portion of the logical record equal to the specified length.
- Only a portion of the logical record is available because the partner transaction program truncated it or the transmission the full logical record is not yet complete. The length of the truncated portion is less than or equal to the length the program specified in the max_length parameter.

In these two cases, the what_received parameter indicates DATA_INCOMPLETE. The transaction program issues a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb (or more than one) to receive the remainder of the logical record.

For a definition of complete and incomplete logical records, see the description of the SEND_DATA verb on page 4-38.

- The transaction program specifies fill(BUFFER) to receive data independently of its logical record format. This specification directs the program to receive whatever data is available, up to the max_length amount. When it specifies fill(BUFFER), the program must perform its own tracking of the data's logical record format.
- A transaction program can use a RECEIVE_IMMEDIATE verb to determine the type of information available without receiving any information. To do this, specify max_length as 0 and examine the values in the return code and the what_received parameters. If fill is specified as LL and data is available, what_received indicates DATA_INCOMPLETE. If fill is specified as BUFFER and data is available, what_received indicates DATA.
- If the partner transaction program truncated a logical record, the local transaction program learns of this by receiving DATA_INCOMPLETE on the first RECEIVE_IMMEDIATE verb and SEND or CONFIRM, CONFIRM_DEALLOCATE, or CONFIRM_SEND on the next RECEIVE verb.
- The local transaction program usually receives a REQUEST_TO_SEND verb when it is in Send state. The router reports the verb with a SEND_DATA verb or with a SEND_ERROR verb it issues in Send state. However, the program can receive the REQUEST_TO_SEND verb when its conversation is in Receive state under the following conditions:
  - When the local transaction program enters Receive state and the remote transaction program issues the REQUEST_TO_SEND verb before it enters the Send state.
  - When the remote transaction program issues the PREPARE_TO_RECEIVE verb (but not the RECEIVE_AND_WAIT verb) in Send state, the remote transaction program transfers send control to the local transaction program, and then issues the REQUEST_TO_SEND verb before the local transaction program enters Send state. This condition can occur because the REQUEST_TO_SEND verb is an expedited request and this expedited request can arrive ahead of the request carrying the SEND indication.
  - The local transaction program might not be able to distinguish this condition from the preceding case. The remote transaction program can avoid this condition by waiting until it receives information from the local transaction program before it issues the REQUEST_TO_SEND verb.

- When the remote transaction program issues the REQUEST_TO_SEND verb in Send state.

- The offset portion of the data buffer address plus the length must not exceed the segment size; that is, the data buffer must be entirely within the data segment.

Verb Record Format

Table 4-10 (Page 1 of 2). RECEIVE_IMMEDIATE Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'0C'	Receive immediate verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK (remote program replied CONFIRMED)
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0006'	DEALLOC_ABEND_PROG
		X'0007'	DEALLOC_ABEND_SVC
		X'0008'	DEALLOC_ABEND_TIMER
		X'0009'	DEALLOC_NORMAL
		X'000C'	PROG_ERROR_NO_TRUNC
		X'000D'	PROG_ERROR_TRUNC
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0011'	SVC_ERROR_NO_TRUNC
		X'0012'	SVC_ERROR_TRUNC
		X'0013'	SVC_ERROR_PURGING
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'00000004'	ALLOCATION_FAILURE_NO_RETRY
		X'00000005'	ALLOCATION_FAILURE_RETRY
		X'00000006'	INVALID_DATA_SEGMENT
		X'000000C1'	RCV_IMMD_BAD_STATE
		X'080F6051'	SECURITY_NOT_VALID
		X'084B6031'	TRANS_PGM_NOT_AVAIL_RETRY
		X'084C0000'	TRANS_PGM_NOT_AVAIL_NO_RETRY
		X'10086021'	TP_NAME_NOT_RECOGNIZED
		X'10086031'	PIP_NOT_ALLOWED
		X'10086032'	PIP_NOT_SPECIFIED_CORRECTLY
		X'10086034'	CONVERSATION_TYPE_MISMATCH
		X'10086041'	SYNC_LEVEL_NOT_SUPPORTED
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID
38	1		**What received value**
		X'00'	Data
		X'01'	Data complete
		X'02'	Data incomplete
		X'03'	Confirm
		X'04'	Confirm send
		X'05'	Confirm deallocate
		X'06'	Send indicator
39	1		Fill
		X'00'	Buffer
		X'01'	LL

Table 4-10 (Page 2 of 2). RECEIVE_IMMEDIATE Record Format

Offset	Length	Value	Description
40	1		Request to send received
		X'00'	No
		X'01'	Yes
41	2		Maximum length
43	2		Received data length
45	4		Received data address

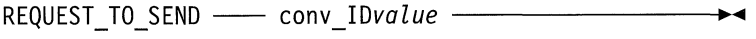
REQUEST_TO_SEND

Purpose

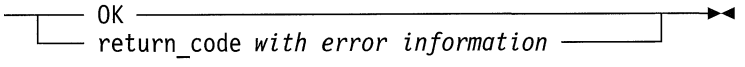
Notifies the partner transaction program that the local transaction program is requesting to enter Send state. The router places the conversation in Send state when the local transaction program subsequently receives SEND in the what_received parameter of a Receive verb from the partner transaction program.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*
: The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for the conversation.

Return Codes

The following indicates that the verb ran successfully:

OK
: The verb ran successfully.

The following return codes provide error information:

PARAMETER_CHECK
: The verb did not run because of a parameter error for the following reason:

  BAD_CONV_ID
  : The router does not recognize the specified conv_ID.

STATE_CHECK
The verb did not run because it is in the wrong state for the following reason:

R_T_S_BAD_STATE
The conversation is in the wrong state.

The remaining return codes (listed in Table 4-11) are described in "Verb Return Codes" on page 4-45.

Additional Information

The transaction program can issue this verb when the conversation is in either Send or Receive state. If the conversation is in Send state, the router flushes its send buffer, sending all buffered information and the send indicator to the partner transaction program. The conversation is now in Receive state. The router then waits for information to arrive from the partner transaction program after it receives the send indicator.

A local transaction program usually receives a request to send indicator when it is in Send state. It is reported in a SEND_DATA or SEND_ERROR verb.

A local program in Receive state, however, can receive a request to send indicator when:

- The partner transaction program issues a REQUEST_TO_SEND verb in Send state.
- The local transaction program has entered Receive state and the partner transaction program issued a REQUEST_TO_SEND verb before the local transaction program's state changed to Receive state.
- The partner transaction program issues a PREPARE_TO_RECEIVE or a RECEIVE_AND_POST verb and then issues a REQUEST_TO_SEND verb before the local transaction program can enter Send state. A REQUEST_TO_SEND verb is an expedited request that can arrive before the send indicator, which is ambiguous to the local transaction program. The partner transaction program can prevent this from happening by waiting to receive information before issuing a REQUEST_TO_SEND verb.

Verb Record Format

Table 4-11. REQUEST_TO_SEND Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'0E'	Request to send verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK (remote program replied CONFIRMED)
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'000000E1'	R_T_S_BAD_STATE
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID

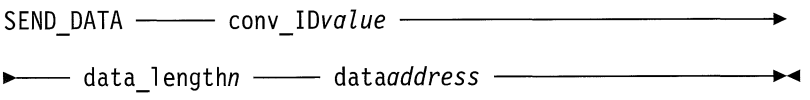
SEND_DATA

Purpose

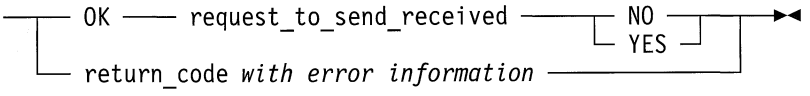
Sends data to the partner transaction program.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*
: The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for the conversation.

data_length*n*
: Specifies the number of bytes of data the local transaction program sends (0 to 65 535). This data length is not the same as the length of a logical record. The router uses this information only to determine the length of the data at the location specified by the data parameter.

data*address*
: The address of the buffer that contains the data to be sent.

  The data buffer must be entirely within the data segment, so the offset portion of the address plus the length of the data buffer must not exceed the segment size.

  The data may include multiple records or part of a record. Each logical record consists of a 2-byte length field (denoted as LL) followed by a data field; the length of the data field can range from 0 to 32 765 bytes.

  The 2-byte length field contains the 15-bit binary length of the record and a high-order bit; it must be the opposite of the byte-reversed order of a personal computer because it must conform to SNA format.

  The router ignores this high-order bit when performing basic conversations. The router's mapped conversation component uses this bit to support the mapped conversation verbs.

The length of the record includes the 2-byte length field; that is, a record is 2 bytes longer than the length of the data. For this reason, record length values of X'0000', X'0001', X'8000', and X'8001' are not valid.

Return Codes

The following indicates that the verb ran successfully:

OK The verb ran successfully.

The following parameter is returned to the transaction program by the router if the return code is OK:

request_to_send_received
: Indicates whether the router received a REQUEST_TO_SEND verb from the partner transaction program.

  NO The router did not receive an indicator from the partner transaction program.

  YES
  : The router received an indicator that the partner program sent a REQUEST_TO_SEND verb, which requests the local program both to enter Receive state and to place the partner program in Send state.

The following return codes provide error information:

PARAMETER_CHECK
: The verb did not run because of a parameter error for the following reason:

  BAD_CONV_ID
  : The router does not recognize the specified conv_ID.

STATE_CHECK
: The verb did not run because it is in the wrong state for the following reason:

  SEND_DATA_NOT_SEND_STATE
  : The conversation is not in Send state.

The remaining return codes (listed in Table 4-12 on page 4-42) are described in "Verb Return Codes" on page 4-45.

State Change

The state of the conversation does not change when the return code indicates OK.

Additional Information

- The data sent by the transaction program consists of logical records. The data_length parameter of the SEND_DATA verb determines the amount of data a program sends. This amount is independent of the length of the logical records. The data may consist of one or more complete records, the beginning of a record, the middle of a record, or the end of a record. The following combinations of complete and partial records are possible:
  - One or more complete records, followed by the beginning of a record.
  - The end of a record, followed by one or more complete records.

- The end of a record, followed by one or more complete records, followed by the beginning of a record.
- The end of a record, followed by the beginning of a record.

- A complete logical record contains a 2-byte LL field followed by data so that the total number of bytes equals what the program specifies in the LL field. (A value of 2 in the LL field specifies that the complete logical record contains only the 2-byte length field.)

  An incomplete logical record contains any amount of data less than a complete record. It can consist of only the first byte of the 2-byte LL field, the LL field plus all of the data field except the last byte, or any amount in between. A logical record remains incomplete until the program sends the last byte of the data field.

  If the value of the LL field is 2 (indicating a data field with a length of 0), the logical record is complete after the transaction program sends the second byte of the LL field.

- The transaction program must finish sending a logical record before it can issue any of the following verbs:

  CONFIRM
  DEALLOCATE with type specified as FLUSH
  DEALLOCATE with type specified as SYNC_LEVEL
  PREPARE_TO_RECEIVE
  RECEIVE_AND_WAIT

  A transaction program finishes sending a logical record when it sends a complete record or when it truncates an incomplete record.

- A program can truncate a logical record by issuing the SEND_ERROR verb. When a program issues a SEND_ERROR verb, the LU first flushes its buffer and sends whatever portion of a logical record it has accumulated. The LU then treats the first 2 bytes of data specified in the next SEND_DATA verb as the LL field for the next record.

  The program can also truncate a logical record by issuing a DEALLOCATE verb with type specified as ABEND_PROG.

- The router keeps the data from the program until it accumulates a sufficient amount for transmission from one or more SEND_DATA verbs or until the program issues a verb that forces the router to flush its send buffer. The amount of data that is sufficient for transmission varies from one session to another, depending on the characteristics of the session allocated for the conversation.

- When the request_to_send_received parameter indicates YES, the partner transaction program requests that the local transaction program enter Receive state and place the partner transaction program in Send state. A transaction program enters Receive state by issuing a PREPARE_TO_RECEIVE or a RECEIVE_AND_WAIT verb. The local transaction program enters Send state after it issues a RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verb and receives the SEND value in the what_received parameter from the partner transaction program.

Verb Record Format

Table 4-12. SEND_DATA Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'0F'	Send data verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK (remote program replied CONFIRMED)
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0006'	DEALLOC_ABEND_PROG
		X'0007'	DEALLOC_ABEND_SVC
		X'0008'	DEALLOC_ABEND_TIMER
		X'0009'	DEALLOC_NORMAL
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0013'	SVC_ERROR_PURGING
		X'F002'	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'00000004'	ALLOCATION_FAILURE_NO_RETRY
		X'00000005'	ALLOCATION_FAILURE_RETRY
		X'00000006'	INVALID_DATA_SEGMENT
		X'000000F2'	SEND_DATA_NOT_SEND_STATE
		X'080F6051'	SECURITY_NOT_VALID
		X'084B6031'	TRANS_PGM_NOT_AVAIL_RETRY
		X'084C0000'	TRANS_PGM_NOT_AVAIL_NO_RETRY
		X'10086021'	TP_NAME_NOT_RECOGNIZED
		X'10086031'	PIP_NOT_ALLOWED
		X'10086032'	PIP_NOT_SPECIFIED_CORRECTLY
		X'10086034'	CONVERSATION_TYPE_MISMATCH
		X'10086041'	SYNC_LEVEL_NOT_SUPPORTED
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID
38	1		Request to send received
		X'00'	No
		X'01'	Yes
39	1	X'00'	Not used by the router
40	2		Length of data to send
42	4		Address of data to send

SEND_ERROR

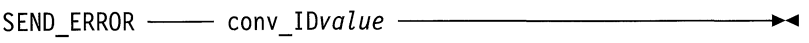
Purpose

Informs the partner transaction program that the local transaction program detected an error. If the local conversation is in Send state, the router flushes its send buffer.

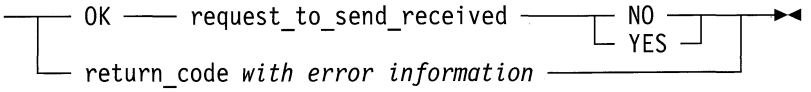
After the successful completion of this verb, the local transaction program is in Send state and the partner transaction program is in Receive state. The transaction program must take the appropriate actions to correct the problem.

Syntax

The following is a conceptual scheme of the verb, its supplied parameters, and the parameter options:



The following is a conceptual scheme of the returned parameters:



Parameters

conv_ID*value*
: The identifier of the conversation on which the verb is being issued. This value is returned in an ALLOCATE verb for the conversation.

Return Codes

The following indicates that the verb ran successfully:

OK
: The verb ran successfully.

The following parameter is returned to the transaction program by the router if the return code is OK:

request_to_send_received
: Indicates whether the router received a REQUEST_TO_SEND verb from the partner transaction program.

  NO
  : The router did not receive an indicator from the partner transaction program.

  YES
  : The router received an indicator that the partner program sent a REQUEST_TO_SEND verb, which requests the local program both to enter Receive state and to place the partner program in Send state.

The following return code provides error information:

PARAMETER_CHECK
: The verb did not run because of a parameter error for the following reason:

  BAD_CONV_ID
  : The router does not recognize the specified conv_ID.

The APPC_BUSY return code provides error information and can be returned independently of the state of the conversation. The DEALLOC_NORMAL return code can occur when the conversation is in Receive state. For detailed information about these return codes, see "Verb Return Codes" on page 4-45.

The remaining return codes (listed in Table 4-13 on page 4-44) are described in "Verb Return Codes" on page 4-45.

State Change

If the return code indicates OK, the conversation enters Send state when the transaction program issues the verb in Receive or Confirm state.

No state change occurs when the program issues the verb in Send or Send-Pending state.

Additional Information

- The router sends the error indicator immediately, before the verb is complete. The indicator is not held in the send buffer.
- The transaction program may use the SEND_ERROR verb for various application-level functions. For example, the program can issue this verb to truncate an incomplete logical record it is sending, to inform the remote program of an error it detected in data it received, or to reject a confirmation request.
- For detailed information about these return codes, see "Verb Return Codes" on page 4-45.

Verb Record Format

Table 4-13 (Page 1 of 2). SEND_ERROR Record Format

Offset	Length	Value	Description
0	12	X'00' (12 bytes)	Not used by the router
12	1	X'10'	Send error verb operation code
13	7	X'00' (7 bytes)	Not used by the router
20	2		**Primary Return Code**
		X'0000'	OK (remote program replied CONFIRMED)
		X'0001'	PARAMETER_CHECK
		X'0002'	STATE_CHECK
		X'0003'	ALLOCATION_ERROR
		X'0006'	DEALLOC_ABEND_PROG
		X'0007'	DEALLOC_ABEND_SVC
		X'0008'	DEALLOC_ABEND_TIMER
		X'0009'	DEALLOC_NORMAL
		X'000E'	PROG_ERROR_PURGING
		X'000F'	CONV_FAILURE_RETRY
		X'0010'	CONV_FAILURE_NO_RETRY
		X'0013'	SVC_ERROR_PURGING
		X'F002''	APPC_BUSY
22	4		**Secondary Return Codes**
		X'00000002'	BAD_CONV_ID
		X'00000004'	ALLOCATION_FAILURE_NO_RETRY
		X'00000005'	ALLOCATION_FAILURE_RETRY
		X'00000101'	SEND_ERROR_NOT_ALLOWED
		X'080F6051'	SECURITY_NOT_VALID
		X'084B6031'	TRANS_PGM_NOT_AVAIL_RETRY
		X'084C0000'	TRANS_PGM_NOT_AVAIL_NO_RETRY
		X'10086021'	TP_NAME_NOT_RECOGNIZED
		X'10086031'	PIP_NOT_ALLOWED
		X'10086032'	PIP_NOT_SPECIFIED_CORRECTLY
		X'10086034'	CONVERSATION_TYPE_MISMATCH
		X'10086041'	SYNC_LEVEL_NOT_SUPPORTED

Table 4-13 (Page 2 of 2). SEND_ERROR Record Format

Offset	Length	Value	Description
26	8	X'00' (8 bytes)	Not used by the router
34	4		Conversation ID
38	1		Request to send received
		X'00'	No
		X'01'	Yes

Verb Return Codes

This section describes the primary and secondary return codes, the cause, examples of the cause in many cases, and the action.

Overview of Return Codes

The router reports several types of information to a transaction program after a verb has been issued. The information always includes one return code to indicate the status of the verb. A return code of OK indicates that an issued verb ran successfully. Most of the others provide error information. Refer to the "Returned Parameters" section for each verb description in this chapter for details and other information reported.

This section describes the return codes by providing the hexadecimal return code value, an explanation of the condition causing the return code, and the corrective actions required if the return code indicates an error condition.

A router verb returns a 2-byte primary return code and a 4-byte secondary return code. The primary return code denotes the type of error. The secondary return code specifies the cause. Return codes are stored with the most significant byte first rather than in Intel format.

If the primary return code is not 0, then you must still check the request_to_send_received field in the verb record. This field is set no matter what the primary return code is.

If the router cannot recognize a verb (that is, the verb operation code is not valid), it returns the INVALID_VERB return code. This return code is not listed with each verb description.

How to Handle Router Return Codes

Router return codes have three uses:

1. To control the flow and operation of a transaction program
2. To aid the writers and testers of the programs during development and testing
3. To help the user of the programs

These three uses generally do not overlap. When writing transaction programs:

1. You should ensure that the PC program knows the state of its conversations at all times, in addition to knowing what input is arriving from the outside world (such as mouse input or keystrokes). The program cannot repair communications cable if it breaks. It cannot correct errors by its partner program. It cannot correct errors made by its programmer.

2. You should exhaustively test the pairs of programs you write. You should then rewrite those programs that supply incorrect parameters or end up in incorrect conversation states. No user should encounter these types of errors in the production environment.
3. You should assume that the users of your programs know very little about communications. They have two tasks: to get the program configured and running correctly the first time, and to get it running again if it stops. They cannot correct or debug the program and they cannot necessarily reconnect cables without guidance. They are the first users of the return code information once it is in the production environment. You must make that information useful to them and to the person who must correct the real problem.

For each verb, establish four categories for handling. Treat the return codes differently by group.

Group 1: Normal program operation

Your program should proceed to the next line of operation. This should be the condition you expect your program to see much of the time.

Examples of these conditions are:

```
0000 OK
0009 DEALLOC_NORMAL
```

Group 2: Recoverable by the program

Your program should handle the situation since it is reasonable to expect problems to occur some percentage of the time. The program should take the appropriate recovery action and continue.

The following list of examples is long, but many of these apply only to specific verbs. Examples of these conditions are:

```
0005 DEALLOC_ABEND
0006 DEALLOC_ABEND_PROG
0007 DEALLOC_ABEND_SVC
0008 DEALLOC_ABEND_TIMER
000C PROG_ERROR_NO_TRUNC
000D PROG_ERROR_TRUNC
000E PROG_ERROR_PURGING
0011 SVC_ERROR_NO_TRUNC
0012 SVC_ERROR_TRUNC
0013 SVC_ERROR_PURGING
0014 UNSUCCESSFUL
F002 APPC_BUSY
```

Note: Although the APPC_BUSY return code is easy to recover from, you may want to avoid designing your transactions in such a way that it is an expected condition.

Group 3: Human intervention might help

Since your program cannot correct the PC Support configuration, the cables, or itself, human intervention may be needed in order for the program to run successfully. The secondary return codes supply additional information about the cause of the error.

These are the only conditions that the users of your program should expect to see. You must translate each of these into useful directions that an unskilled user can follow in order to correct the situation.

Examples of these conditions are:

```
0003 ALLOCATION_ERROR
000F CONV_FAILURE_RETRY
0010 CONV_FAILURE_NO_RETRY
```

Group 4: Unexpected errors

Users of your program should never see these errors. Recovery requires corrective changes to your local or remote program or both. The secondary return codes supply additional information about the cause of the error.

These errors will probably occur frequently during program development and testing, but should not be seen in a normal production environment. If one of these errors does occur in a production environment, your program should produce a detailed log entry for the problem. The log entry (a message log, an error log, or both) should contain enough information so that the user can tell if the problem was caused by a bug (a suspected defect in a current, unaltered release of PC Support) or a user error. Your user should be told that the program has a defect and there should be enough information recorded so that you can perform problem analysis (for example, module, function, location, and error).

After creating a log entry, the program should end. The conversation in error should end by attempting to issue DEALLOCATE with type(ABEND). This occurs automatically if the program issues a TP_ENDED verb.

Examples of these conditions are:

```
0001 PARAMETER_CHECK
0002 STATE_CHECK
```

Primary Return Codes

Following are the primary return codes, in alphabetical order:

ALLOCATION_ERROR	0003
APPC_BUSY	F002
CONV_FAILURE_NO_RETRY	0010
CONV_FAILURE_RETRY	000F
DEALLOC_ABEND	0005
DEALLOC_ABEND_PROG	0006
DEALLOC_ABEND_SVC	0007
DEALLOC_ABEND_TIMER	0008
DEALLOC_NORMAL	0009
OK	0000
PARAMETER_CHECK	0001
PROG_ERROR_NO_TRUNC	000C
PROG_ERROR_PURGING	000E
PROG_ERROR_TRUNC	000D
STATE_CHECK	0002
SVC_ERROR_NO_TRUNC	0011
SVC_ERROR_PURGING	0013
SVC_ERROR_TRUNC	0012
UNSUCCESSFUL	0014

Numeric Listing of Primary Return Codes

0000 OK

Explanation: The verb ran successfully.

Programmer Response: No action is required.

0001 PARAMETER_CHECK

Explanation: Your local PC program issued a router verb, specifying a parameter with an incorrect parameter value. The specific error conditions vary with each verb. The secondary return code indicates the parameter that is in error.

This error is not an indication that there is a mismatch between your PC Support configuration and the configuration at the remote location. If there was an error between the remote configuration and your local configuration, you would see an ALLOCATION ERROR return code in your program and BIND failures in the local error log.

Programmer Response: Correct the parameter, taking the action described in the secondary return code.

A PARAMETER CHECK means that there is a major defect in your program. When your program receives this return code, the recommended recovery technique is to end your program. If a (presumably debugged) program has written incorrect data into one of its parameters, the entire program is suspect.

This return code will continue to occur until the local program is changed.

0002 STATE_CHECK

Explanation: The router returns STATE_CHECK for conversation verbs.

Your local transaction program issued a verb on a conversation that was in a conversation state in which the verb is not allowed. This indicates that one or both of the transaction programs has not been properly tested and debugged. Your transaction program has not properly handled the state information returned by the router.

Your program should be looking at the router return codes and at the what_received values returned by the RECEIVE verbs. This information can be used with the state tables in Appendix B to determine the current state.

Programmer Response: Take the action described in the secondary return code.

For conversation verbs, correct your local program to be aware of its current state by using the information returned by the router and then issuing the verb only in a valid state.

0003 ALLOCATION_ERROR

Explanation: The router could not allocate a conversation with the remote transaction program.

The conversation has been ended; it is in Reset state. A conversation ID (conv_id) was returned as a parameter by the verb that received this return code. That conv_id is not valid.

Programmer Response: The router has deallocated and cleaned up the conversation. If your program issues any more verbs for this conversation, the router replies with the PARAMETER_CHECK return code. Take the action described in the secondary return code.

User Response: The most probable causes are mistakes in the configuration at either the local or the remote machine (or both), or improper cabling and hardware setup. Look for message CPF 1269 in the AS/400 message queue QSYSOPR to find more information about this problem.

0005 DEALLOC_ABEND

Explanation: The source of the error notification is either the remote transaction program or the remote system. The remote transaction program can cause this error notification by issuing a DEALLOCATE verb specifying the type(ABEND) parameter. Alternatively, the remote system can issue a DEALLOCATE because of a remote transaction program ABEND condition.

The conversation has been ended; it is in Reset state. A conversation ID (conv_id) was supplied as a parameter by the verb that received this return code. That conv_id is no longer valid.

Programmer Response: The router has deallocated and cleaned up the conversation. If your program issues any more verbs for this conversation, the router will reply with the PARAMETER_CHECK return code. Check the transaction program for errors.

0006 DEALLOC_ABEND_PROG

Explanation: The remote transaction program or the remote system causes this error notification by issuing a DEALLOCATE verb specifying type(ABEND_PROG). If the conversation for the remote transaction program is in Receive state when DEALLOCATE occurs, information sent by the local transaction program but not yet received by the remote transaction program is purged (not sent to the local transaction program).

The router reports this return code on a verb the program issues in Send or Receive state when the remote transaction program or remote system deallocates the conversation.

The conversation has been ended; it is in Reset state. A conversation ID (conv_id) was supplied as a parameter by the verb that received this return code. That conv_id is no longer valid.

Programmer Response: The router has deallocated and cleaned up the conversation. If your program issues any more verbs for this conversation, the router will reply with the PARAMETER_CHECK return code. Check the transaction program for errors.

0007 DEALLOC_ABEND_SVC

Explanation: The remote transaction program or the remote system causes this error notification by issuing a DEALLOCATE verb specifying type(ABEND_SVC). If the conversation for the remote transaction program is in Receive state when DEALLOCATE occurs, information sent by the local transaction program but not yet received by the remote transaction program is purged (not sent to the local transaction program).

The router reports this return code on a verb the program issues in Send or Receive state when the remote transaction program or remote system deallocates the conversation.

The conversation has been ended; it is in Reset state. A conversation ID (conv_id) was supplied as a parameter by the verb that received this return code. That conv_id is no longer valid.

Programmer Response: The router has deallocated and cleaned up the conversation. If your program issues any more verbs for this conversation, the router will reply with the PARAMETER_CHECK return code. Check the transaction program for errors.

0008 DEALLOC_ABEND_TIMER

Explanation: The remote transaction program causes this error notification by issuing a DEALLOCATE verb specifying type(ABEND_TIMER). If the conversation for the remote transaction program is in Receive state when DEALLOCATE occurs, information sent by the local transaction program but not yet received by the remote transaction program is purged (not sent to the local transaction program).

The router reports this return code on a verb the program issues in Send or Receive state when the remote transaction program or remote system deallocates the conversation.

The conversation has been ended; it is in Reset state. A conversation ID (conv_id) was supplied as a parameter by the verb that received this return code. That conv_id is no longer valid.

Programmer Response: The router has deallocated and cleaned up the conversation. If your program issues any more verbs for this conversation, the router will reply with the PARAMETER_CHECK return code. Check the transaction program for errors.

0009 DEALLOC_NORMAL

Explanation: The remote transaction program issued a DEALLOCATE verb, specifying type(FLUSH) or type(SYNC_LEVEL) with the synchronization level of the conversation specified as NONE. APPC reports this return code to the local transaction program on a subsequent verb it issues in Receive state.

The conversation has been ended; it is in Reset state. A conversation ID (conv_id) was supplied as a parameter by the verb that received this return code. That conv_id is no longer valid.

Programmer Response: No action is required. This return code is not an error indication. The router has deallocated and cleaned up the conversation. If your program issues any more verbs for this conversation, the router will reply with the PARAMETER_CHECK return code.

000C PROG_ERROR_NO_TRUNC

Explanation: The remote transaction program issued a SEND_ERROR verb, specifying type(PROG) and the conversation for the remote transaction program was in Send state; the SEND_ERROR did not truncate a logical record. No truncation occurs when a program issues SEND_ERROR before sending any logical records or after sending a complete logical record.

The router reports this return code on a RECEIVE verb and the conversation remains in Receive state.

Programmer Response: Correct the reported error.

000D PROG_ERROR_TRUNC

Explanation: The remote transaction program issued a SEND_ERROR verb, specifying the type(PROG) parameter. The conversation for the remote transaction program was in Send state, and the SEND_ERROR truncated a logical record. Truncation occurs when a program begins sending a logical record and then issues SEND_ERROR before sending the complete logical record. The router reports this return code on a RECEIVE verb the local transaction program issues after receiving the truncated logical record.

Programmer Response: Correct the reported error and resend the truncated record.

000E PROG_ERROR_PURGING

Explanation: The remote transaction program issued a SEND_ERROR verb, specifying type(PROG) while in Receive or Confirm state. If the remote transaction program issues a SEND_ERROR verb when it is in Receive state, information sent to, but not yet received by, the remote transaction program is deleted.

The router normally reports this return code on a verb the local transaction program issues after sending information to the remote transaction program. However, the router can also report this return code on a verb the program issues before sending any information, depending on the verb and when the program issues it. The conversation is in Receive state.

Programmer Response: The program should correct the reported error and prepare to resend data that may have been lost.

000F CONV_FAILURE_RETRY

Explanation: A temporary failure prematurely ended the conversation.

Examples:

- The router was notified of a session outage occurring in the network.
- The router or remote system deactivated the session in the middle of a conversation.

The conversation has been ended; it is in Reset state. A conversation ID (conv_id) was supplied as a parameter by the verb that received this return code. That conv_id is no longer valid.

Programmer Response: The router has deallocated and cleaned up the conversation. If your program issues any more verbs for this conversation, the router will reply with the PARAMETER_CHECK return code. You may want to try the operation again.

User Response: The local program should attempt to allocate the conversation again. It is possible for some permanent failures to be initially reported as temporary, but the allocation of subsequent conversations would also fail. Look at the job log on the AS/400 system to find more information about this problem.

0010 CONV_FAILURE_NO_RETRY

Explanation: A permanent failure prematurely ended the conversation. The condition is not temporary; operator intervention is required to correct the problem. This return code probably indicates that the link between the local and partner systems has been broken for some reason.

Examples:

- The local and remote systems have been disconnected from each other. The link or session has been abruptly ended.
- The remote system may have violated internal SNA protocols.

The conversation has been ended; it is in Reset state. A conversation ID (conv_id) was supplied as a parameter by the verb that received this return code. That conv_id is no longer valid.

Programmer Response: The router has deallocated and cleaned up the conversation. If your program issues any more verbs for this conversation, the router will reply with the PARAMETER_CHECK return code.

User Response: Check the communications cables at the local machine and at the remote machine, if possible. Look at the job log on the AS/400 system to find more information about this problem.

0011 SVC_ERROR_NO_TRUNC

Explanation: The remote transaction program or remote LU issued a SEND_ERROR verb, specifying the type(SVC) parameter. The conversation for the remote transaction program was in Send state, and the SEND_ERROR did not truncate a logical record. No truncation occurs when a program issues the verb before sending any logical records or after sending a complete logical record. The router reports this return code on a RECEIVE verb and the conversation remains in Receive state.

Programmer Response: Correct the reported error.

0012 SVC_ERROR_TRUNC

Explanation: The remote transaction program or remote LU issued a SEND_ERROR verb, specifying type(SVC) while the conversation for the remote transaction program was in Send state and the SEND_ERROR truncated a logical record. Truncation occurs if a transaction program begins sending a logical record and the transaction program or remote LU issues a SEND_ERROR verb before a complete logical record has been sent. The router reports this return code on a RECEIVE verb that the local transaction program issues after receiving the truncated logical record.

Programmer Response: Correct the reported error and resend the truncated record.

0013 SVC_ERROR_PURGING

Explanation: The remote transaction program or remote system issued a SEND_ERROR verb, specifying type(SVC) in Receive or Confirm state. If the remote transaction program issues a SEND_ERROR verb when it is in Receive state, information sent to, but not yet received by, the remote transaction program is deleted.

The router normally reports this return code on a verb the local transaction program issues after sending information to the remote transaction program. However, the router can also report this return code on a verb that the program issues before sending any information, depending on the verb and when the program issues it.

Programmer Response: The program should correct the error and prepare to resend the data that may have been deleted.

0014 UNSUCCESSFUL

Explanation: This return code has different meanings, depending on whether the verb issued was ALLOCATE or RECEIVE_IMMEDIATE.

- For an ALLOCATE verb, the router could not allocate a session when requested to do so.
- For a RECEIVE_IMMEDIATE verb, there is nothing to receive.

Programmer Response: The requested verb did not run. No explicit action is required. Try the verb again as appropriate.

F002 APPC_BUSY

Explanation: The router did not run the verb for a transaction program because the router was processing another verb for that transaction program on a different thread. The router can process only one verb from a given transaction program at a time. This return code can occur if more than one thread of a transaction program issues verbs using the same tp_id.

Programmer Response: The requested verb did not run. No explicit action is required. The verb can be issued again by your program.

Secondary Return Codes

Following are the secondary return codes, in alphabetical order.

ALLOCATION_FAILURE_NO_RETRY	00000004
ALLOCATION_FAILURE_RETRY	00000005
BAD_CONV_ID	00000002
BAD_CONV_TYPE	00000011
BAD_FILL	000000B4
BAD_SYNC_LEVEL	00000012
CONFIRM_ON_SYNC_LEVEL_NONE	00000031
CONFIRMED_BAD_STATE	00000041
CONVERSATION_TYPE_MISMATCH	10086034
DEALLOC_BAD_TYPE	00000051
INVALID_DATA_SEGMENT	00000006

INVALID_VERB	00000000
PIP_LEN_INCORRECT	00000016
PIP_NOT_ALLOWED	10086031
PIP_NOT_SPECIFIED_CORRECTLY	10086032
RCV_IMMD_BAD_STATE	000000C1
R_T_S_BAD_STATE	000000E1
SECURITY_NOT_VALID	080F6051
SEND_DATA_NOT_SEND_STATE	000000F2
SEND_ERROR_BAD_TYPE	00000103
SEND_ERROR_NOT_ALLOWED	00000101
SYNC_LEVEL_NOT_SUPPORTED	10086041
TP_NAME_LENGTH_ERROR	00000010
TP_NAME_NOT_RECOGNIZED	10086021
TRANS_PGM_NOT_AVAIL_NO_RETRY	084C0000
TRANS_PGM_NOT_AVAIL_RETRY	084B6031
UNKNOWN_PARTNER_SYSTEM	00000018

Numeric Listing of Secondary Return Codes

00000000 INVALID VERB

Explanation: The verb passed was not correct.

Programmer Response: Correct the verb code and issue the request again.

00000002 BAD_CONV_ID

Explanation: The router does not recognize the specified conv_id parameter. Some likely causes are:

- Your program never issued an ALLOCATE verb.
- Your program issued an ALLOCATE verb, but the router replied with a return code other than OK on that verb.
- An earlier conversation verb issued for this conversation received a return code other than OK, which ended the conversation.
- The conv_id returned by the ALLOCATE verb was not copied correctly into the verb control block for the current verb.
- Your program issued a successful DEALLOCATE verb, which makes the conv_id no longer valid.

Programmer Response: If the conversation has not ended, issue the verb again with the correct conv_id.

00000004 ALLOCATION_FAILURE_NO_RETRY

Explanation: The router replies with this return code when it has determined that it is impossible to allocate a conversation without some human intervention. For example, the router can tell when there is no adapter in the local machine, making a conversation impossible.

If there is any possibility that a conversation might be established, the router responds with the ALLOCATION_FAILURE_RETRY return code (00000005). For example, the router cannot look in the remote machine and see if there is an adapter there.

This distinction is justified, because most of the time--in normal production operation--the problem is a transient link failure, which is recoverable. Retry will generally be successful. This situation should be contrasted with a test environment, where the conversation is being established for the first time. In this environment, either return code is likely.

Table 4-14 on page 4-54 describes which router return code a transaction program will receive when an allocation failure occurs because of a problem at the local or remote system.

Entry	Description
No Retry	Primary return code 0003, secondary return code 00000004: ALLOCATION_FAILURE_NO_RETRY The transaction program should not try to allocate the conversation again until the condition is corrected. The implication of this return code is that something is wrong at the local machine, but this is not the case.
Retry	Primary return code 0003, secondary return code 00000005: ALLOCATION_FAILURE_RETRY There was probably a difficulty in activating a link or a session. The transaction program may try again to allocate the conversation. To avoid congesting the network with attempted allocation requests, the transaction program should pause or wait for a keystroke before trying the allocation again and should limit the number of retries. The implication of this return code is that something is wrong at the remote machine or with the connection between the two machines, but this is not the case. **Note:** Even though ALLOCATION_FAILURE_RETRY is the return code, the allocation request might never succeed, especially if the partner is never activated or has been configured wrong.
—	This situation either cannot occur on this machine or does not cause an allocation failure.

When looking at the Local System column in Table 4-14, assume the remote system is installed and configured correctly; the return code thus indicates a hardware or software problem at the local system. Conversely, when looking at the Remote System column, assume the local system is installed and configured correctly. (However, it is possible that both systems could be incorrectly installed or configured.)

Message numbers indicate example messages logged on the system making the allocation request.

Programmer Response: Correct the problem as indicated in the following table.

Table 4-14 (Page 1 of 2). Causes of 0003-00000004 (NO_RETRY) and 0003-00000005 (RETRY)

Problem Description	Problem Is on Local System	Problem Is on Remote System
Operational problems		
System not powered on.	—	Retry
No communications adapter.	No Retry	Retry
No communications cable or the wrong cable. Example (using SDLC): Although you ordered Feature Code 2067 cables with your 5853 modems, you could not get them and substituted some clones. The clone cables have a full set of 23 pins, but they do not have enough wire. The data clocking lines (15 and 17) are missing.	No Retry	Retry
Communications adapter is not installed properly or is not functioning properly. Example: A token-ring network adapter has detected an error condition and is 20 must be corrected. Example: A token-ring network multi-station access unit (MAU) has been cabled incorrectly.	No Retry	Retry

Table 4-14 (Page 2 of 2). Causes of 0003-00000004 (NO_RETRY) and 0003-00000005 (RETRY)

Problem Description	Problem Is on Local System	Problem Is on Remote System
Dual in-line package (DIP) switches on the adapter are set incorrectly. Refer to the technical reference manual for the adapter. Interrupt addresses for different cards may conflict within one machine, causing one or more adapters to appear unusable.	Retry	Retry
Unable to reach destination address. Example: The system containing the remote system is on a different token-ring than the local machine. The two rings are connected by a bridge. The bridge is powered off or has failed.	Retry	Retry
Configuration problems		
Duplicate LAN Adapter address configured.	No Retry	No Retry
Wrong Network name.	No Retry	No Retry
DLC configuration does not match the adapter or modem or their switch settings. Example: The Non-return to zero inverted (NRZI) setting for the modem when configuring SDLC does not match the setting expected by the modem. Example: The Link station role setting (that is, primary or secondary) does not match the setting expected by the partner.	Retry	Retry

00000005 ALLOCATION_FAILURE_RETRY

Explanation: See the preceding discussion for 00000004, ALLOCATION_FAILURE_NO_RETRY.

Programmer Response: Correct the problem as indicated in Table 4-14 on page 4-54.

00000006 INVALID_DATA_SEGMENT

Explanation: The segment that contains the data buffer is too small for the specified data length. The data length may be too large, or the address of the data buffer may be wrong.

Programmer Response: Determine the size and starting address of the data segment. Make sure that the data length and data address parameters specified in the verb control block correspond to the actual size and location of the desired segment.

00000010 TP_NAME_LENGTH_ERROR

Explanation: The value passed as the TP name length on an allocate request must be from 1 to 64.

Programmer Response: Ensure that the length of the TP name is from 1 to 64.

00000011 BAD_CONV_TYPE

Explanation: The router does not recognize the specified conversation_type parameter. The specified value must be either BASIC_CONVERSATION (X'00') or MAPPED_CONVERSATION (X'01').

Programmer Response: Look at the verb control block being issued for this verb. Correct your local program so that it specifies a valid value for this parameter.

This return code will continue to occur until the local program is changed.

00000012 BAD_SYNC_LEVEL

Explanation: The router does not recognize the specified sync_level parameter. The specified value must be either NONE (X'00') or CONFIRM (X'01').

Programmer Response: Look at the verb control block being issued for this verb. Correct your local program so that it specifies a valid value for this parameter.

This return code will continue to occur until the local program is changed.

00000016 PIP_LEN_INCORRECT

Explanation: The pip_data_length is longer than 168 bytes.

Programmer Response: Look at the verb control block being issued for this verb. Correct your local program so that it specifies a value less than or equal to 168.

This return code will continue to occur until the local program is changed.

00000018 UNKNOWN_PARTNER_SYSTEM

Explanation: The router does not recognize the specified partner_system_name.

Programmer Response: Look at the verb control block being issued for this verb. Correct your local program so that it specifies a partner_system_name that has been configured.

The most probable cause for this return code is a partner system name that has been spelled wrong in your program or in the configuration. The names used in your program must match the names specified in your PC Support configuration file (CONFIG.PCS).

If the names are less than 8 characters in length, your program must pad them on the right with blanks. If you think that these names are spelled correctly, padded correctly, and that they use the proper character set , it is possible that your program has not moved them into your ALLOCATE verb control block correctly or that they were accidentally overwritten after they were moved.

This return code will continue to occur until either the local program or the local configuration is changed.

Operator Response: Make sure that the proper partner remote system name is configured. The name must match the parameter name specified on the ALLOCATE verbs in this program.

00000031 CONFIRM_ON_SYNC_LEVEL_NONE

Explanation: The router does not permit a transaction program to issue an CONFIRM verb if the synchronization level of this conversation was allocated as NONE.

Programmer Response: Either change the ALLOCATE verb to specify a sync_level other than NONE or prevent the program from issuing the CONFIRM verb.

This return code will continue to occur until the local program is changed.

00000041 CONFIRMED_BAD_STATE

Explanation: The conversation is not in Confirm state.

Programmer Response: Your transaction program has not properly handled the state information returned by the router.

Your program should be looking at the router return codes and at the what_received values returned by the RECEIVE verbs. This information can be used with the state tables in Appendix B to find the current state.

Using the information returned by the router, correct your local program to be aware of its current state.

00000051 DEALLOC_BAD_TYPE

Explanation: The router does not recognize the specified type parameter.

Programmer Response: Look at the verb control block being issued for this verb. Correct your local program so that it specifies a valid value for this parameter.

This return code will continue to occur until the local program is changed.

000000B4 BAD_FILL

Explanation: The transaction program specified an incorrect value for the fill parameter. The specified value must be either BUFFER (X'00') or LL (X'01').

Programmer Response: Look at the verb control block being issued for this verb. Correct your local program so that it specifies a valid value for this parameter.

This return code will continue to occur until the local program is changed.

000000C1 RCV_IMMD_BAD_STATE

Explanation: The conversation is not in Receive state.

Programmer Response: Your transaction program has not properly handled the state information returned by the router.

Your program should be looking at the router return codes and at the what_received values returned by the RECEIVE verbs. This information can be used with the state tables in Appendix B to find the current state.

Using the information returned by the router, correct your local program to be aware of its current state.

000000E1 R_T_S_BAD_STATE

Explanation: The conversation is in the wrong state.

Programmer Response: Your transaction program has not properly handled the state information returned by the router.

Your program should be looking at the router return codes and at the what_received values returned by the RECEIVE verbs. This information can be used with the state tables in Appendix B to find the current state.

Using the information returned by the router, correct your local program to be aware of its current state.

000000F2 SEND_DATA_NOT_SEND_STATE

Explanation: The conversation is not in Send state.

Programmer Response: Your transaction program has not properly handled the state information returned by the router.

Your program should be looking at the router return codes and at the what_received values returned by the RECEIVE verbs. This information can be used with the state tables in Appendix B to find the current state.

Using the information returned by the router, correct your local program to be aware of its current state.

00000101 SEND_ERROR_NOT_ALLOWED

Explanation: The conversation is in Reset state.

Programmer Response: Your transaction program has not properly handled the state information returned by the router.

Your program should be looking at the router return codes and at the what_received values returned by the RECEIVE verbs. This information can be used with the state tables in Appendix B to find the current state.

Using the information returned by the router, correct your local program to be aware of its current state.

080F6051 SECURITY_NOT_VALID

Explanation: The remote system rejected the user ID or password received on an incoming allocation request. The router reports this return code on a verb issued after the ALLOCATE verb, except for GET_ATTRIBUTES or FLUSH.

Operator Response: If the problem is with the remote configuration, make sure that the user ID and password combination has been correctly defined there.

Also, make sure that if the user ID and password are required (see the remote location's transaction program definition), they will be accepted by the remote system (see the remote location's definition of its remote system).

084B6031 TRANS_PGM_NOT_AVAIL_RETRY

Explanation: The remote system rejected the incoming allocation request because it could not start the specified program immediately. The router reports this return code on a conversation verb issued after the ALLOCATE verb.

Programmer Response: Retry the allocation request. However, to avoid congesting the network with attempted allocation requests, your local program should pause or wait for a keystroke before retrying the conversation.

084C0000 TRANS_PGM_NOT_AVAIL_NO_RETRY

Explanation: The remote system rejected the incoming allocation request because it could not start the specified program. The router reports this return code on a verb issued after the ALLOCATE verb.

Examples:

- The intended program does not exist at the remote location.
- The operating system or communications subsystem at the remote location has used all of its available resources and is unable to start the remotely attachable program.

Programmer Response: Do not retry the allocation request.

Operator Response: Make sure that the intended program has been defined correctly at the remote location and that it has been compiled and linked correctly for its operating system.

10086021 TP_NAME_NOT_RECOGNIZED

Explanation: The remote system rejected the incoming allocation request because the local transaction program specified a tp_name that the remote system does not recognize. This return code can also indicate that the remote system recognized the transaction program name but could not initiate the transaction program using the designated remote system or mode name.

Programmer Response: Check the validity of the transaction program name and the designated remote system and mode names.

This return code will continue to occur until either the local program or the remote configuration is changed.

Operator Response: On the remote system, check the list of transaction program names to be recognized. Make sure that they match the values supplied for the tp_name values on the ALLOCATE verbs in the local system.

If the transaction program is correctly defined on the remote system, make sure that the remote transaction program is properly authorized. Some implementations of the router check an incoming user_id and password for each specific transaction program and reject the allocation request with this return code if your program is not authorized to start that transaction program.

10086031 PIP_NOT_ALLOWED

Explanation: The remote system rejected the allocation request because the local transaction program specified program initialization parameters (PIP data) and either the remote system does not support PIP data or the remote transaction program has no PIP variables defined. The router reports this return code on a conversation verb issued after the ALLOCATE verb.

Programmer Response: Do not use PIP data when communicating with this remote transaction program.

This return code will continue to occur until the local program is changed.

10086032 PIP_NOT_SPECIFIED_CORRECTLY

Explanation: The remote system rejected the incoming allocation request because the remote transaction program has one or more program initialization parameter (PIP) variables defined, and either the local transaction program has specified that no PIP variables are to be used or the number of PIP variables defined by the local transaction program is different from the number specified by the remote transaction program. The router reports this return code on a conversation verb issued after the ALLOCATE verb.

Programmer Response: Specify that PIP data is to be used in the ALLOCATE verb, and make sure that the number of PIP variables agrees with the number required by the remote transaction program.

This return code will continue to occur until the local program is changed.

10086034 CONVERSATION_TYPE_MISMATCH

Explanation: The remote system rejected the incoming allocation request because it or the remote transaction program does not support the specified conversation type. The router reports this return code on a conversation verb issued after the ALLOCATE verb.

Programmer Response: Change the transaction program so it uses the proper conversation type (basic or mapped).

This return code will continue to occur until either the local program or the remote configuration is changed.

Operator Response: Alternatively, have the remotely attachable transaction program profile at the remote location changed to reflect the conversation type used by the program.

10086041 SYNC_LEVEL_NOT_SUPPORTED

Explanation: The remote system rejected the incoming allocation request because the local transaction program specified an unacceptable sync_level parameter. The router reports this return code on a conversation verb issued after the ALLOCATE verb. For example, the local transaction program issued an ALLOCATE verb with sync_level(CONFIRM), but at the remote system it was configured as sync_level(NONE).

Programmer Response: Change the sync_level parameter on the ALLOCATE verb.

This return code will continue to occur until either the local program or the remote configuration is changed.

Operator Response: Alternatively, have the transaction program profile at the remote location changed to reflect the sync_level used by the local transaction program.

Program Examples

The example in this section shows an application that uses the router's API to communicate with another program running on the AS/400 system.

The AS/400 program is written in the RPG III programming language. The PC program is presented in three different programming languages: C, Modula-2, and Macro Assembler. The example first describes how the application works and how the APPC conversation flows between the personal computer and the AS/400 system. Then the example describes the PC program using flow charts and code segments. Finally, the AS/400 program is described.

All of the actual code for the example can be found in the PC Support tools folder (QIWSTOOL) on the AS/400 system. However, to help you, some of the code is shown in examples in this book. See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

Some knowledge of the following topics is necessary to understand the examples:

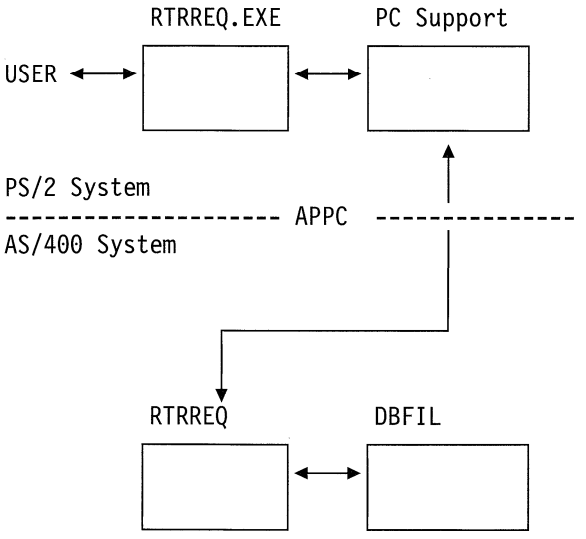
- The IBM Personal Computer or PS/2* family of personal computers
- The DOS operating system
- The PC Support/400 licensed program
- The IBM C/2 programming language
- The IBM AS/400 family of computers
- The OS/400* operating system
- The AS/400 RPG/400* programming language
- The AS/400 control language (CL)
- AS/400 data description specifications (DDS)

The sample programs in this example are not copyright protected and may be used in part or in their entirety without permission from the author. These programs are supplied "as is" without warranty of any kind. IBM assumes no responsibility for any errors that may be present in the programs.

Example Overview

This application example demonstrates a PS/2 program module doing an inquiry to an AS/400 database using the services provided by the PC Support router.

The following diagram shows the application flow for the example programs. The user interfaces to the RTRREQ program on the PS/2 system. The PS/2 RTRREQ program uses PC Support to communicate with the AS/400 program, RTRREQ. The AS/400 program accesses the AS/400 database, DBFIL, and returns requested information to the PS/2 program.



User Interface

This section shows the application interface to the user.

1. The user is prompted for a part number.

```
Enter Part Number =
```

2. The user enters a five-digit part number.

```
Enter Part Number = 00001
```

3. The part number is retrieved from the AS/400 database and an output display is formatted and returned to the user. The user is then prompted for the next part number. If the part number cannot be retrieved from the database, an error message is displayed. The user is then prompted for the next part number. If ENDPGM is entered in the part number field, the program ends.

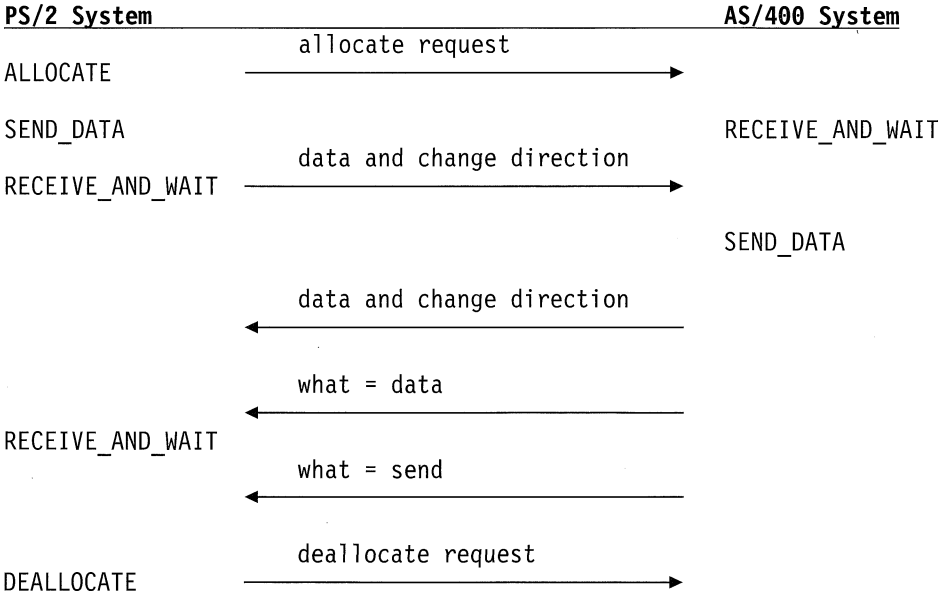
```
Part Number = 00001
Description = MACHINE TOOLS
Quantity    = 00005

*** Enter New Part Number or ENDPGM to stop ***

Part Number =
```

Note: The appearance of the user interface may vary depending on the programming language used for the sample program.

Conversation Flow

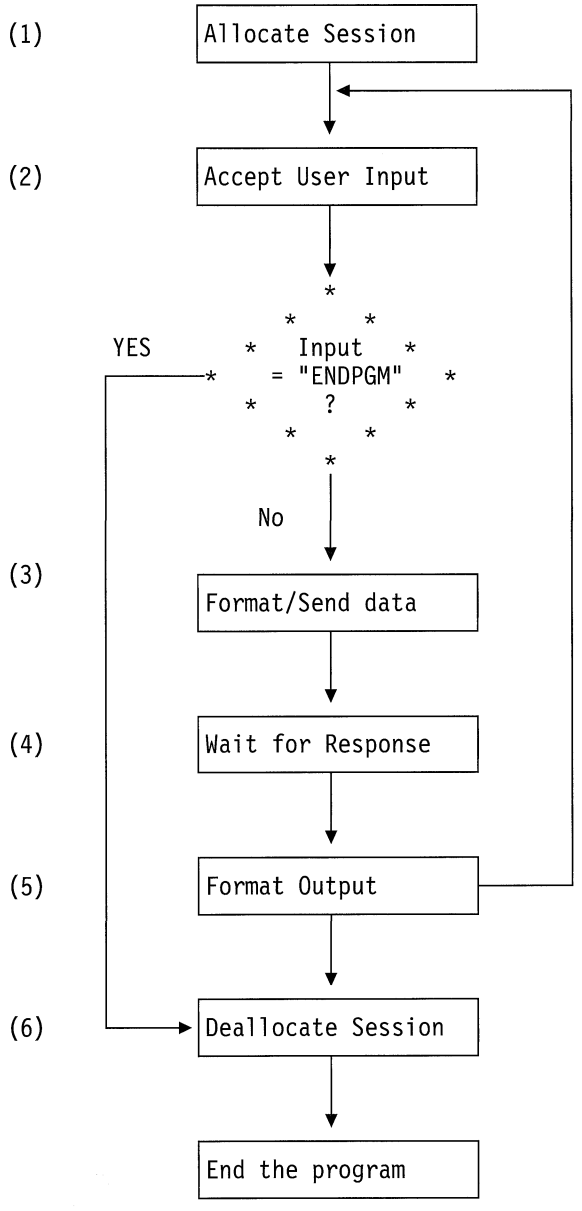
The following chart depicts the APPC conversation flow. It shows the interaction that occurs between the PS/2 system and the AS/400 system as a result of issuing the APPC verbs. The PS/2 system initiates the conversation with the ALLOCATE verb and ends the conversation with the DEALLOCATE verb. The entire conversation flow is described along with the data and control information passed between the systems.

The RPG III program runs with a mapped conversation. This means that the PC program must allocate the conversation as a mapped conversation and use the general data stream identifier (GDSID) of X'12FF' for all records sent to the AS/400 application.



PS/2 Processing Flow

The RTRREQ module provides PS/2 support. This section shows the flow of the program.



C Program Example

The APIRTREX package is available in the PC Support tools folder (QIWSTOOL). See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

A subroutine library is used to make the calls to the router. The application program makes calls to subroutines that have been previously defined. This library can be used by any program that needs to use the router.

The subroutines are in two parts: a header file and the implementation file. The header file defines the interfaces to the routine through function prototypes that describe the types of parameters expected by the different subroutines. The header file also has an enumeration that defines all the return codes. The subroutines map the 6-byte router return codes into an easier to deal with 1-byte return code. The mapping of the return codes is in the implementation file in a routine called *MapRC*.

Program Interface

The method of requesting services from the router is to use software interrupts and PC registers to pass the verbs and parameters. Although the use of software interrupts and registers lends itself to assembler language type interfaces, many other PC languages and compilers provide access to the software interrupts and registers. The next section is an example of using high-level languages to perform router requests.

Finding the Router Interrupt

The router must be loaded before your programs can request services. The router is loaded when the first STARTRTR command is run and remains loaded until it is removed with the RMVPCS command.

Normally, the X'68' interrupt vector is used to request services from the router. The interrupt vector can be changed to any interrupt vector from X'60' to X'68' by using the INTL parameter in the PC Support configuration file (CONFIG.PCS).

Interrupt vectors are 4-byte fields that contain pointers to the program instructions to run when the interrupt occurs. Your programs can use these pointers to look for the router's signature or to see if the router is loaded. Since the router can be installed in any interrupt vector from X'60' to X'68', your program should search all of these vectors looking for the router's signature. The logic for searching for the signature is as follows:

```
DO i = 60H to 68H while not_found;
  Call DOS function 35H to get Interrupt_Vector(i) value;
  Add 5 bytes to the pointer from Interrupt_Vector(i);
  If the resulting pointer points to the ASCII string 'APPC'
    the router is found and the interrupt is i;
END;
```

In order to determine the interrupt being used by the router, the applications must search the interrupt vector table looking for the signature used by the router. Only interrupt vectors X'60' through X'68' need be searched.

Microsoft C Version 5.1 provides the *_dos_getvect* function call that retrieves the value of an interrupt vector. The value in the interrupt vector is a pointer to the start of the router program. The router program can be identified by looking 5 bytes past the start of the program for the characters APPC.

The following is an example of how to determine which interrupt is being used by the router. If the router is found, the value for the router interrupt being used is saved in a variable called *routerInterrupt* so that subsequent calls can use it. If the router is not found on any of these calls, it has not been started.

```
/*============================================================================*/
/* FindRtr                                                                    */
/*   - Checks if PC Support router is installed and returns TRUE or FALSE     */
/*   - Sets global variable routerInterrupt to the interrupt being used by    */
/*     the router.  The router can use any interrupt from X'60' to X'68'.      */
/*============================================================================*/

int FindRtr(void)
{
  int  i;
  char (far *handler_ptr) [9];

  routerInterrupt=0;
  for (i=0x60; i<=0x68; i++)
    {
     handler_ptr =(char (far *)[9]) _dos_getvect(i);
     if ((*handler_ptr)[5] == 'A'  &&
         (*handler_ptr)[6] == 'P'  &&
         (*handler_ptr)[7] == 'P'  &&
         (*handler_ptr)[8] == 'C'     )
      {
       routerInterrupt = i;
       return(TRUE);
      }
    }
  return(FALSE);
}
```

The following routine shows how the interrupt can be called:

```
/*============================================================================*/
/* CallRouter                                                                 */
/*      Calls the router using the interrupt for the router                   */
/*      determined by FindRtr.                                                */
/*      Input: ADDRESS - far ptr to router verb control block                 */
/*             Request - type of verb (1=service 1=conversation)              */
/*============================================================================*/
CallRouter(void far *address, int  Request)
{
  union REGS       inregs,outregs;       /* defined in <dos.h>  */
  struct SREGS     segregs;              /* defined in <dos.h>  */
  int              result;

  inregs.x.dx = FP_OFF(address);
  segregs.ds  = FP_SEG(address);
  inregs.x.ax = Request * 0x100;

  result = int86x(routerInterrupt,&inregs,&outregs,&segregs);

  if(outregs.x.cflag)
    {
     printf("\n error in CallRouter interrupt - ax= %u\n", result);
    }
}
```

Issuing Requests to the Router

To use a router verb, your program must do the following:

1. Set register AH value to indicate the type of verb:

   1 = service verb
   2 = conversation verb

2. Set register pair DS:DX to point to the associated verb record.
3. Use the software interrupt operation with the router interrupt number as the operand.

When the router has finished processing the verb, the router sets a return code in the parameter list and returns control to your program. Your program should then examine the return code to determine the results of the request.

Data Conversion

The router maintains two translation tables: ASCII to EBCDIC and EBCDIC to ASCII. The following code segments show how to get access to these tables:

```
/*=========================================================================*/
/* GetTablePointers                                                        */
/*      Calls router to get pointers to ASCII to EBCDIC and                */
/*      EBCDIC to ASCII tables.                                            */
/*=========================================================================*/
GetTablePointers(void)
{
  struct controlrec
    {
      unsigned char rc;
      unsigned char op;
      unsigned char (far *tblptr)[256];
    } ctlrec, far *verbAddress = &ctlrec;

  ctlrec.rc = 0x00;
  ctlrec.op = 0x08;
  CallRouter(verbAddress,controlRequest);
  A2Eptr = ctlrec.tblptr;

  ctlrec.rc = 0x00;
  ctlrec.op = 0x09;
  CallRouter(verbAddress,controlRequest);
  E2Aptr = ctlrec.tblptr;

}
```

The following code segment uses the table for translations:

```
/*============================================================*/

ConvertToEBCDIC(unsigned char *String,
                int  Length     )
{
  int  i;
  for(i=0;i<Length;i++)
    String[i]=(*A2Eptr)[String[i]];
}
/*============================================================*/


ConvertToASCII(unsigned char *String,
               int  Length     )
{
  int  i;
  for(i=0;i<Length;i++)
    String[i]=(*E2Aptr)[String[i]];
}
```

PS/2 Program Detailed Description

1. **Allocate a conversation with the corresponding AS/400 module RTRREQ, which resides in library JDE.**

   A unique conversation ID is returned by the router for use by the application in all subsequent operations to identify the conversation.

   The format of the *tp_name* field, which contains the name of the AS/400 program, is PROGRAM NAME.LIBRARY NAME. It must be converted to the EBCDIC format.

   The allocate requests a mapped conversation with a synchronization level of none. The buffer size is set to 271 bytes and the variable *locName* contains a null string (this is the method used to request the default system). The *tpName* variable contains the name of the host program to start. No pip data is sent because the pip length parameter is set to 0. The *conv_id* field is returned from the allocate request. All subsequent verbs must use this value to request functions on this conversation.

```
char              pip;
void far          *conv_id;        /*  conversation id */
char far *tpName = "RTRREQ.JDE";
char far *locName= "";

 /* Allocate a mapped conversation                          */
      rc = Allocate(271,
                    Mapped,
                    syncLevelNone,
                    locName,
                    tpName,
                    0,
                    &pip,
                    &conv_id);
        if (rc != 0)
          quit(rc);
```

2. **Accept input from the terminal user and save it in a variable.**

```
printf("Enter Part Number = ");
for (i=0;i<6;i++)
    {
       part[i] = getchar();
       part[i] = toupper(part[i]);
    }
printf("%\n");
```

3. **Send the input to the AS/400 system.**

   The format of the transaction sent to the AS/400 system is as follows:

	Length	Use
Transaction length	2	Trx length
Gdsid	2	X'12FF'
Application data	5 (user defined)	Application data

   The length field contains a value equal to the length of the application data plus 4. The value in this field must be transformed from the PC format to the AS/400 format. In this example, the field will contain X'0009' (AS/400 format) rather than X'0900' (PC format). The generalized data stream identifier (gdsid) is set to X'12FF' for a mapped conversation.

   The application data must be converted from ASCII format to EBCDIC format.

   There are two send data requests in this program. The first send request sends the LL and GDSID for the record. This send is immediately followed with the data (part number). Notice that the record is sent in two parts; this is perfectly valid and often convenient. It would also be valid to send the header and the data in one send request, as long as the record is set up with the LL and GDSID before the data value.

```
unsigned char   hdr[4];
unsigned char   part[5];
unsigned char   part_eb[5];

/* the part number will be passed to the AS/400            */
/* application program in EBCDIC format                    */
        memcpy(&part_eb,&part,5);
        ConvertToEBCDIC(&part_eb[0],5);
/* First send LL and GDSID                                 */
        rc = SendData(conv_id,
                      4,
                      hdr,
                      &rqs);
        if (rc != 0)
          quit(rc);
/* Now send part number                                    */
        rc = SendData(conv_id,
                      5,
                      part_eb,
                      &rqs);
        if (rc != 0)
          quit(rc);
```

4. **Wait for a response from the AS/400 system.**

   The format of the transaction received from the AS/400 system is as follows:

	Length	Use
Transaction length	2	Trx length
Gdsid	2	X'12FF'
Application data	70 (user defined)	Application data

The application data must be converted from EBCDIC to ASCII for use on the PS/2 system.

The *max_len* field must be set to a value large enough to handle the data to be received from the AS/400 program.

The receive and wait verb is issued to receive the information from the AS/400 system. When the receive and wait verb is issued, the local program goes into the receive state and control of the session is given to the remote program. A second receive and wait verb must be issued to receive the Send indication (in the what received parameter) before the local program can send any more data. The send indication must be explicitly received by the local program before it goes into send state.

```
struct data{
            unsigned char    hdr[4];
            unsigned char    data[70&brk.;
} input;
char            what;


/* Wait for data to be returned                                    */
          rc = ReceiveAndWait(conv_id,
                              ll,
                              80,
                              &input,
                              &what,
                              &reqsToSend,
                              &receivedLength);
          if (rc != 0)
            quit(rc);
/* Must issued second receive to get Send indicator                */
          rc = ReceiveAndWait(conv_id,
                              ll,
                              0,
                              &input,
                              &what,
                              &reqsToSend,
                              &receivedLength);
          if (rc != 0)
            quit(rc);

          if (what != receivedSend)
             _exit;
```

5. **Format the response to the screen.**

   If position 31 is not blank, it indicates that the part number was not found in the AS/400 database.

```
if (verb.dlen > 0)
 {
  printf("Part Number  = ");
  for (i=0;i<5;i++)
       printf("%1c",part[i]);
  printf("%\n");
  ebcdic_to_ascii(input.data,70);
  if (input.data[30] == ' ')
   {
    printf("Description  = ");
    for (i=0;i<25;i++)
       printf("%1c",input.data[i]);
    printf("%\n");
    printf("Quantity     = ");
    for (i=26;i<31;i++)
       printf("%1c",input.data[i]);
   }
   else
   {
    printf("%\n");
    printf("***********Part Number not Found***********");
   }
    printf("%\n");
    printf("*** Enter new Part Number or 'ENDPGM' to stop  ***");
    printf("%\n");
    printf("%\n");
  }
```

6. **When the user is finished, deallocate the session.**

```
Deallocate(conv_id,deallocateSyncLevel);
```

Data Structures and Code Points

This section contains the data structures, operation codes, return codes, and verb parameter values for the router API.

Operation Codes

```
/*************************************************************/
/*              Operation Codes                              */
/*************************************************************/

   #define  AP_B_ALLOCATE                          0x01
   #define  AP_B_CONFIRM                           0x03
   #define  AP_B_CONFIRMED                         0x04
   #define  AP_B_DEALLOCATE                        0x05
   #define  AP_B_FLUSH                             0x06
   #define  AP_B_GET_ATTRIBUTES                    0x07
   #define  AP_B_PREPARE_TO_RECEIVE                0x0A
   #define  AP_B_RECEIVE_AND_WAIT                  0x0B
   #define  AP_B_RECEIVE_IMMEDIATE                 0x0C
   #define  AP_B_REQUEST_TO_SEND                   0x0E
   #define  AP_B_SEND_DATA                         0x0F
   #define  AP_B_SEND_ERROR                        0x10
```

Primary Return Codes

```
/* PRIMARY Return Codes */

   #define  AP_OK                                  0x0000
   #define  AP_PARAMETER_CHECK                     0x0100
   #define  AP_STATE_CHECK                         0x0200
   #define  AP_ALLOCATION_ERROR                    0x0300
   #define  AP_DEALLOC_ABEND                       0x0500
   #define  AP_DEALLOC_ABEND_PROG                  0x0600
   #define  AP_DEALLOC_ABEND_SVC                   0x0700
   #define  AP_DEALLOC_ABEND_TIMER                 0x0800
   #define  AP_DEALLOC_NORMAL                      0x0900
   #define  AP_PROG_ERROR_NO_TRUNC                 0x0C00
   #define  AP_PROG_ERROR_TRUNC                    0x0D00
   #define  AP_PROG_ERROR_PURGING                  0x0E00
   #define  AP_CONV_FAILURE_RETRY                  0x0F00
   #define  AP_CONV_FAILURE_NO_RETRY               0x1000
   #define  AP_SVC_ERROR_NO_TRUNC                  0x1100
   #define  AP_SVC_ERROR_PURGING                   0x1300
   #define  AP_SVC_ERROR_TRUNC                     0x1200
   #define  AP_UNSUCCESSFUL                        0x1400
   #define  AP_APPC_BUSY                           0x02F0
```

Secondary Return Codes

```
 /* Secondary Return Code - Parameter Check */

   #define  AP_INVALID_VERB                        0x00000000
   #define  AP_BAD_CONV_ID                         0x02000000
   #define  AP_DATA_AREA_ACROSS_SEGMENT            0x06000000
   #define  AP_TP_NAME_LENGTH_ERROR                0x10000000
   #define  AP_SYNC_LEVEL_NOT_SUPPORTED            0x12000000
   #define  AP_RETURN_CONTROL_NOT_SUPPORTED        0x14000000
   #define  AP_PIP_LEN_INCORRECT                   0x16000000
   #define  AP_UNKNOWN_PARTNER                     0x18000000
   #define  AP_CONFIRM_ON_SYNC_LEVEL_NONE          0x31000000

 /* Secondary Return Code - State Check */
   #define  AP_CONFIRMED_BAD_STATE                 0x41000000
   #define  AP_RCV_IMMD_BAD_STATE                  0xC1000000
   #define  AP_R_T_S_BAD_STATE                     0xE1000000
   #define  AP_NOT_SEND_STATE                      0xF2000000
   #define  AP_SEND_ERROR_BAD_STATE                0x01010000


 /* Secondary Return Code - Allocate Failure */
   #define  AP_ALLOCATION_FAILURE_NO_RETRY         0x04000000
   #define  AP_ALLOCATION_FAILURE_RETRY            0x05000000
   #define  AP_TRANS_PGM_NOT_AVAIL_NO_RETRY        0x00004C08
   #define  AP_TP_NAME_NOT_RECOGNIZED              0x21600810
   #define  AP_TRANS_PGM_NOT_AVAIL_RETRY           0x31604B08
   #define  AP_SECURITY_NOT_VALID                  0x51600F08
   #define  AP_CONVERSATION_TYPE_MISMATCH          0x34600810
   #define  AP_PIP_NOT_ALLOWED                     0x31600810
   #define  AP_PIP_NOT_SPECIFIED_CORRECTLY         0x32600810
   #define  AP_SYNC_LEVEL_MISMATCH                 0x41600810
```

Verb Parameter Values

```
/* Request to Send Received */
  #define  AP_NO                                  0x00
  #define  AP_YES                                 0x01

/* Conversation Type            */
  #define  AP_BASIC_CONVERSATION                  0x00
  #define  AP_MAPPED_CONVERSATION                 0x01
/* Deallocate Types             */
  #define  AP_SYNCH_LEVEL                         0x00
  #define  AP_FLUSH                               0x01
  #define  AP_ABEND_PROG                          0x02
/* Fill parameter on RcvAndWait and RcvImmediate */
  #define  AP_BUFFER                              0x00
  #define  AP_LL                                  0x01
/* SyncLevel on Allocate        */
  #define  AP_NONE                                0x00
  #define  AP_CONFIRM                             0x01

/* What received values */

  #define  AP_DATA                                0x00
  #define  AP_DATA_COMPLETE                       0x01
  #define  AP_DATA_INCOMPLETE                     0x02
  #define  AP_CONFIRM_WHAT_RECEIVED               0x03
  #define  AP_CONFIRM_SEND                        0x04
  #define  AP_CONFIRM_DEALLOCATE                  0x05
  #define  AP_SEND                                0x06
```

Verb Structures

```
/*************************************************************/
/*                  Common Verb Header                       */
/*************************************************************/

 struct appc_hdr
     {
     unsigned char   res1[12];          /* 0 reserved                */
     unsigned char   opcode;            /* 12 Verb operation code    */
     unsigned char   res2[7];           /* 13 reserved               */
     unsigned short  primary_rc;        /* 20 Primary RETURN CODE    */
     unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE  */
     unsigned char   res3[8];           /* 26 reserved               */
     unsigned long   conv_id;           /* 34 CONV_ID                */
     };


/*************************************************************/
/*            Application Control Interface                  */
/*************************************************************/

/*************************************************************/
/*Transaction Programming Interface - Basic Conversation  */
/*************************************************************/

 struct allocate
     {

     unsigned char   res1[12];           /* 0 reserved               */
     unsigned char   opcode;             /* 12 Verb operation code   */
     unsigned char   res2[7];            /* 13 reserved              */
     unsigned short  primary_rc;         /* 20 Primary RETURN CODE   */
     unsigned long   secondary_rc;       /* 22 Secondary RETURN CODE */
     unsigned char   res3[8];            /* 26 reserved              */
     unsigned long   conv_id;            /* 34 CONV_ID               */
     unsigned char   conv_type;          /* CONVERSATION_TYPE        */
                                         /*    AP_BASIC_CONVERSATION */
                                         /*    AP_MAPPED_CONVERSATION*/
     unsigned char   sync_level;         /* SYNC_LEVEL               */
                                         /*    AP_NONE               */
                                         /*    AP_CONFIRM            */
     unsigned char   res4[11];           /* Reserved                 */
     unsigned char   plu_name[8];        /* PARTNER_LU_NAME          */
     unsigned char   res5[8];            /* MODE_NAME                */
     unsigned char   tpn_length;         /* length of tp name        */
     unsigned char   tp_name[64];        /* TP_NAME                  */
     unsigned char   res6[34];           /* reserved                 */
     unsigned short  pip_dlen;           /* PIP_DATA_LENGTH          */
                                         /*   (0 if no pip data)     */
     unsigned char   far *pip_dptr;      /* PIP_DATA                 */
     };
```

```
struct confirm
    {

    unsigned char   res1[12];          /* 0 reserved                  */
    unsigned char   opcode;            /* 12 Verb operation code      */
    unsigned char   res2[7];           /* 13 reserved                 */
    unsigned short  primary_rc;        /* 20 Primary RETURN CODE      */
    unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE    */
    unsigned char   res3[8];           /* 26 reserved                 */
    unsigned long   conv_id;           /* 34 CONV_ID                  */
    unsigned char   rts_rcvd;          /* REQUEST_TO_SEND_RECEIVED    */
                                       /*    AP_NO                    */
                                       /*    AP_YES                   */
    };


struct confirmed
    {

    unsigned char   res1[12];          /* 0 reserved                  */
    unsigned char   opcode;            /* 12 Verb operation code      */
    unsigned char   res2[7];           /* 13 reserved                 */
    unsigned short  primary_rc;        /* 20 Primary RETURN CODE      */
    unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE    */
    unsigned char   res3[8];           /* 26 reserved                 */
    unsigned long   conv_id;           /* 34 CONV_ID                  */
    };


struct deallocate
    {
    unsigned char   res1[12];          /* 0 reserved                  */
    unsigned char   opcode;            /* 12 Verb operation code      */
    unsigned char   res2[7];           /* 13 reserved                 */
    unsigned short  primary_rc;        /* 20 Primary RETURN CODE      */
    unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE    */
    unsigned char   res3[8];           /* 26 reserved                 */
    unsigned long   conv_id;           /* 34 CONV_ID                  */
    unsigned char   res4;              /* Reserved                    */
    unsigned char   dealloc_type;      /* TYPE                        */
                                       /*    AP_SYNCH_LEVEL           */
                                       /*    AP_FLUSH                 */
                                       /*    AP_ABEND_PROG            */
    };


 struct flush
    {
    unsigned char   res1[12];          /* 0 reserved                  */
    unsigned char   opcode;            /* 12 Verb operation code      */
    unsigned char   res2[7];           /* 13 reserved                 */
    unsigned short  primary_rc;        /* 20 Primary RETURN CODE      */
    unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE    */
    unsigned char   res3[8];           /* 26 reserved                 */
    unsigned long   conv_id;           /* 34 CONV_ID                  */
    };
```

```
struct get_attributes
   {
   unsigned char   res1[12];          /* 0 reserved                 */
   unsigned char   opcode;            /* 12 Verb operation code     */
   unsigned char   res2[7];           /* 13 reserved                */
   unsigned short  primary_rc;        /* 20 Primary RETURN CODE     */
   unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE   */
   unsigned char   res3[8];           /* 26 reserved                */
   unsigned long   conv_id;           /* 34 CONV_ID                 */
   unsigned char   res4[9];           /* Reserved                   */
   unsigned char   sync_level;        /* SYNC_LEVEL                 */
                                      /*    AP_NONE                 */
                                      /*    AP_CONFIRM              */
   unsigned char   res5[8];           /* Reserved                   */
   unsigned char   res6[8];           /* Reserved                   */
   unsigned char   lu_name[8];        /* LU_NAME                    */
   unsigned char   plu_name[8];       /* PARTNER_LU_NAME            */
   unsigned char   res7[19];          /* Reserved                   */
   unsigned char   userid_Len;        /* Length of User ID          */
   unsigned char   user_id[10];       /* USER_ID                    */
   };


struct prepare_to_receive
   {
   unsigned char   res1[12];          /* 0 reserved                 */
   unsigned char   opcode;            /* 12 Verb operation code     */
   unsigned char   res2[7];           /* 13 reserved                */
   unsigned short  primary_rc;        /* 20 Primary RETURN CODE     */
   unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE   */
   unsigned char   res3[8];           /* 26 reserved                */
   unsigned long   conv_id;           /* 34 CONV_ID                 */
   };
```

```
struct receive_and_wait
   {
   unsigned char   res1[12];          /* 0 reserved                    */
   unsigned char   opcode;            /* 12 Verb operation code        */
   unsigned char   res2[7];           /* 13 reserved                   */
   unsigned short  primary_rc;        /* 20 Primary RETURN CODE        */
   unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE      */
   unsigned char   res3[8];           /* 26 reserved                   */
   unsigned long   conv_id;           /* 34 CONV_ID                    */
   unsigned char   what_rcvd;         /* WHAT_RECEIVED                 */
                                      /*    AP_DATA                    */
                                      /*    AP_DATA_COMPLETE           */
                                      /*    AP_DATA_INCOMPLETE         */
                                      /*    AP_SEND                    */
                                      /*    AP_CONFIRM                 */
                                      /*    AP_CONFIRM_SEND            */
                                      /*    AP_CONFIRM_DEALLOCATE      */
   unsigned char  fill;               /* FILL                          */
                                      /*    AP_BUFFER                  */
                                      /*    AP_LL                      */
   unsigned char  rts_rcvd;           /* REQUEST_TO_SEND_RECEIVED      */
                                      /*    AP_NO                      */
                                      /*    AP_YES                     */
   unsigned short  max_len;           /* MAX_LENGTH                    */
   unsigned short  dlen;              /* DATA_LENGTH                   */
   unsigned char   far *dptr;         /* DATA_PTR                      */
   };


struct receive_immediate
   {
   unsigned char   res1[12];          /* 0 reserved                    */
   unsigned char   opcode;            /* 12 Verb operation code        */
   unsigned char   res2[7];           /* 13 reserved                   */
   unsigned short  primary_rc;        /* 20 Primary RETURN CODE        */
   unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE      */
   unsigned char   res3[8];           /* 26 reserved                   */
   unsigned long   conv_id;           /* 34 CONV_ID                    */
   unsigned char   what_rcvd;         /* WHAT_RECEIVED                 */
                                      /*    AP_DATA                    */
                                      /*    AP_DATA_COMPLETE           */
                                      /*    AP_DATA_INCOMPLETE         */
                                      /*    AP_SEND                    */
                                      /*    AP_CONFIRM                 */
                                      /*    AP_CONFIRM_SEND            */
                                      /*    AP_CONFIRM_DEALLOCATE      */
   unsigned char  fill;               /* FILL                          */
                                      /*    AP_BUFFER                  */
                                      /*    AP_LL                      */
   unsigned char  rts_rcvd;           /* REQUEST_TO_SEND_RECEIVED      */
                                      /*    AP_NO                      */
                                      /*    AP_YES                     */
   unsigned short  max_len;           /* MAX_LENGTH                    */
   unsigned short  dlen;              /* DATA_LENGTH                   */
   unsigned char   far *dptr;         /* DATA_PTR                      */
   };
```

```
struct request_to_send
   {
   unsigned char   res1[12];          /* 0 reserved                 */
   unsigned char   opcode;            /* 12 Verb operation code     */
   unsigned char   res2[7];           /* 13 reserved                */
   unsigned short  primary_rc;        /* 20 Primary RETURN CODE     */
   unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE   */
   unsigned char   res3[8];           /* 26 reserved                */
   unsigned long   conv_id;           /* 34 CONV_ID                 */
   };


struct send_data
   {
   unsigned char   res1[12];          /* 0 reserved                 */
   unsigned char   opcode;            /* 12 Verb operation code     */
   unsigned char   res2[7];           /* 13 reserved                */
   unsigned short  primary_rc;        /* 20 Primary RETURN CODE     */
   unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE   */
   unsigned char   res3[8];           /* 26 reserved                */
   unsigned long   conv_id;           /* 34 CONV_ID                 */
   unsigned char   rts_rcvd;          /* REQUEST_TO_SEND_RECEIVED   */
                                      /*    AP_NO                   */
                                      /*    AP_YES                  */
   unsigned char   res4;              /* reserved                   */
   unsigned short  dlen;              /* DATA_LENGTH                */
   unsigned char   far *dptr;         /* DATA_PTR                   */
   };


struct send_error
   {
   unsigned char   res1[12];          /* 0 reserved                 */
   unsigned char   opcode;            /* 12 Verb operation code     */
   unsigned char   res2[7];           /* 13 reserved                */
   unsigned short  primary_rc;        /* 20 Primary RETURN CODE     */
   unsigned long   secondary_rc;      /* 22 Secondary RETURN CODE   */
   unsigned char   res3[8];           /* 26 reserved                */
   unsigned long   conv_id;           /* 34 CONV_ID                 */
   unsigned char   rts_rcvd;          /* REQUEST_TO_SEND_RECEIVED   */
                                      /*    AP_NO                   */
                                      /*    AP_YES                  */
   };
```

RTRREQ.C Program Source

The following listing shows the RTRREQ.C program. The APPCSUBS file, which is included, follows this section. This program was compiled using the Microsoft C 6.0 compiler. The large memory model was specified.

```
#define LINT_ARGS

#include <appcsubs.h>
#include <stdio.h>
#include <stddef.h>
#include <string.h>
#include <dos.h>
#include <malloc.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0


        char            pip;
        void far        *conv_id;         /*  conversation id */
        char far *tpName = "RTRREQ.JDE";
        char far *locName= "";
        unsigned char   hdr[4];
        unsigned char   part[5];
        unsigned char   part_eb[5];
        int                  i;
        char               rqs;
        ReturnCode          rc;
        struct data{
                    unsigned char    hdr[4];
                    unsigned char    data[70];
        } input;
        char            what;
        char            reqsToSend;
        int             receivedLength;

void do_getPart()
   {
         printf("Enter Part Number = ");
         for (i=0;i<6;i++)
             {
                part[i] = getchar();
                part[i] = toupper(part[i]);
             }
         printf("%\n");
   }
```

```
void do_format()
   {

      printf("Part Number  = ");
      for (i=0;i<5;i++)
           printf("%1c",part[i]);
      printf("%\n");
      ConvertToASCII(&input.data[0],70);
      if (input.data[30] == ' ')
       {
        printf("Description  = ");
        for (i=0;i<25;i++)
           printf("%1c",input.data[i]);
        printf("%\n");
        printf("Quantity     = ");
        for (i=26;i<31;i++)
           printf("%1c",input.data[i]);
       }
       else
       {
        printf("%\n");
        printf("***********Part Number not Found***********");
       }
        printf("%\n");
        printf("*** Enter new Part Number or 'ENDPGM' to stop  ***");
        printf("%\n");
        printf("%\n");
    }

void quit(ReturnCode rv)
    {
      printf("Error Return Code = %02u\n",rv);
      _exit(99);
    }
void main()
    {
 /* Set up header for a Mapped conversation data record      */
       hdr[0] = 0;                      /* LL byte one       */
       hdr[1] = 9;                      /* LL byte two       */
       hdr[2] = 0x12;                   /* GDSID byte one    */
       hdr[3] = 0xff;                   /* GDSID byte two    */
 /* Allocate a mapped conversation                           */
       rc = Allocate(271,
                     Mapped,
                     syncLevelNone,
                     locName,
                     tpName,
                     0,
                     &pip,
                     &conv_id);
         if (rc != 0)
           quit(rc);
         do_getPart();
         do
          {
```

```
/* the part number will be passed to the AS/400                    */
/* application program in EBCDIC format                            */
        memcpy(&part_eb,&part,5);
        ConvertToEBCDIC(&part_eb[0],5);
/* First send LL and GDSID                                         */
        rc = SendData(conv_id,
                      4,
                      hdr,
                      &rqs);
        if (rc != 0)
          quit(rc);
/* Now send part number                                            */
        rc = SendData(conv_id,
                      5,
                      part_eb,
                      &rqs);
        if (rc != 0)
          quit(rc);
/* Wait for data to be returned                                    */
        rc = ReceiveAndWait(conv_id,
                            ll,
                            80,
                            &input,
                            &what,
                            &reqsToSend,
                            &receivedLength);
        if (rc != 0)
          quit(rc);
/* Must issued second receive to get Send indicator                */
        rc = ReceiveAndWait(conv_id,
                            ll,
                            0,
                            &input,
                            &what,
                            &reqsToSend,
                            &receivedLength);
        if (rc != 0)
          quit(rc);

        if (what != receivedSend)
           _exit;
      do_format();
      do_getPart();
       }
      while (part[0] != 'E');
/* when finished deallocate conversation                           */
      Deallocate(conv_id,deallocateSyncLevel);
}
```

APPCSUBS.H Listing

The interface to the APPC subroutines is in this section. The actual code for the subroutines may be found in the PC Support tools folder. Note that the structures defining the verbs are prefixed with the following line:

```
#pragma pack(1)
```

This line causes the structures to be packed. All verb structures passed to the router must be packed for the router to understand the requests. If you send infor-

mation to the AS/400 system and define the records inside of structures, these structures may also have to be packed for the AS/400 program to correctly understand the information.

The return codes created by these subroutines may be found in the PC Support tools folder. Note that the return codes are defined as an enumeration. This method makes it easier for main programs to deal with all the possible APPC return codes.

```
/*====================================================================*/
/*  APPCSUBS.H                                                        */
/*====================================================================*/
/* This module provides the structures, definitions, and function     */
/* prototypes for the APPC interfaces.                                */
/*                                                                    */
/* An enumeration called ReturnCodes has been defined to provide      */
/* feedback on your request.  These return codes are easier to        */
/* deal with than the two part APPC return codes.                     */
/* In all cases where a string parameter is provided by your          */
/* application, these routines will translate it to EBCDIC if         */
/* necessary.  This means you only need to work with ASCII strings,   */
/* except for the data you are sending to your partner program.       */
/* It is your responsibility to format the data correctly.            */
/*====================================================================*/


#pragma pack(1)

typedef enum {  /* enums for "type" on Allocate requests */
    Basic = 0,
    Mapped = 1
} ConversationType;

typedef enum {  /* enums for "whatReceived" on Receive requests */
    data = 0,
    dataComplete = 1,
    dataIncomplete = 2,
    receivedConfirm = 3,
    receivedConfirmSend = 4,
    receivedConfirmDeallocate = 5,
    receivedSend = 6
} WhatReceivedEnum;

typedef enum {  /* for "fill" on Receive requests */
    buffer = 0,
    ll = 1
} FillEnum;
```

```
typedef enum {  /* for "synchLevel" on Allocate requests */
    syncLevelNone    = 0,
    syncLevelConfirm = 1
} SyncLevelEnum;

typedef enum {  /* for "type" on Deallocate requests */
    deallocateSyncLevel  = 0,
    deallocateFlush      = 1,
    deallocateAbend      = 2
} DeallocateEnum;

typedef enum {  /* for "opcode" on all requests */
    allocate = 1,
    confirm = 3,
    confirmed = 4,
    deallocate = 5,
    flush = 6,
    getAttributes = 7,
    prepareToReceive = 10,
    receiveAndWait = 11,
    receiveImmediate = 12,
    requestToSend = 14,
    sendData = 15,
    sendError = 16
} Opcodes;

typedef enum { /* list return codes from all router verb routines */
    Ok = 0,
    DeAllocNormal = 1,
    ProgramErrNoTruncation = 2,
    ProgramErrTruncation = 3,
    ProgramErrPurging = 4,
    ResourceFailureRetry = 5,
    ResourceFailureNoRetry = 6,
    Unsuccessful = 7,
    APPCBusy = 8,
    ParmChkInvalidVerb = 9,
    ParmChkInvalidConverId = 10,
    ParmChkBufferCrossesSegment = 11,
    ParmChkTPNameLength = 12,
    ParmChkInvalidConverType = 13,
    ParmChkBadSynchLevelOnAlloc = 14,
    ParmChkBadReturnControl = 15,
    ParmChkPipTooLong = 16,
    ParmChkBadPartnerName = 17,
    ParmChkConfNotAllowed = 18,
    ParmChkBadDeAllocType = 19,
    ParmChkPrepareToRcvType = 20,
    ParmChkBadFillType = 21,
    ParmChkReceiveMaxLengthTooBig = 22,
    StateChkConfIssuedNotInConfStat = 23,
    StateChkRcvImmedAndNotInReceive = 24,
    StateChkRequestToSendInBadState = 25,
    StateChkSendIssuedInBadState = 26,
    StateChkSendErrInBadState = 27,
```

```
    AllocErrNoRetry = 28,
    AllocErrRetry = 29,
    AllocErrPgmNotAvailableNoRetry = 30,
    AllocErrTpnNotRecognized = 31,
    AllocErrPgmNotAvailableRetry = 32,
    AllocErrSecurityNotValid = 33,
    AllocErrConverTypeMisMatch = 34,
    AllocErrPipNotAllowed = 35,
    AllocErrPipNotCorrect = 36,
    AllocErrSynchLevelNotSupported = 37,
    DeAllocAbendProgram = 38,
    RouterNotInstalled = 39,
    InsufficientMemory = 40,
    Undefined = 41,
    TooManyConversations = 42
} ReturnCode;

/*************************************************************************/
/* The following are external function prototypes for issuing the        */
/* router verbs.  All return an integer that represents the 6-byte actual */
/* router return code.  The 6-byte router return code is mapped to one    */
/* of the values in the previous ReturnCode enumeration.                 */
/*************************************************************************/
extern int
Allocate(unsigned          bufferLength,     /* Alloc will create buffer  */
         char              type,             /* Basic or Mapped           */
         char              synchLevel,
         char             *locationName,     /* Pass as ASCII string      */
         char             *tpn,              /* Pass as ASCII string, if  */
                                             /* first char less than      */
                                             /* X'40' no translation done */
         int               pipLength,
         void             *pipData,          /* no translation done, must */
                                             /* be in GDS format          */
         void far         **conversation   );/* set by router,treat       */
                                             /* like file handle          */
/* allocates the conversation, returning the conversation handle needed   */
/* for all future calls on this conversation                              */

extern int
SendData(void far *conversation,    /* conversation handle (from Allocate) */
         int       sendDataLength,  /* length of data in Send buffer       */
         void     *sendDataBuffer,  /* address of Send buffer              */
         char     *requestToSendRcvd);  /* returns whether request_to_send */

extern int
ReceiveAndWait(void far        *conversation,
               char             fill,
               int              maxLength,
               void            *receiveData,
               char            *whatReceived,
               char            *requestToSendRcvd,
               int             *receiveDataLength );
```

```
extern int
ReceiveImmediate(void far        *conversation,
                 char            fill,
                 int             maxLength,
                 void            *receiveData,
                 char            *whatReceived,
                 char            *requestToSendRcvd,
                 int             *receiveDataLength );

extern  int
Deallocate(void far      *conversation,
           char          type          );
extern  int
Confirm(void far *conversation,
        char             *requestToSendRcvd);
extern  int
SendError(void far *conversation,
          char             *requestToSendRcvd);

extern  int
PrepareToReceive(void far *conversation);

extern  int
Confirmed(void far *conversation);

extern  int
RqsToSend(void far *conversation);

/*************************************************************************/
/* The following are miscellaneous router interface routines             */
/* for doing functions involving or requiring router functions.          */
/*************************************************************************/

CallRouter(void far *address, int  Request);
  static unsigned const verbRequest = 2;
  static unsigned const controlRequest = 1;
/* calls the router with the address of a filled in control block        */

int MapRC(unsigned primary,
          unsigned secondary1,
          unsigned secondary2);
/* maps the 6 byte return code in the control block to an integer value  */

ConvertToASCII(unsigned char *String,
               int  Length     );
/* converts a string of EBCDIC characters to ASCII using the router's    */
/* built-in translation tables                                           */
/* NOTE: only to be used AFTER at least one Allocate call has been made  */

ConvertToEBCDIC(unsigned char *String,
                int  Length     );
/* converts a string of ASCII characters to EBCDIC using the router's    */
/* built-in translation tables                                           */
/* NOTE: only to be used AFTER at least one Allocate call has been made  */
```

Modula-2 Program Example

This section shows an example application using the MODULA-2 programming language. Modula-2 is a modern high-level language that is adapted to separating parts of an application and reusing modules. In this example, a separate module called APPC is defined. The APPC module provides the interface to the router. The interface is described in a file called APPC.DEF and is shown later in this section. The implementation is in a file called APPC.MOD and can be found in the PC Support tools folder. The compiler used in this example was the TOPSPEED** Modula-2 Compiler Version 1.17.

Finding the Router Interrupt

In order to determine the interrupt being used by the router, the application program must search the interrupt vector table looking for the signature used by the router. Only interrupt vectors X'60' through X'68' should be searched.

The value in the interrupt vector is a pointer to the start of the router program. The router program can be identified by looking 5 bytes past the start of the program for the characters APPC.

The following is an example of how to determine which interrupt is being used by the router. If the router is found, the value for the router interrupt being used is saved in a variable called *routerInterrupt* so that subsequent calls can use it. If the router is not found on any of these calls, it has not been started.

```
(* The following types are used to find router, and call router *)
  TYPE RtrId = RECORD
   Fill :       ARRAY[0..4] OF CHAR;
   Id   :       ARRAY[0..3] OF CHAR;
  END;
  TYPE RtrPtr = POINTER TO RtrId;
  TYPE RouterInterruptArray = ARRAY[60H..68H] OF RtrPtr;

  VAR interruptVectors[0:180H]:RouterInterruptArray ;
  VAR routerInterrupt: CARDINAL;
 PROCEDURE FindRtr():BOOLEAN;


  VAR
   i : CARDINAL;
```

```
BEGIN;
  routerInterrupt := 0;
  i := 60H;
  WHILE (routerInterrupt = 0) AND (i <= 68H) DO;
    WITH interruptVectors[i]^ DO;
      IF (Id[0] = 'A') AND
         (Id[1] = 'P') AND
         (Id[2] = 'P') AND
         (Id[3] = 'C') THEN
        routerInterrupt := i;
      ELSE
        INC(i);
      END;
    END;
  END;
  IF routerInterrupt = 0 THEN
    RETURN(FALSE);
  ELSE
    RETURN(TRUE);
  END;
END FindRtr;
```

The following routine shows how the interrupt can be called:

```
TYPE RouterRequest = (undefined,controlRequest,verbRequest);

PROCEDURE CallRouter(addr:ADDRESS;rType:RouterRequest);

VAR type:      CARDINAL;
    r:         Registers;
BEGIN;

 r.AX := ORD(rType)*100H;   (* need to force type into AH *)
 r.DX := Ofs(addr^);
 r.DS := Seg(addr^);
 Intr(r,routerInterrupt);
END CallRouter;
```

Data Conversion

The router maintains two translation tables: ASCII to EBCDIC and EBCDIC to ASCII. The following code segments show how to get access to these tables:

```
VAR a2EPtr:    A2EPtr;
VAR e2APtr:    E2APtr;

PROCEDURE GetTablePointers;

TYPE GetA2ERecord = RECORD
  rc:  CHAR;
  op:  CHAR;
  ptr: A2EPtr;
END;


TYPE GetE2ARecord = RECORD
  rc:  CHAR;
  op:  CHAR;
  ptr: E2APtr;
END;

VAR getA2E:            GetA2ERecord;
VAR getE2A:            GetE2ARecord;
VAR verbAddress:       ADDRESS;

BEGIN;
  getA2E.op := CHAR(8);
  getA2E.rc := CHAR(0);
  verbAddress:= ADR(getA2E);
  CallRouter(verbAddress,controlRequest);
  a2EPtr    := getA2E.ptr;

  getE2A.op := CHAR(9);
  getE2A.rc := CHAR(0);
  verbAddress:= ADR(getE2A);
  CallRouter(verbAddress,controlRequest);
  e2APtr    := getE2A.ptr;

END GetTablePointers;
```

The following code segment uses the table for translations:

```
PROCEDURE ConvertToASCII(
       VAR string      :ARRAY OF BYTE
       );

VAR i: CARDINAL;

 BEGIN;
  IF routerInstalled THEN
    FOR i:= 0 TO HIGH(string) DO;
      string[i] := e2APtr^[ORD(string[i])];
    END;
  END;

END ConvertToASCII;

PROCEDURE ConvertToEBCDIC(
       VAR string      :ARRAY OF BYTE
       );
 VAR i:        CARDINAL;

 BEGIN;
  IF routerInstalled THEN
    FOR i:= 0 TO HIGH(string) DO;
      string[i] := a2EPtr^[ORD(string[i])];
    END;
  END;
END ConvertToEBCDIC;
```

Program Detailed Description

The following are code segments and descriptions of the main portions of the RTRREQ program. A complete listing of the program follows the main parts of the program.

Allocate

The allocate verb requests a mapped conversation with a synchronization level of none. The buffer size is set to 271 bytes and the *locName* variable contains a null string (this is the method used to request the default system). The *tpName* variable contains the name of the host program to start. No pip data is sent because the pip length parameter is set to 0. The *conv_id* field is returned from the allocate request. All subsequent verbs must use this value to request functions on this conversation.

```
(* Allocate conversation with following:                          *)
(* -- set synch level to none                                    *)
(* -- set system name to null so that default will be used       *)
(* -- Name of program is RTRREQ in Library JDE                   *)
(* -- no PIP data is sent                                        *)
  rc := APPC.Allocate(
                      bufferSize,
                      APPC.syncLevelNone,
                      nul,
                      'RTRREQ.JDE',
                      0,
                      nul,
                      conversation);

(* if any errors write message indicating problem                 *)

  IF rc <> APPC.Ok THEN
    WriteReturnCode(rc);
    SetReturnCode(255);
    HALT ;
  END;
```

Send Data

The module APPC provides a mapped conversation interface, so calls do not have to be aware of the LLs and GDSIDs when sending records. Therefore, only one send data is needed to send the part number. The length and GDSID fields are set by the routine within the APPC module.

```
(* Send part number - module appc will handle ll and GDSID        *)
   rc := APPC.SendData(
                      conversation,
                      5,
                      inq,
                      requestToSend
                    );
   IF rc <> APPC.Ok THEN
     WriteReturnCode(rc);
     SetReturnCode(255);
     HALT ;
   END;
```

Receive and Wait

The receive and wait verb is issued to receive the information from the AS/400 system. When the receive and wait verb is issued, the local program goes into receive state and control of the session is given to the remote program. A second receive and wait verb must be issued to receive the Send indication (in the what received parameter) before the local program can send any more data. The send indication must explicitly be received by your program before it goes into send state.

```
(* Receive response                                                    *)
   rc := APPC.ReceiveAndWait(
                        conversation,
                        SIZE(dbRec),
                        dbRec,
                        whatReceived,
                        requestToSend,
                        receivedDataLength
                        );
   IF rc <> APPC.Ok THEN
     WriteReturnCode(rc);
     SetReturnCode(255);
     HALT ;
   END;

 (* receive and Wait to get send indication                           *)

 (* -- set max receive length to 0 since nothing is expected         *)
  rc := APPC.ReceiveAndWait(
                        conversation,
                        0,
                        dbRec,
                        whatReceived,
                        requestToSend,
                        receivedDataLength
                        );
   IF rc <> APPC.Ok THEN
     WriteReturnCode(rc);
     SetReturnCode(255);
     HALT ;
   END;
   IF whatReceived <> APPC.receivedSend THEN
     IO.WrStr('Send indicator not received from host ');
     HALT;
   END;
```

RTRREQ Program Listing

The following listing shows the RTRREQ program written in Modula-2. The APPC subroutine module, which is included, follows this listing.

```
MODULE RtrReq;


IMPORT APPC;
FROM WriteRC IMPORT WriteReturnCode;
IMPORT Strings;
FROM SYSTEM IMPORT SIZE;
FROM Lib IMPORT SetReturnCode;
IMPORT IO;

CONST bufferSize = 271;
      nul        = 0C;

TYPE PartString = ARRAY[0..4] OF CHAR;

TYPE InqReq = RECORD
  partNum      :PartString;
END;

TYPE PartDescription = ARRAY[0..24] OF CHAR;
TYPE Quantity       = ARRAY[0..4] OF CHAR;
TYPE ErrorMsg        = ARRAY[0..39] OF CHAR;

TYPE DBRecord = RECORD
  partds      :PartDescription;
  partq       :Quantity;
  errMsg      :ErrorMsg;
END;

CONST end = PartString('9','9','9','9','9');


VAR
    inq                :InqReq;
    requestToSend      :BOOLEAN;
    receivedDataLength :CARDINAL;
    whatReceived       :APPC.WhatReceivedEnum;
    conversation       :APPC.ConversationHandle;
    rc                 :APPC.ReturnCode;
    dbRec              :DBRecord;
```

```
BEGIN

 (* Allocate conversation with following:                    *)
 (* -- set synch level to none                              *)
 (* -- set system name to nul to use default name           *)
 (* -- Name of program is RTRREQ in Library JDE             *)
 (* -- no PIP data is sent                                  *)
   rc := APPC.Allocate(
                       bufferSize,
                       APPC.syncLevelNone,
                       nul,
                       'RTRREQ.JDE',
                       0,
                       nul,
                       conversation);

(* if any errors write message indicating problem           *)

   IF rc <> APPC.Ok THEN
     WriteReturnCode(rc);
     SetReturnCode(255);
     HALT ;
   END;
(* Ask for part number *)
  IO.WrStr('Enter Part Number or 99999 To End: ');
  IO.RdStr(inq.partNum);
  IO.WrLn;
  WHILE inq.partNum <> end DO;
  APPC.ConvertToEBCDIC(inq.partNum);
(* Send part number - module appc will handle ll and GDSID *)
   rc := APPC.SendData(
                       conversation,
                       5,
                       inq,
                       requestToSend
                     );
   IF rc <> APPC.Ok THEN
     WriteReturnCode(rc);
     SetReturnCode(255);
     HALT ;
   END;
(* Receive response *)
   rc := APPC.ReceiveAndWait(
                       conversation,
                       SIZE(dbRec),
                       dbRec,
                       whatReceived,
                       requestToSend,
                       receivedDataLength
                       );
   IF rc <> APPC.Ok THEN
     WriteReturnCode(rc);
     SetReturnCode(255);
     HALT ;
   END;
```

```
 (* receive and Wait to get send indication                    *)

 (* -- set max receive length to 0 since nothing is expected *)
  rc := APPC.ReceiveAndWait(
                       conversation,
                       0,
                       dbRec,
                       whatReceived,
                       requestToSend,
                       receivedDataLength
                       );
    IF rc <> APPC.Ok THEN
      WriteReturnCode(rc);
      SetReturnCode(255);
      HALT ;
    END;
    IF whatReceived <> APPC.receivedSend THEN
      IO.WrStr('Send indicator not received from host ');
      HALT;
    END;
 (* Write received record to screen *)
  APPC.ConvertToASCII(dbRec.partds);
  APPC.ConvertToASCII(dbRec.partq);
  APPC.ConvertToASCII(dbRec.errMsg);
  IO.WrStr('Description : ');
  IO.WrStr(dbRec.partds);
  IO.WrLn;
  IO.WrStr('Quantity : ');
  IO.WrStr(dbRec.partq);
  IO.WrLn;
  IO.WrStr(dbRec.errMsg);
  IO.WrLn;

  (* Ask for next part number *)
    IO.WrStr('Enter Part Number or 99999 To End: ');
    IO.RdStr(inq.partNum);
    IO.WrLn;
  END;
  (* Deallocate conversation when done *)
    rc := APPC.Deallocate(
                       conversation,
                       APPC.deallocateSynchLevel);

  END RtrReq.
```

APPC Subroutine Definition

The definition module describes the procedure interfaces, return codes, and parameter values for the APPC module. The implementation module is in the PC Support tools folder.

```
DEFINITION MODULE APPC;


(* This module provides an interface for the AS/400 router API.   *)
(* All functions provided by the router can be accessed through   *)
(* procedures in this module.                                     *)
(*                                                                *)
(* An enumeration called ReturnCodes has been defined to provide  *)
(* feedback on your request.  These return codes are easier to    *)
(* deal with than the two part APPC return codes.                 *)
(* In all cases where a string parameter is provided by your      *)
(* application, these routines will translate it to EBCDIC if     *)
(* necessary.  This means you only need to work with ASCII        *)
(* strings except for the data you are sending to your partner    *)
(* program.  It is your responsibility to format the data         *)
(* correctly.                                                     *)

FROM SYSTEM IMPORT ADDRESS,BYTE;
(*EXPORT
(*  Types                                  *)
                Location,
                LocationList,
                ReturnCode,
                TpNameString,
                SyncLevelEnum,
                DeallocateEnum,
                WhatReceivedEnum,
                FillEnum,
                ModeName,
                NetName,
                LuName,
                QualName,
                UserID,
                ConversationHandle,
                LastOperation,
```

```
(*  Procedures                          *)
                Allocate,
                Confirm,
                Confirmed,
                Deallocate,
                GetAttributes,
                PrepareToReceive,
                ReceiveImmediate,
                ReceiveAndWait,
                SendData,
                SendError,
                ConvertToASCII,
                ConvertToEBCDIC,
                GetLocationList,
                GetLastOp,
                Flip;   *)


(*     General constants.     *)
CONST

  MaxTpNameLength       = 64;    (* max of transaction pgm name          *)
  MaxLocationLength     = 8;     (* max length of location name          *)
  MaxLocations          = 32;

(*     General types.     *)

 TYPE TpNameString   = ARRAY [0..MaxTpNameLength-1] OF CHAR;
 TYPE Location       = ARRAY [0..MaxLocationLength-1] OF CHAR;

 TYPE LocationList = ARRAY [0..MaxLocations-1] OF Location;



 TYPE LuName   = ARRAY [0..7] OF CHAR;
 TYPE UserID   = ARRAY [0..9] OF CHAR;
 TYPE LastOperation = ARRAY[0..19] OF CHAR;

 TYPE ConversationHandle;
```

```
TYPE  ReturnCode =      (Ok,
                        DeallocateNormal,
                        ProgramErrorNoTruncation,
                        ProgramErrorTruncation,
                        ProgramErrorPurging,
                        ResourceFailureRetry,
                        ResourceFailureNoRetry,
                        Unsuccessful,
                        APPCBusy,
                        ParmCheckInvalidVerb,
                        ParmCheckInvalidConversationId,
                        ParmCheckBufferCrossesSegment,
                        ParmCheckTPNameLength,
                        ParmCheckInvalidConversationType,
                        ParmCheckBadSynchLevelOnAllocate,
                        ParmCheckBadReturnControl,
                        ParmCheckPipTooLong,
                        ParmCheckBadPartnerName,
                        ParmCheckConfirmNotAllowed,
                        ParmCheckBadDeallocateType,
                        ParmCheckPrepareToRcvType,
                        ParmCheckBadFillType,
                        ParmCheckReceiveMaxLengthTooBig,
                        StateCheckConfirmIssuedNotInConfirmState,
                        StateCheckRcvImmediateAndNotInReceive,
                        StateCheckRequestToSendInBadState,
                        StateCheckSendIssuedInBadState,
                        StateCheckSendErrorInBadState,
                        AllocateErrorNoRetry,
                        AllocateErrorRetry,
                        AllocateErrorPgmNotAvailableNoRetry,
                        AllocateErrorTpnNotRecognized,
                        AllocateErrorPgmNotAvailableRetry,
                        AllocateErrorSecurityNotValid,
                        AllocateErrorConversationTypeMisMatch,
                        AllocateErrorPipNotAllowed,
                        AllocateErrorPipNotCorrect,
                        AllocateErrorSynchLevelNotSupported,
                        DeallocateAbendProgram,
                        RouterNotInstalled
                        );


TYPE SyncLevelEnum     = (syncLevelNone,
                          syncLevelConfirm);

TYPE PipLengthRange = [0..128];

(*     Deallocate types.     *)

TYPE DeallocateEnum =   (deallocateSynchLevel,
                         deallocateFlush,
                         deallocateAbend);
```

```
TYPE WhatReceivedEnum = (data,
                         dataComplete,
                         dataIncomplete,
                         receivedConfirm,
                         receivedConfirmSend,
                         receivedConfirmDeallocate,
                         receivedSend);

TYPE FillEnum =         (buffer, ll);


(* Procedure Allocate is used to start a conversation with a partner.     *)
(* Buffer Length must be at least 271.                                    *)
(* LocationName is the name of the host to talk with. Set to nul if       *)
(* you want to use the default system.                                    *)
(* If no Pip data is to be sent, set pipLength to 0 and pipData to        *)
(* any variable.                                                          *)
(* The conversation Handle returned must be used on all subsequent        *)
(* requests for this conversation.                                        *)



 PROCEDURE Allocate(
       bufferLength    :CARDINAL;      (* Router will create buffer      *)
       synchLevel      :SyncLevelEnum;
       locationName    :ARRAY OF CHAR; (* Pass as ASCII string           *)
       tpn             :ARRAY OF CHAR; (* Pass as ASCII string, if       *)
                                       (* first char less than hex 40    *)
                                       (* not translation done           *)
       pipLength       :PipLengthRange;
       pipData         :ARRAY OF BYTE; (* no translation done, must      *)
                                       (* be in GDS format               *)

       VAR conversation:ConversationHandle)(* set by router,treat        *)
                                       (* like file handle               *)
       :ReturnCode;

(* Confirm is used to request confirmation from partner                   *)

 PROCEDURE Confirm(
       conversation           :ConversationHandle;
       VAR rqsToSendReceived  :BOOLEAN)
       :ReturnCode;


 PROCEDURE Confirmed(
       conversation    :ConversationHandle)
       :ReturnCode;

(* Deallocate type can be DeallocateFlush which is a                      *)
(* normal deallocate or DeallocateAbend which indicates an error          *)

 PROCEDURE Deallocate(
       conversation:ConversationHandle;
       type            :DeallocateEnum)
       :ReturnCode;
```

```
(* GetAttributes will return information about your conversation.        *)
(* All string values are returned as ASCII strings.                      *)

 PROCEDURE GetAttributes(
        conversation            :ConversationHandle;
        VAR syncLevel           :SyncLevelEnum;
        VAR ownLuName           :LuName;
        VAR partnerLuName       :LuName;
        VAR userID              :UserID)
        :ReturnCode;


 PROCEDURE PrepareToReceive(
        conversation:ConversationHandle)
        :ReturnCode;

 (* Receive Immediate will check to see if something has been            *)
 (* returned.  If not a return code- unsuccessful                        *)
 (* is returned.                                                         *)
 (* See ReceiveAndWait for parameter meanings.                           *)

 PROCEDURE ReceiveImmediate(
        conversation            :ConversationHandle;
        maxLength               :CARDINAL;
        VAR receiveData         :ARRAY OF BYTE;
        VAR whatReceived        :WhatReceivedEnum;
        VAR requestToSendRcvd   :BOOLEAN;
        VAR receiveDataLength   :CARDINAL)
        :ReturnCode;

 (* ReceiveAndWait is used to wait for information to arrive on          *)
 (* the conversation and then receives the information.                  *)
 (* MaxLength is set to the largest amount of data your program          *)
 (* will accept.  Note that it should be less than or equal to the       *)
 (* variable passed as received Data.                                    *)
 (* The whatReceived variable will tell you whether Data, Confirm        *)
 (* Send Indication, and so on has been received.                        *)
 (* The maxLength field must be in Intel format.  The received           *)
 (* data length field will be set in Intel format.                       *)

 PROCEDURE ReceiveAndWait(
        conversation            :ConversationHandle;
        maxLength               :CARDINAL;
        VAR receiveData         :ARRAY OF BYTE;
        VAR whatReceived        :WhatReceivedEnum;
        VAR requestToSendRcvd   :BOOLEAN;
        VAR receiveDataLength   :CARDINAL)
        :ReturnCode;
```

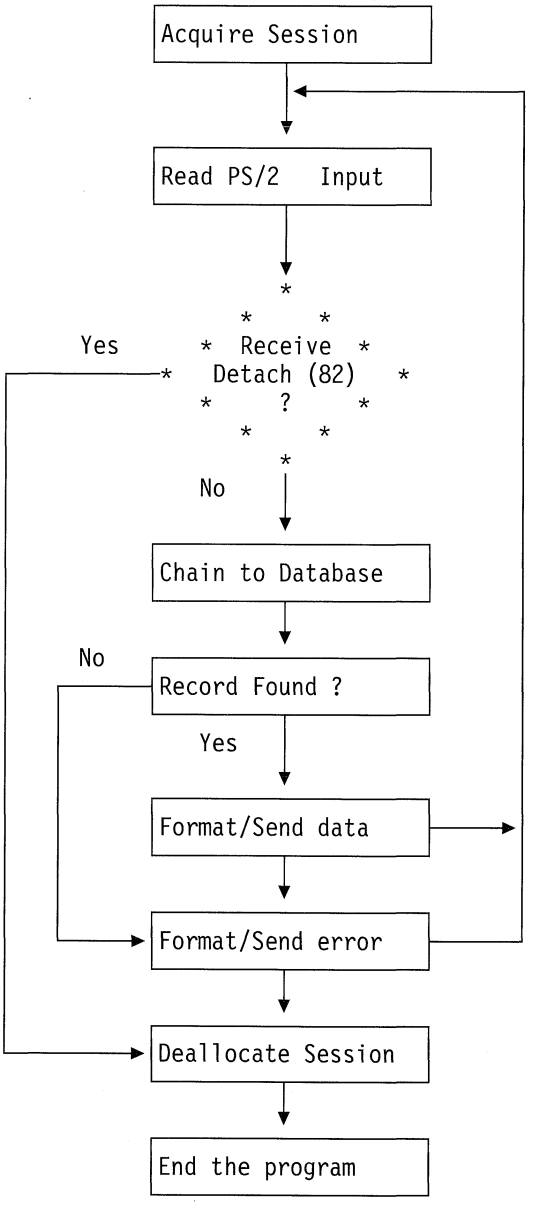
```
(* Sends data to your partner program.                                    *)
(* The data format consists of logical records as follows:                *)
(*                                                                        *)
(*       |-------|-------|-----------------------------|                  *)
(*       |       |       |                             |                  *)
(*       | LL    | GDSID |  User Data                  |                  *)
(*       |       |       |                             |                  *)
(*       |-------|-------|-----------------------------|                  *)
(*       0       2       4                                                *)
(*  The LL field is two bytes long and contains the total length          *)
(*  of the record, including its own 2 bytes, and the GDSID               *)
(*  field.  This field MUST NOT be in Intel format.                       *)
(*  The GDSID field contains an identifier of the record type.            *)
(*  You can use any values you want, because this is not checked by APPC.*)
(*  The data field contains your data in the format you want.             *)
(*  The router does not translate this field.                             *)


PROCEDURE SendData(
       conversation            :ConversationHandle;
       sendDataLength          :CARDINAL;
       VAR sendData            :ARRAY OF BYTE;
       VAR requestToSendRcvd   :BOOLEAN)
       :ReturnCode;



PROCEDURE SendError(
       conversation            :ConversationHandle;
       VAR requestToSendRcvd   :BOOLEAN)
       :ReturnCode;


PROCEDURE ConvertToASCII(
       VAR string      :ARRAY OF BYTE
       );

PROCEDURE ConvertToEBCDIC(
       VAR string      :ARRAY OF BYTE
       );

PROCEDURE GetLocationList(
       VAR locationList:LocationList
       );

(* Switch high low format of one word variable.  If in Intel format,     *)
(* this routine makes it host format and vice-versa.                     *)

 PROCEDURE Flip(VAR value:ARRAY OF BYTE);

(* Get last op is used by the WriteRc routine to retrieve the last       *)
(* operation tried.                                                      *)

 PROCEDURE GetLastOp(VAR lastOp:LastOperation);

END APPC.
```

Macro Assembler Program Example

A macro assembler program example may be found in the PC Support tools folder. It is a RTRREQ program written in assembler language. This program was assembled using the IBM Macro Assembler version 2.00.

AS/400 Program Example

The RTRREQ program provides the AS/400 support for the PS/2 application program. This section describes the flow of the program and provides RPG code to demonstrate program logic.



Note: The source code for the sample RPG program and the supporting file is in the PC Support tools folder. After downloading the package file, run the RTRSAMP.BAT program. The program will upload all the necessary files and create the RPG program. On the personal computer, compile and link

the version of the RTRREQ program you wish to use (C, Modula-2, or Macro Assembler), and then type RTRREQ.

The next two sections describe the RPG III program, RTRREQ, that runs on the AS/400 system. This RPG program uses the following files:

DBFIL The database file that contains the part numbers and associated information.

RTRAPI The OS/400 ICF file used to receive part numbers and send responses to the PS/2 program.

DDS Source for DBFIL Physical File

DBFIL is the AS/400 database that contains the part number records. The data description specifications (DDS) for the DBFIL file is shown below:

```
A                R DBRCD
A                  ITEMNM          5
A                  ITEMD          25
A                  ITEMQ           5
A                K ITEMNM
```

RTRAPI ICF File

An intersystem communication function (ICF) file is used to define the layout of the data sent and received by the AS/400 program. The AS/400 application program opens the ICF file and all communications functions are issued through the ICF file. The ICF file used for the AS/400 program is shown below:

```
A
A                                        INDARA
A                                        RCVFAIL(66)
A          R ITEMRQ
A                                        RCVDETACH(82)
A
A            PARTNM          5A
A
A          R ITEMDS
A            PARTDS         25A
A            PARTQ           5A
A            ERRORD         40A
```

AS/400 ICF File Creation and Program Device Entry Definition

The application program on the AS/400 system communicates through an ICF file. An ICF file must be created and have a device entry added to it. See the *ICF Programmer's Guide*, for details on creating ICF files.

The command used to create the OS/400 ICF file is:

```
CRTICFF
   File(RWM/RTRAPI)
   Srcfile(RWM/APPCSRCC)
   Srcmbr(RTRAPI)
   Waitfile(60)
   Share(*YES)
```

The command used to define the program device entry is:

```
ADDICFDEVE
   File(RWM/RTRAPI)
   PGMDEV(RQSDEV)
   RMTLOCNAME(*REQUESTER)
   CMNTYPE(*APPC)
```

RTRREQ RPG III Program Source

The following descriptions cross-reference the structure of the program example illustrated on page 4-102.

1 RTRAPI is the ICF file used to receive records from and send records to the PS/2 system. DBFIL is the database file that contains the part number information.

The continuation lines for the RTRAPI file specification define the following:

KNUM Specifies the maximum number of devices to be acquired.

KINFDS Specifies that the data structure to be used for error handling is named FEEDBK.

KINFSR Specifies that subroutine (EXCPTH) is called when an exception condition occurs.

2 The program first establishes a session using program device RQSDEV in ICF file RTRAPI. This is the program device that is associated with a remote location name of *REQUESTER. A read operation is issued to the program device to receive the request. If the read is successful, control passes to the READR subroutine.

3 This routine ends the job. It is called from the error routine if an error has occurred. It is also run when a detach (interrupt 82) is received from the PS/2 system.

4 The READR subroutine is called to process a request from the PS/2 program. It is conditioned to continue to run until a detach (interrupt 82) is received. If the part number is found in the database, the quantity and description is returned; otherwise, an error message is returned.

5 The SNDREC subroutine is called when the part number is found in the database.

6 The RECNF subroutine is called when the part number is not found in the database.

7 The EXCPTH subroutine is called when an error condition occurs.

Following is the RPG III program source code for the RTRREQ program:

```
1 H
  FDBFIL   IF  E           K        DISK
  FRTRAPI  CF  E                    WORKSTN
  F                                            KNUM        1
  F                                            KINFDS FEEDBK
  F                                            KINFSR EXCPTH
  E                    MSG     1   2 40
  IFEEDBK      DS
  I                                         38  45 FMTNM
  I                                        401 404 MAJMIN
  I                                        401 402 MAJCOD
  I                                        403 404 MINCOD
  C*This program is not copyright protected and may be used in part
  C*or in its entirety without permission from the author.
  C*This program is supplied "as is" without warranty of any kind.
  C*IBM assumes no responsibility for any errors that may be present.
2 C* START OF PROGRAM                                                  *
  C           'RQSDEV'  ACQ  RTRAPI
  C                     READ ITEMRQ                   88
  C                     EXSR READR
3 C* END OF JOB                                                        *
  C           'RQSDEV'  REL  RTRAPI
  C                     CLOSE*ALL
  C           ENDRUN    TAG
  C                     SETON                         LR
  C                     RETRN
  C* PROCESS INPUT                                                     *
4 C           READR     BEGSR
  C           *IN82     DOWEQ'0'
  C                     MOVE PARTNM    ITEMNM
  C           ITEMNM    CHAINDBRCD               98
  C   98                EXSR RECNF
  C  N98                EXSR SNDREC
  C                     READ ITEMRQ                   88
  C                     END
  C                     ENDSR
5 C           SNDREC    BEGSR
  C                     MOVE ITEMD     PARTDS
  C                     MOVE ITEMQ     PARTQ
  C                     MOVE *BLANK    ERRORD
  C                     WRITEITEMDS
  C                     ENDSR
6 C           RECNF     BEGSR
  C                     MOVE *BLANK    PARTDS
  C                     MOVE *BLANK    PARTQ
  C                     MOVE MSG,1     ERRORD
  C                     WRITEITEMDS
  C                     ENDSR

7 C* ICF FILE ERROR HANDLER                                           *
  C           EXCPTH    BEGSR
  C           MAJMIN    IFNE '0000'
  C                     GOTO ENDRUN
  C                     END
  C                     ENDSR
**
THE NUMBER WAS NOT FOUND
```

Chapter 5. Transfer Function

Low-Level Application Program Interface

This chapter describes the PC Support transfer function's application program interface (API) when using the DOS or OS/2 operating system. The information applies when using either operating system, unless otherwise noted.

Transfer Function Overview

The transfer function transfers data between the AS/400 system and a personal computer. PC Support provides interactive, automatic, and API programs for the transfer functions. This support gives you flexibility in sharing AS/400 system data with your personal computer.

The transfer function consists of the following:

- AS/400 transfer function
- Personal computer transfer function, which consists of two parts:
  - Interactive or automatic transfer function programs
  - API program

The AS/400 transfer function program runs on the AS/400 system. The personal computer transfer function program (the API and interactive or automatic transfer function programs) runs on the personal computer.

The interactive or automatic transfer function programs pass requests to the AS/400 transfer function program through the appropriate router and communications support, depending on whether you use the DOS or OS/2 operating system.

The AS/400 transfer function programs use the transfer request to send data or to retrieve data from the specified AS/400 database files.

Figure 5-1 on page 5-2 shows how the interactive and automatic programs relate to each other when you use the DOS operating system.

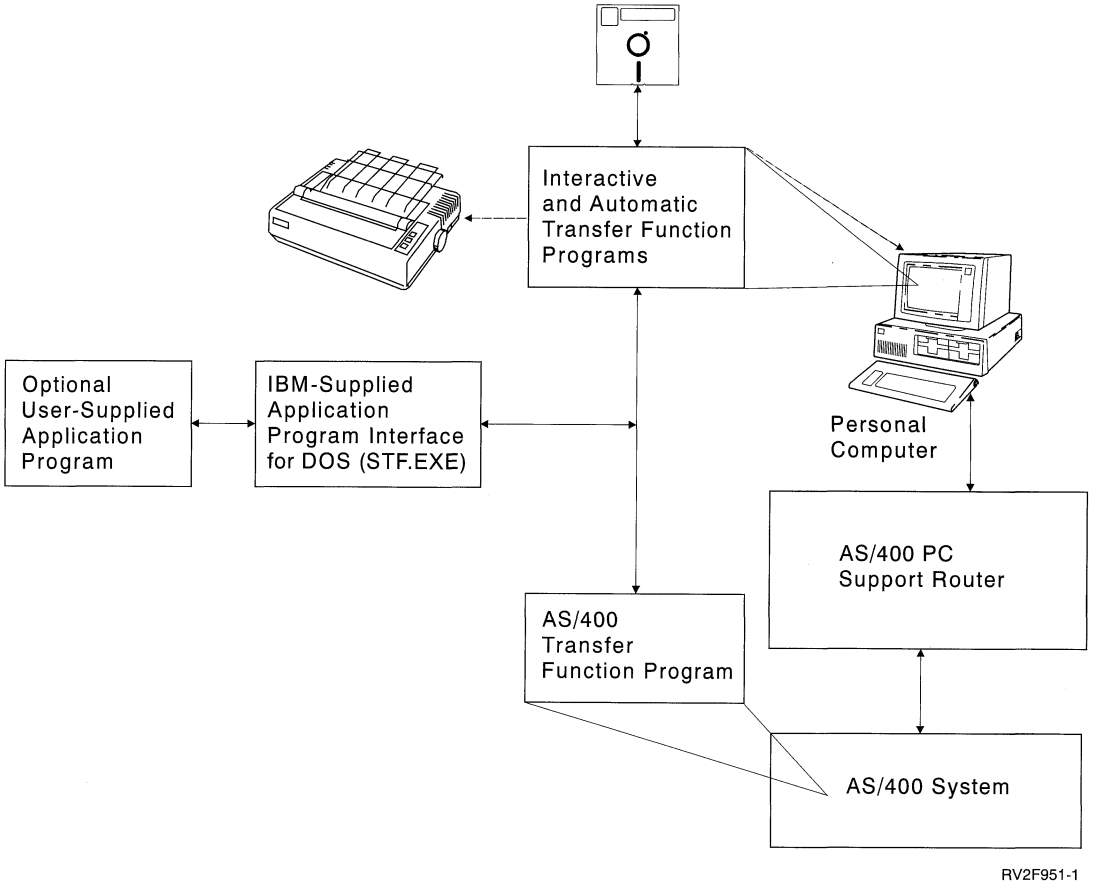


Figure 5-1. Transfer Function Overview Using the DOS Operating System

Figure 5-2 on page 5-3 shows how the interactive and automatic programs relate to each other when you use the OS/2 operating system.

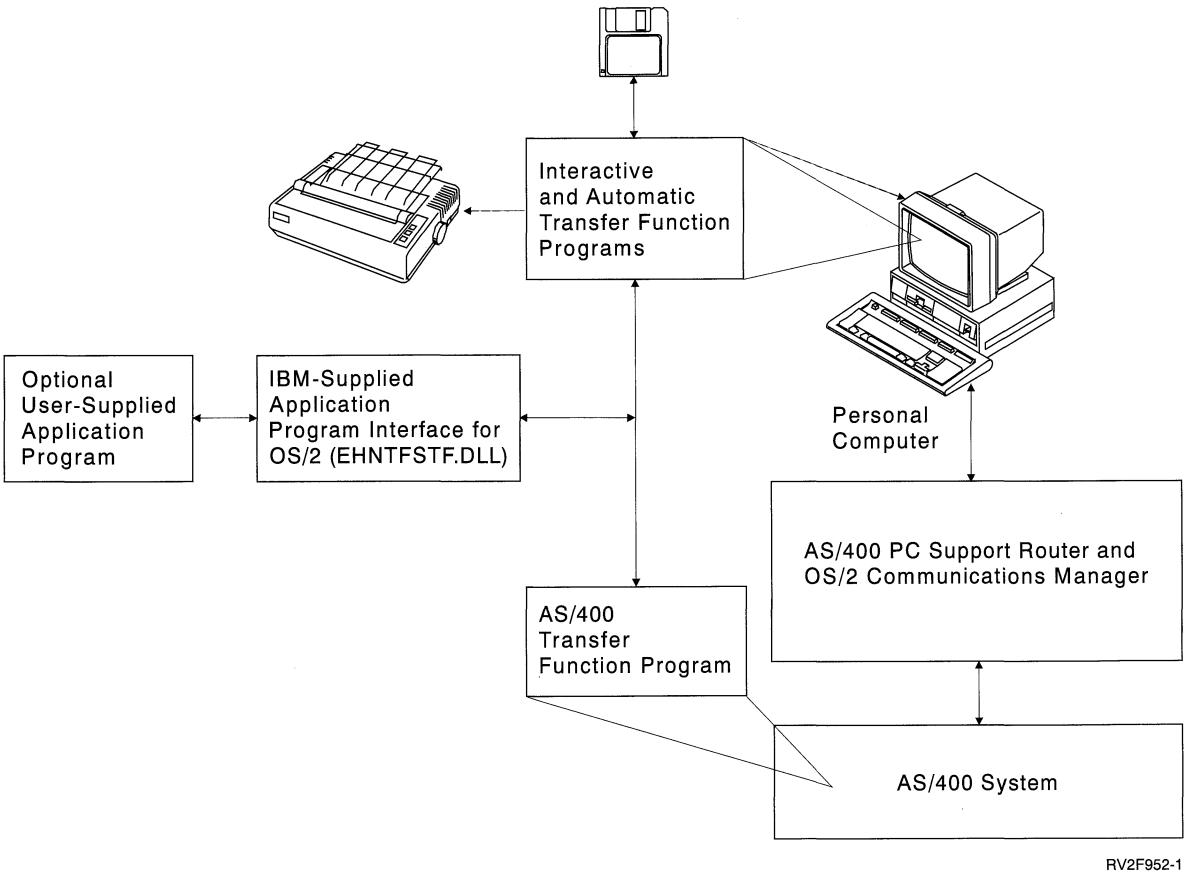


Figure 5-2. Transfer Function Overview Using the OS/2 Operating System

For an AS/400 system-to-personal computer transfer request, the interactive or automatic transfer function program retrieves the data, one record at a time, and changes the system data records to the format needed by the personal computer. The interactive or automatic transfer function program then does one of the following:

- Displays the data on the personal computer display
- Prints the data on the personal computer printer
- Writes the data to a personal computer disk or diskette file

For a personal computer-to-AS/400 transfer request, the interactive or automatic transfer function program sends data records, one at a time, to the AS/400 transfer function program. The AS/400 transfer function program writes the data to a physical database file. For more information on the AS/400 transfer function and the interactive or automatic transfer function programs, see the *PC Support/400: DOS and OS/2 Technical Reference*, SC41-8091.

An API is supplied which allows you to write personal computer application programs to create transfer requests instead of using the interactive or automatic transfer function programs. This allows the personal computer application programs to send data to or receive data from an AS/400 database file.

Transferring Files from the AS/400 System to the Personal Computer

When using a personal computer, you can retrieve and use data from the following file types on the AS/400 system:

- Physical database
- Logical database
- DDM

When retrieving files, you can:

- Control which records (and which fields within a record) are retrieved.
- Control the ordering of records and the ordering of fields within the record.
- Control the format of the date and time data and the separator used.
- Group records into summary records.
- Join two or more files.

Transferring Files from the Personal Computer to the AS/400 System

You can transfer data from a file on a personal computer to an AS/400 database file member. When transferring data to the AS/400 system, you can optionally create the file or the file member, or both. Personal computer data transferred to the AS/400 file member replaces data in the file member.

Note: Because existing data in a file member is replaced when you send data to that file member, it might be a safer practice to send your data to a work file on the AS/400 system and then have an AS/400 application merge the contents of this work file into the primary file.

For the AS/400 system, field definitions are contained in the file or in a data dictionary.

Database physical files are transferred by individual field. That is, numeric and character data fields are translated from personal computer format to AS/400 format as part of the sending process. The translation is specific to each field.

If the AS/400 file has fields not in the personal computer file you are sending, default values are used for the fields. If a field of the AS/400 file has a default value specified through the DFT DDS keyword, the default value is used. If no default value is specified, the field default values are as follows:

Field Type	Default Value
Character	EBCDIC blanks
Numeric	Zeros
Date	Current AS/400 system date
Time	Current AS/400 system time
Timestamp	Current AS/400 system timestamp

You can also transfer data to an AS/400 source file. A record length is required to transfer data to an AS/400 source file. A source sequence number and date are stored with each record of the data.

Transfer Requests

Your application program can perform these types of transfer requests with the AS/400 system:

- SELECT
- EXTRACT
- REPLACE
- OPTIONS

SELECT Transfer Requests

A SELECT transfer request transfers data from the system database file to a personal computer.

For example:

```
SELECT NAME,ADDRESS,SALARY
FROM PAYROLL WHERE DEPT='200'
```

retrieves records from the system file named PAYROLL. The records contain the name, address, and salary information for everyone in department 200.

The transfer request consists of a series of parameters identified by a keyword. Table 5-1 shows the keywords that correspond to prompts displayed by the interactive AS/400 system-to-personal computer transfer function (RTOPC). Note that in Table 5-1 the keywords are the same as their corresponding prompts for RTOPC.

Table 5-1. Select Function Keywords and Prompts

Keyword	RTOPC Prompt
SELECT	SELECT
FROM	FROM
WHERE	WHERE
GROUP BY	GROUP BY
HAVING	HAVING
ORDER BY	ORDER BY

Figure 5-3 on page 5-6 shows the SELECT transfer request syntax.

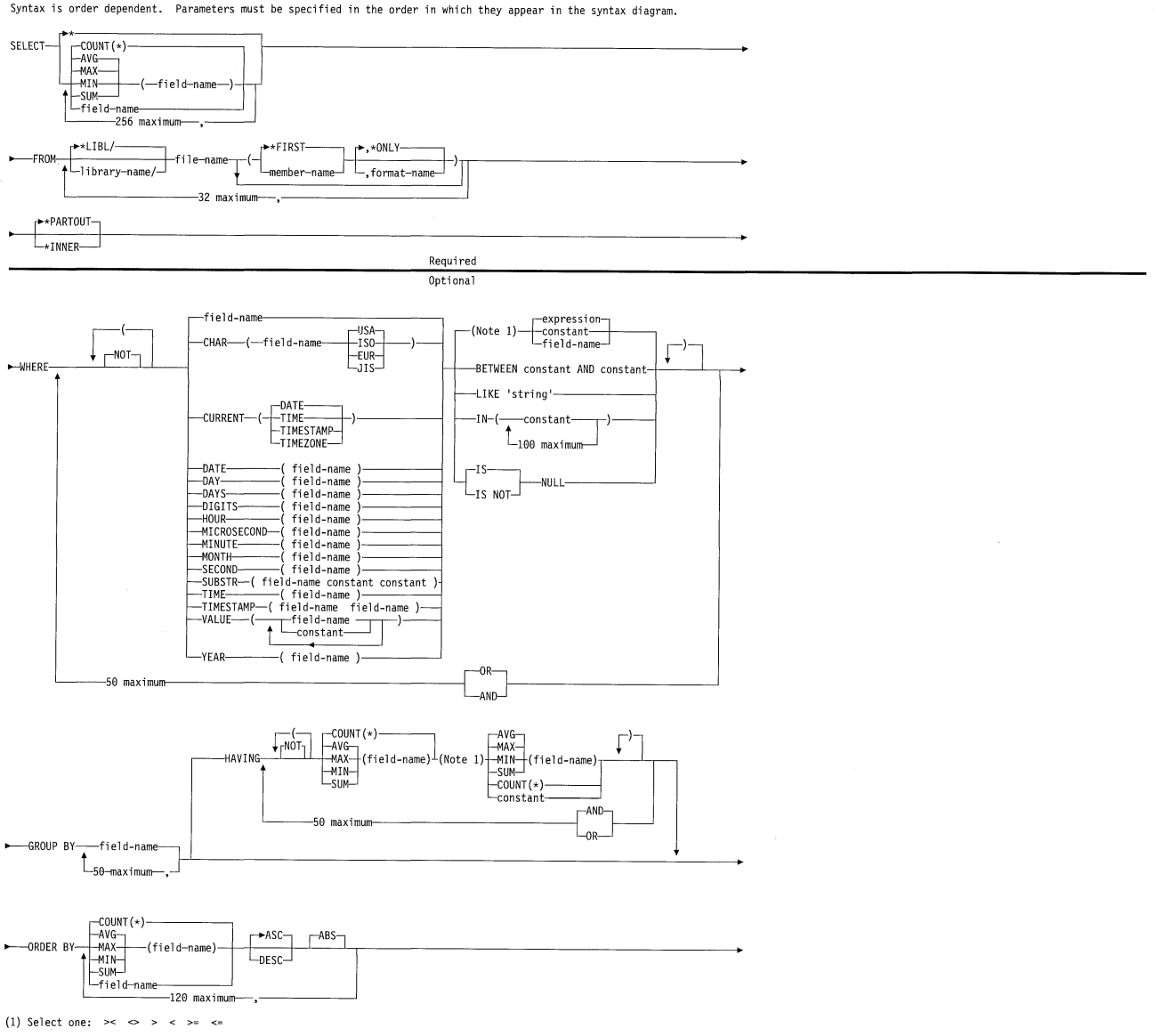


Figure 5-3. SELECT Transfer Request Syntax

SELECT Transfer Request Considerations

Consider the following when creating a SELECT transfer request:

- The *PARTOUT/*INNER parameter is an optional part of the FROM clause. Specify this parameter only if more than one file is specified in the FROM clause. This parameter specifies whether records with missing fields will transfer, and corresponds with the *Return records with missing fields* prompt that the interactive transfer function displays.

  *PARTOUT specifies that records with missing fields should transfer. It is also the default.

  *INNER specifies that records with missing fields should not transfer.

- The WHERE clause serves two purposes:
  - To identify JOIN conditions. These are conditions for joining records from more than one file.

- To identify WHERE conditions. These are conditions that records must meet in order to be transferred or included in that group.

JOIN conditions are required if more than one file is specified in the FROM clause. The first portion of the WHERE clause identifies the JOIN conditions. JOIN conditions must immediately follow the WHERE keyword and must be separated from WHERE conditions by AND. WHERE conditions must be enclosed in parentheses when JOIN conditions are specified.

- When joining records from more than one file, a field name has the following format:

  [Tx.]field-name

  Tx. is an optional file qualifier that identifies which file in the FROM clause contains the specified field name. In this file qualifier, **x** is a number in the range 1 through 32. You must specify a file qualifier if the field-name is found in more than one file.

  Following is an example of a transfer request that joins records from more than one file:

```
SELECT T1.CPRTNO, DESCRIPTION, PRICE
FROM SUPPLIERS,INVENTORY
WHERE T1.CPRTNO = T2.CPRTNO AND (SUPPNO = 51)
ORDER BY T1.CPRTNO
```

  Note the following about the previous example:

  - More than one file is specified in the FROM clause.
  - One JOIN condition T1.CPRTNO = T2.CPRTNO is specified in the WHERE clause.
  - One WHERE condition (SUPPNO = 51) is specified in the WHERE clause. The WHERE condition is enclosed in parentheses.
  - The JOIN condition is separated from the WHERE condition by AND.

For more information, refer to the *PC Support/400 DOS Installation and Administration Guide* and to the *PC Support/400 OS/2 Installation and Administration Guide*.

EXTRACT Transfer Requests

Use EXTRACT transfer requests to get a list of library, file, member, format, or field names.

There are two formats for the EXTRACT transfer request:

- EXTRACT TABLES
- EXTRACT COLUMNS

EXTRACT TABLES: This format retrieves a list of library, file, member, or format names. There are two forms of the EXTRACT TABLES transfer request:

- The following form retrieves information about physical *and* logical files:

  EXTRACT TABLES {x}

  Note: This form of the EXTRACT request should be used when retrieving information for a SELECT transfer request because you can retrieve information from both physical files and logical files.

- The following form retrieves information about physical files only.

  EXTRACT TABLES(PHY) {x}

  Note: This form of the EXTRACT request should be used when retrieving information for a REPLACE transfer request because you can send information only to physical files.

- The following form retrieves information about the user's authorization to the files in addition to the information retrieved about the physical and logical files.

  EXTRACT TABLES (AUT) {x}

- The following form retrieves information about the user's authorization to the files in addition to the information retrieved about the physical files.

  EXTRACT TABLES (PHY) (AUT) {x}

  Note: This form of the EXTRACT request should be used when retrieving information for physical files only.

For either form of the EXTRACT TABLES transfer request, **x** is an optional field that can be one of the following:

library-name/
This form returns an alphabetical list of files that you have authority to in that library. For example:

EXTRACT TABLES EMPLIB/

returns a list of physical and logical files in library EMPLIB.

library-name/filename, or **filename**
This form returns an alphabetical list of members in the specified file. For example:

EXTRACT TABLES EMPLIB/DEPT

returns a list of members for file DEPT in library EMPLIB.

library-name/filename(member-name), or **filename(member-name)**
This form returns an alphabetical list of record formats associated with the member. For example:

EXTRACT TABLES EMPLIB/DEPT(SALARY)

returns a list of record formats associated with member SALARY in file DEPT in library EMPLIB.

library-name/filename(member-name,format-name), or **filename(member-name,format-name)**
This form returns a list of all the libraries defined in the user portion (*USRLIBL) of the AS/400 job's library list. You can personalize this library list returned from the AS/400 system by changing your Job Description (CHGJOBD command) on the AS/400 system. The transfer function assumes that, because you specified a complete file name, you want to select another file name.

Note: By not specifying **x** (by leaving it blank), you also get this library list.

The maximum length for any name is 10 characters.

The following special values are supported for library-name, member-name, and format-name:

- Specify *USRLIBL to indicate that the user portion of the AS/400 job's library list is used to search for the file.
- Specify *LIBL as the library name to indicate that the user portion and the system portion of the AS/400 job's library list is used to search for the file.
- Specify *FIRST as the member name to indicate use of the first member in the file.
- Specify *ONLY as the record format name to indicate that the only record format in the specified file is used. *ONLY is valid only if the file has a single record format. For database files with more than one record format, a record format name is required.

For file, member, and record format names, you can specify a partial name followed by an asterisk (*). For example, if you specify ARLIB/AR*, you receive a list of all the file names beginning with the characters AR contained in the library ARLIB.

Note: The user authority information is returned only if a list of files is returned. The (AUT) parameter is ignored if a list of libraries, members, or record formats is returned.

Figure 5-4 shows the EXTRACT TABLES transfer request syntax.

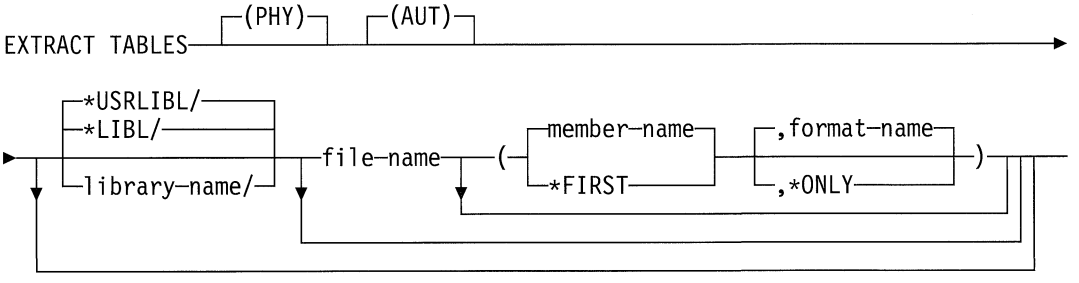


Figure 5-4. EXTRACT TABLES Transfer Request Syntax

EXTRACT COLUMNS: This format retrieves a list of the field definitions for the file name and the specified format name. The format of the EXTRACT COLUMNS transfer request is:

EXTRACT COLUMNS y[,y] ...

y is one of the following:

- **library-name/filename,** or **filename**
- **library-name/filename(member-name),** or **filename(member-name)**
- **library-name/filename(member-name,format-name),** or

  filename(member-name,format-name)

Specify up to 32 AS/400 file names using EXTRACT COLUMNS. When you specify more than one file, field names for the first file are returned, followed by field names for the second, and so on. To identify which fields belong to which

files, each field name is preceded by a file qualifier (the character T followed by a number and a period).

For example:

EXTRACT COLUMNS EMPLIB/DEPT

returns a list of field descriptions for the only record format associated with the first member in file DEPT in library EMPLIB.

But the following:

EXTRACT COLUMNS COMPANYLIB/EMPFILE, PAYLIB/PAYROLL(SHIPPING)

returns a list of field descriptions for:

- The only record format associated with the first member in file EMPFILE in library COMPANYLIB. File qualifier T1 identifies the field names.
- The only record format associated with member SHIPPING in file PAYROLL in library PAYLIB. The field names are identified by file qualifier T2.

Notes:

1. To list information about a file, you must have operational (*OPER) authority to the file.
2. The file will be allocated as shared-no-update (*SHRNUP) while the AS/400 system processes the EXTRACT transfer request.

Figure 5-5 shows the EXTRACT COLUMNS transfer request syntax.

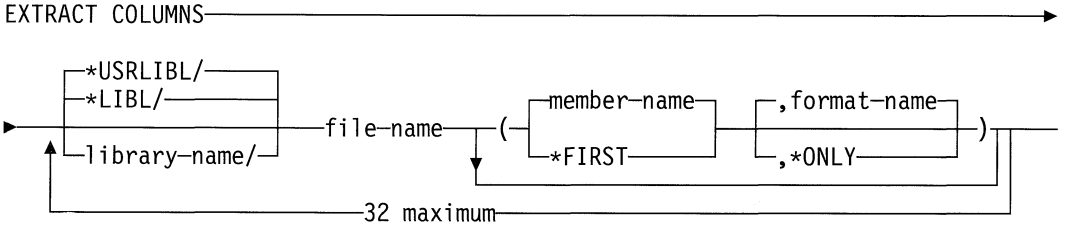


Figure 5-5. EXTRACT COLUMNS Transfer Request Syntax

EXTRACT Transfer Request Data Format: Your application program passes an EXTRACT transfer request to the IBM-supplied API program (STF.EXE, if you are using DOS and EHNTFSTF.DLL, if you are using the OS/2 operating system) in the same way that a SELECT transfer request is passed. Refer to "Transfer Function API for DOS Users" on page 5-18 or "Transfer Function API for OS/2 Users" on page 5-43 for information on the transfer function APIs.

The IBM-supplied API program returns templates and records to your application program for an EXTRACT transfer request in the same way it returns templates and records for a SELECT transfer request. For a SELECT transfer request, the templates describe the fields within the record and the records contain the actual data from the file. For an EXTRACT transfer request, the templates describe what the list looks like and each record contains one row of the list such as the description for a library, file, member, record-format, or field, depending on the type of list you requested on the EXTRACT transfer request.

Note: The templates sent on an EXTRACT transfer request can be thought of as the column headings for the list. These templates are used by the interactive transfer function programs to construct the column headings you see when you request a list with the PF4 key.

Refer to "Retrieve the Templates Function (AL=02)" on page 5-23 if you are using the DOS operating system, or to "Retrieve the Templates Function (Function=02)" on page 5-47 if you are using the OS/2 operating system for the format of the templates on an EXTRACT TABLES transfer request.

The following is the format of the records for an EXTRACT TABLES transfer request:

- If you do not specify **x**:

  Bytes 1 through 10 (character): Library-name

  Bytes 11 through 60 (character): Library-text

  The digits byte of the first template will have 2 bits set on:

  X'80' to indicate that this is the first part of the four-part AS/400 file name.

  X'40' to indicate that this name is the library name.

  These bits are used by the interactive transfer function to automatically add a comma before the library name and a slash after the library name when it is selected from the list.

- If you specify only a library name:

  Bytes 1 through 21 (character): Library-name/file-name

  Bytes 22 through 25 (character): File-type

  – For an EXTRACT TABLES (PHY) transfer request, file-type can have the following values:

    DATA Indicates that the file is a physical data file

    SRC Indicates that the file is a source physical file

  – For an EXTRACT TABLES transfer request, file-type can have the following values:

    PHY Indicates that the file is a physical file

    LGL Indicates that the file is a logical file

  For an EXTRACT TABLES(PHY) request, the file contains either DATA (physical data file) or SRC (source physical file). For an EXTRACT TABLES request, it contains either PHY (physical file) or LGL (logical file).

  Bytes 26 through 75 (character): File-text

  The following fields are returned only for an EXTRACT TABLES (AUT) or EXTRACT (PHY) (AUT) request. A capital letter Y in the field indicates the user has that level of authorization to the file and an N indicates the user does not have that level of authorization.

  Byte 76 (character): Object-operational authority

  Byte 77 (character): Object-management authority

  Byte 78 (character): Object-existence authority

Byte 79 (character): Data-read authority

Byte 80 (character): Data-add authority

Byte 81 (character): Data-update authority

Byte 82 (character): Data-delete authority

Byte 83 (character): Private authority

- If you specify a file name:

  Bytes 1 through 33 (character): Library-name/file-name(member-name)

  Bytes 34 through 83 (character): Member-text

- If you specify a file name and a member name:

  Bytes 1 through 44 (character): Library-name/file-name(member-name,format-name)

  Bytes 45 and 46 (binary): Record length in number of bytes

  Bytes 47 through 95 (character): Format-text

The following is the format of the records for an EXTRACT COLUMNS transfer request:

- If you specify one file:

  Bytes 1 through 10 (character): Field-name

  Bytes 11 through 15 (character): Data type of field. Contains one of the following:

CHAR	Character
BIN	Binary
PACK	Packed decimal
ZONED	Zoned numeric
FLOAT	Floating point (not supported by the transfer function)
DATE	Date data type
TIME	Time data type
TSTMP	Timestamp data type
VCHAR	Variable-length character data type
VHEX	Variable-length hexadecimal data type
OPEN	IGC OPEN data type (DBCS only)
ONLY	IGC ONLY data type (DBCS only)
ETHER	IGC EITHER data type (DBCS only)
VOPEN	Variable-length IGC OPEN data type (DBCS only)
VONLY	Variable-length IGC ONLY data type (DBCS only)
VETHR	Variable-length IGC EITHER data type (DBCS only)
GRAPH	Systems Application Architecture* (SAA*) IGC graphic data type (DBCS only)
VGRPH	Variable-length SAA IGC graphic data type (DBCS only)
(spaces)	Unsupported data type

  Bytes 16 and 17 (binary): Length of field in number of bytes.

  Bytes 18 and 19 (binary): Total number of digits to the left and to the right of the decimal point. If the field type is character, this field is 0.

  Bytes 20 and 21 (binary): Number of digits to the right of the decimal point. If the field type is character, this field is 0.

Bytes 22 through 71 (character): Field-text.

Bytes 72 through 74 (character): Null-capable. If the returned value is "YES," the field is null-capable. If the returned value is "NO ," the field is not null-capable.

- If you specify more than one file:

  Bytes 1 through 14 (character): Qualified field-name.

  Bytes 15 through 19 (character): Data type of field. Contains one of the following:

CHAR	Character
BIN	Binary
PACK	Packed decimal
ZONED	Zoned numeric
FLOAT	Floating point (not supported by the transfer function)
DATE	Date data type
TIME	Time data type
TSTMP	Timestamp data type
VCHAR	Variable-length character data type
VHEX	Variable-length hexadecimal data type
OPEN	IGC OPEN data type (DBCS only)
ONLY	ICG ONLY data type (DBCS only)
EITHER	IGC EITHER data type (DBCS only)
VOPEN	Variable-length IGC OPEN data type (DBCS only)
VONLY	Variable-length IGC ONLY data type (DBCS only)
VETHR	Variable-length IGC EITHER data type (DBCS only)
GRAPH	SAA IGC graphic data type (DBCS only)
VGRPH	Variable-length SAA IGC graphic data type (DBCS only)
(spaces)	Unsupported data type

  Bytes 20 and 21 (binary): Length of field in number of bytes.

  Bytes 22 and 23 (binary): Total number of digits to the left and to the right of the decimal point. If the field type is character, this field is 0.

  Bytes 24 and 25 (binary): Number of digits to the right of the decimal point. If the field type is character, this field is 0.

  Bytes 26 through 75 (character): Field-text.

  Bytes 76 through 78 (character): Null-capable. If the returned value is "YES," the field is null-capable. If the returned value is "NO ," the field is not null-capable.

REPLACE Transfer Requests

A REPLACE transfer request transfers data from the personal computer to the AS/400 system.

You can transfer data to an existing AS/400 member. In this case, the data from the personal computer replaces any data already in the AS/400 member.

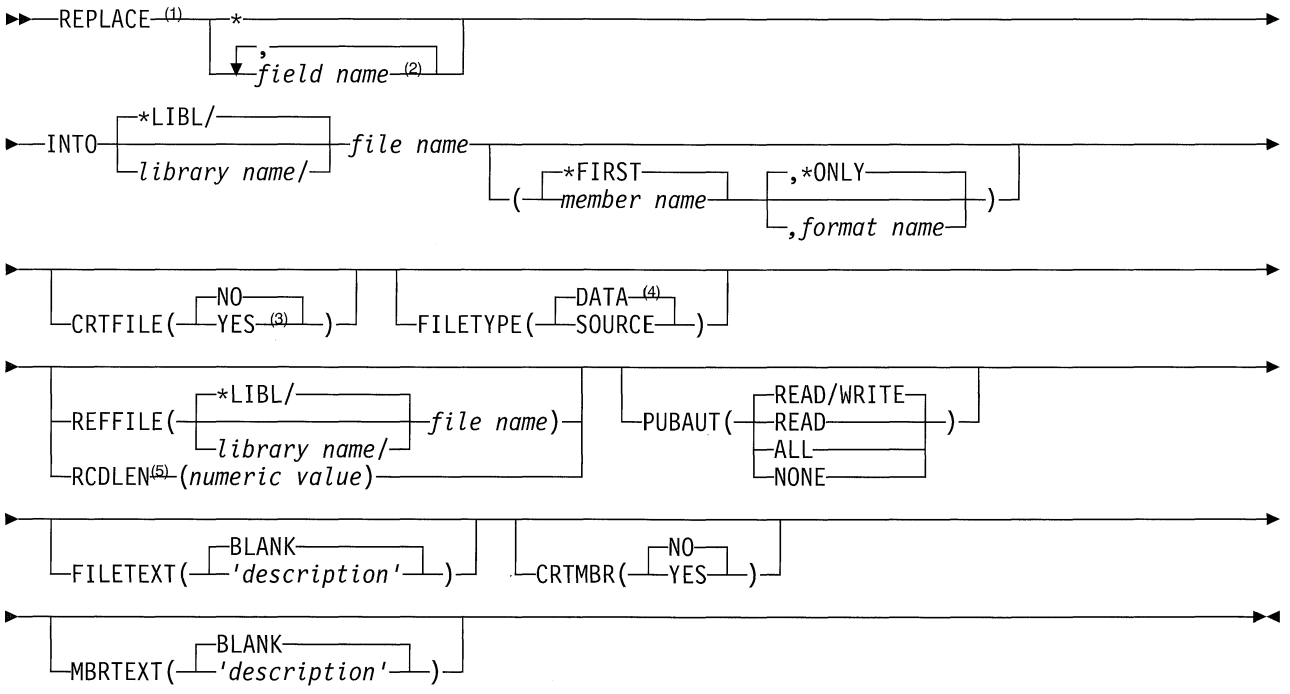
You can also transfer data to a new AS/400 member, or to a new member in a new file. In other words, you can use the REPLACE transfer request to create new AS/400 files and members.

The transfer request consists of a series of parameters identified by a keyword. Table 5-2 on page 5-14 shows the keywords that correspond to prompts displayed by the interactive PC-to-AS/400 transfer function.

Table 5-2. Transfer Function Keywords and Prompts

Keyword	RFROMPC Prompt
INTO	To
CRTFILE	Create AS/400 objects
FILETYPE	AS/400 file type
REFFILE	Field reference filename
RCDLEN	Record length
PUBAUT	Authority
FILETEXT	File text
CRTMBR	Create AS/400 objects
MBRTEXT	Member text

Figure 5-6 shows the REPLACE transfer request syntax.



Notes:

1. Syntax is order-dependent. Parameters must be specified in the order in which they appear in the syntax diagram.
2. A maximum of 256 field names may be specified.
3. If CRTFILE(YES) is specified, the CRTMBR(YES) is also required.
4. If FILETYPE(DATA) is specified, then a list of field names is required for REPLACE.
5. If RCDLEN is specified, then * is required for REPLACE.

Figure 5-6. REPLACE Transfer Request Syntax

For more information on transfer function keywords and prompts, refer to the *PC Support/400 User's Guide for DOS* or the *PC Support/400 User's Guide for OS/2*.

The following examples show the variations of the REPLACE request:

- The following transfers data to an existing file member in an existing file:

```
REPLACE NAME,SALARY
  INTO CUSTLIB/ACCOUNTS(MONTH1)
```

  Records containing the fields NAME and SALARY are transferred to the AS/400 member MONTH1 in file ACCOUNTS in library CUSTLIB.

  If you do not specify the AS/400 member name, *FIRST is assumed and the data transfers to the first member in the specified file.

  If you are transferring records made up of one character field per record, the REPLACE clause can contain an asterisk (*) and no field names. In this case, the file must be a source physical file.

  If the AS/400 member in the INTO clause exists and contains data, the open request returns a 0400 return code. This warns you that, if you continue the transfer request by sending data, the original data in the AS/400 member is replaced with the data you are sending. You can send a close request if you want to save the data already in the member.

- The following transfers data to a new member in an existing file:

```
REPLACE NAME,SALARY
  INTO CUSTLIB/ACCOUNTS(MONTH2)
  CRTFILE(NO) CRTMBR(YES)
  MBRTEXT('Customer accounts for February')
```

  Records containing the fields NAME and SALARY are transferred to a new AS/400 member in file ACCOUNTS in library CUSTLIB. The member is named MONTH2.

  MBRTEXT is an optional parameter that allows you to describe the new member. You must enclose the descriptive text in single quotation marks. Do not exceed 50 bytes in length. Specify any single quotation marks required within the descriptive text as two single quotation marks ('').

  If you are transferring records to the AS/400 system that are made up of one character field per record, the REPLACE clause can contain an asterisk (*) and no field names. In this case, the member that is created is a physical source member.

- The following transfers data to a new member in a new file:

```
REPLACE NAME,SALARY
  INTO CUSTLIB/EMPFILE(MONTH1,FMT1)
  CRTFILE(YES) FILETYPE(DATA)
  REFFILE(CUSTLIB/FIELDREF) PUBAUT(READ/WRITE)
  FILETEXT('Employee accounts file')
  CRTMBR(YES)
  MBRTEXT('Employee accounts for January')
```

  Records containing the fields NAME and SALARY transfer to a new AS/400 member in a new file in library CUSTLIB. The member is name MONTH1. The file is name EMPFILE.

  The record format is name FMT1 and uses field descriptions contained in the record format of the field reference file FIELDREF in CUSTLIB. If you do not

specify a record format name for the file to be created, the default name of QDFTFMT is used. For this type of transfer, where fields are defined, and a new file and a new member are created, specify REFFILE.

EMPFILE has a public authority value of READ/WRITE.

- The following transfers data to a new file and a new member:

```
REPLACE * INTO CUSTLIB/EMPFILE(MONTH1,FMT1)
  CRTFILE(YES) FILETYPE(SOURCE)
  RCDLEN(256) PUBAUT(READ)
  FILETEXT('Employee accounts file')
  CRTMBR(YES)
  MBRTEXT('Employee accounts for January')
```

In this instance, the data transferred consists of one field per record. This transfer request creates a source physical file called EMPFILE. The name of the new member is MONTH1. The name of the record format is FMT1.

Note: This request transfers records that are composed of one character field to an AS/400 source physical file. A REPLACE value of an asterisk (*) and a FILETYPE of SOURCE is specified. This is the only type of transfer where RCDLEN is valid. All other transfers (those where fields are defined) must use the REFFILE parameter instead of RCDLEN.

OPTIONS Transfer Requests

To use the OPTIONS statement, use OPEN (AL=01) with the OPTIONS statement in the buffer and then check for errors. Subsequent OPEN calls with SELECT or REPLACE statements in the buffer use the options you specified.

Note: Any number of OPTIONS keywords may be used. However, each keyword may be specified only once per OPTIONS statement. If duplicate keywords are specified, the last request specified is used.

Use an OPTIONS transfer request to specify the following:

IGNDECERR
: Specifies whether to ignore decimal data errors in AS/400 packed decimal or zoned decimal fields during Select requests. The valid values are Y and N. The default is Y. If the default is overridden by specifying N, performance improvements can be obtained when numeric key indexes are used.

TIMFMT
: The format to use for data in an AS/400 TIME field. The valid values are USA, ISO, EUR, JIS, HMS, DDS (AS/400 data description specifications), and DFT (AS/400 default format).

TIMSEP
: The character used to separate the different parts of a time field. This value is only recognized if you use the time format HMS. The valid values are colon (:), period (.), comma (,), blank ( ), NUL (no separator), or DFT (defaults to the AS/400 default separator).

DATFMT
: The format to use for data in an AS/400 DATE field. The valid values are USA, ISO, EUR, JIS, MDY, DMY, YMD, JUL, DDS (AS/400 data description specifications), or DFT (AS/400 default format).

DATSEP

The character used to separate the different parts of a date field. This can only be specified if you use the date format MDY, YMD, DMY, or JUL. The valid values are slash (/), hyphen (-), period (.), comma (,), blank ( ), NUL (no separator), or DFT (AS/400 default separator).

SRTSEQ

The sort sequence that will be used if an ORDER-BY clause is specified. The sort sequence affects all character comparisons that depend on the order of the alphabet. These comparisons can occur in the WHERE, GROUP BY, HAVING, and JOIN BY clauses, the IN, LIKE, BETWEEN, <, >, and = predicates, and the MAX and MIN functions. The valid values are *JOB, *HEX, *LANGIDSHR, *LANDIDUNQ, or a user-specified AS/400 table name. The user-specified table name should be in the form LIBRARY/TABLE, where the library is *LIBL, *CURLIB, or the actual library name. If no library is specified, the current library list will be searched for a matching table name.

LANGID

The language ID used to select the sort sequence table when *LANGIDSHR or *LANGIDUNQ is specified for the SRTSEQ option. The valid values are *JOB and a user-specified language ID.

STYLE The type of data stream used to communicate between the transfer API program and the host AS/400 system. The valid values are OLD and NEW. The old style was used with Version 2 Release 1.1 and earlier of PC Support/400 and allows for a larger maximum record size of 4096 bytes. The new data stream is available with Version 2 Release 2.0 and later of PC Support/400. It supports date, time or timestamp, NULL-capable and variable length fields, but has a reduced maximum record size. The maximum record size is:

```
maximum bytes = 4096 - 2 - (the number of fields in the record)
```

The default value is NEW.

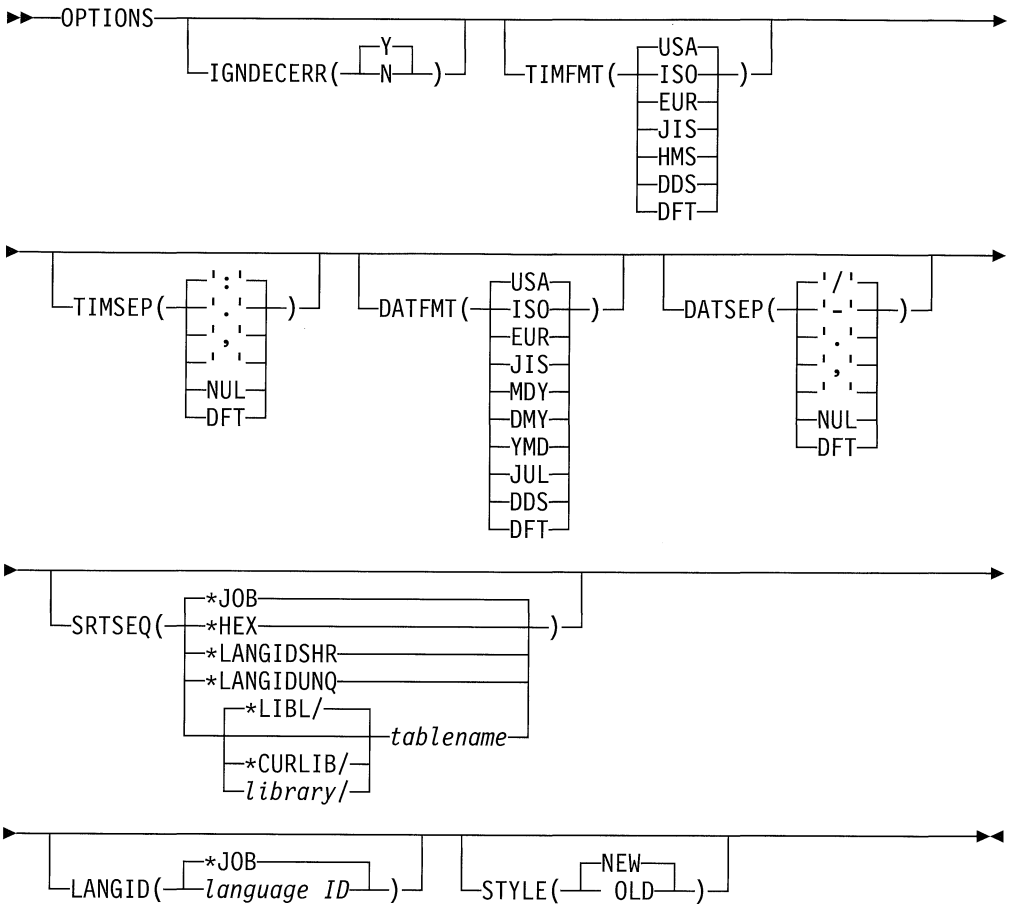


Figure 5-7. OPTIONS Transfer Request Syntax

The following sets user options for a subsequent SELECT request:

```
OPTIONS  IGNDECERR(Y)
         DATFMT(YMD)
         DATSEP('-')
```

In this example, decimal data errors are ignored, and any DATE fields are received in a yy-mm-dd format.

The following sets user options for a subsequent REPLACE request:

```
OPTIONS  TIMFMT(DDS)
```

In this example, any TIME fields sent to the AS/400 system use the format and separator as defined for the corresponding field.

Transfer Function API for DOS Users

If you use the DOS version of the transfer function, your application program must access the PC registers and perform interrupts to send transfer requests from the personal computer to the system and transfer records. Your application program could also interface to an assembler language subroutine that does these things. Refer to "DOS Program Example" on page 5-35 for an example of an application program, written in PASCAL, that links to two assembler language subroutines.

The IBM-supplied API program (STF.EXE) must be loaded and run before your application program can run. (The PC Support router must be started before STF.EXE can run.) STF.EXE installs the IBM-supplied API program as a DOS-resident interrupt handler servicing interrupt line X'69' (or the interrupt line you specified on the EIMPCS device driver in your CONFIG.SYS file).

Enter the following to install STF:

```
[d:] [path] STF [number_sessions]
```

d: Drive letter where STF.EXE resides

path
Path on the specified drive where STF.EXE resides

number_sessions
How many transfer request sessions you want to have active at the same time. Valid values are 1 through 8. The default value is 1. This parameter is optional.

For example, your application program might be designed to handle retrieving data from two different AS/400 files, and then sending some of the data from these two files to a third file on the AS/400 system. In this case, your application could be designed to have all three transfer requests active at the same time. You would need to specify 3 for the number_sessions parameter.

Note: Your application program must supply a unique 4096-byte buffer for each transfer request session.

After STF has been installed, you can start it again by calling STF again. You can also increase the number of sessions if necessary. The following happens when you have restarted STF:

1. Any currently active transfer requests are ended.
2. Additional storage is allocated to support any additional sessions you may have requested with the number_sessions parameter.

   Note: This method of increasing the number of sessions is not recommended because the storage use will not be as efficient. If storage is critical on your personal computer and you need STF to support additional sessions, it is recommended that you remove STF from storage using the RMVPCS command and then install STF again and specify the desired number of sessions.

To determine if STF.EXE is installed, check offset X'103' for each interrupt from X'60' to X'69'. If STF.EXE is installed, you will find the STF identifier, where STF is an ASCII character string. The example STFFOUND assembler routine performs this check. Refer to the STFFOUND routine in the "DOS Program Example" on page 5-41 for an example of this routine.

The interface to STF.EXE is similar to that of a DOS function call. Your application program must do the following:

1. Set up the personal computer registers for the function to perform.
2. Supply a buffer for STF.EXE to use.
3. Perform a program interrupt.

Then STF.EXE does the following:

1. Receives control.
2. Attempts to perform the requested function.
3. Sets register AX to indicate whether the operation is successful.
4. Returns control to your application program with the personal computer registers set to indicate the status of the request.

The return codes could be as follows:

Return Code (hex)	Description
0	Operation was unconditionally successful.
1 to 1FFF	Operation was successful, but a warning condition was encountered.
2000 and over	Operation was not successful due to severe errors.

Refer to "DOS Return Code Summary" on page 5-34 for a summary of the return codes set by STF.EXE.

Requesting Functions Using the DOS Operating System

To request functions, set register AL to the function you want to perform. (Refer to "Register Settings for Functions" on page 5-21 for information on register settings for each function.) You must also provide a 4096-byte buffer, called the transfer request buffer, to be associated with the transfer request while it is open. A transfer request is associated with its transfer request buffer until the transfer request closes. Any functions requested for that transfer request must specify that buffer. Your application program and STF.EXE use this buffer to pass the data back and forth to one another.

Your application program can request the following functions:

- Open a transfer request
- Retrieve the templates
- Retrieve the records
- Close the transfer request
- End all transfer request conversations
- Send records
- End the transfer request conversation

Refer to "Register Settings for Functions" on page 5-21 for more information on these functions.

A transfer request is considered open when an *open a transfer request* function is completed. Then the application program can request or send data.

An application program can have up to eight transfer requests open simultaneously, and can send and retrieve data from all eight in any order, as long as each transfer request is associated with its own 4096-byte buffer. You may, however, be limited by the number of router sessions available, or by the availability of system resources to run the system transfer function program.

You cannot send a "SELECT . . ." transfer request and a "REPLACE . . ." transfer request using the same buffer. If you want to use both types of transfer requests, you must allocate a transfer request buffer for "SELECT . . ." requests and allocate another transfer request buffer for "REPLACE . . ." requests. You can send an "EXTRACT . . ." transfer request in either of the transfer request buffers.

Once a request is open, your application program can perform the *retrieve the templates function* (AL=02) for either a SELECT, EXTRACT, or REPLACE transfer request.

Then, for an AS/400 system-to-personal-computer transfer request, you could receive data using the *retrieve the records* function (AL=03). For a personal-computer-to-AS/400 transfer request, you could send data using the *send records* function (AL=06).

Register Settings for Functions

This section provides information on the functions your application program may use and on setting the PC registers for these functions when using the transfer function application program with the DOS operating system. The registers are first discussed in general, and then discussed for each function.

The input registers to STF.EXE generally contain the following values:

AL — The code for the function you want STF.EXE to perform.

AH — The code for the type of translation you want STF.EXE to perform when retrieving and sending records.

ES:DI — A segment:offset address that points to a string containing the name of the system in which you want your transfer request to run.

DS:BX — A segment:offset address pointing to the transfer request buffer.

CX — The length of the data being sent to STF.EXE.

DX — The length of the transfer request buffer (must be 4096).

The following values are generally returned in the output registers by STF.EXE:

AX — A return code indicating whether the requested function was successful. Refer to "DOS Return Code Summary" on page 5-34 for a summary of the return codes set by STF.EXE.

DS:BX — A segment:offset address pointing to the transfer request buffer location where STF.EXE placed the requested data.

Note: If register AX contains X'3000', X'3100', or X'0400', register DS:BX will point to error message text and register CX contains the length of this error message text.

CX — Contains one of the following:

- The length of the data returned from STF.EXE.
- A count of the number of templates that can be retrieved.

DX — Contains one of the following:

- A secondary error code.
- The number of templates that may be retrieved if a warning message occurred during the open of a REPLACE transfer request.

Open a Transfer Request Function (AL=01)

This function passes a transfer request to STF.EXE identifying the records it wants to transfer to or from the requested system.

You must provide a 4096-byte buffer, called the transfer request buffer, to be associated with the transfer request while it is open. A transfer request is associated with its transfer request buffer until the transfer request closes. Any functions requested for that transfer request must specify that buffer. Your application program and STF.EXE use this buffer to pass the data back and forth to one another.

Put the following values in the input registers to STF.EXE:

AL — The code for the function you want to perform. Set AL to 01 to open a transfer request.

ES:DI — The segment:offset address of the following string:

```
SYSTEM(SYSTEM01)
```

In this string, SYSTEM is a keyword and SYSTEM01 is the system name where you want your request to run. The field should be ASCII, left-justified, from 1 to 8 bytes long, and padded with spaces. If you want the default system, do one of the following:

- Set the registers ES:DI to zeros.
- Set the address of ES:DI to point to the string SYSTEM().
- Set the address of ES:DI to point to an 8-byte string of hexadecimal zeros.

Once you supply a system name and that transfer request opens, that system name cannot change. If you want a different system, your application program must do one of the following:

- Open a new transfer request specifying a different buffer.
- Close and end the current transfer request, then open a new transfer request specifying the same buffer.

DS:BX — The address of a 4096-byte buffer. This buffer contains the transfer request describing the data your PC application program sends or retrieves.

The transfer request must be an EXTRACT, REPLACE, OPTIONS, or SELECT transfer request, and must be in ASCII format. (Refer to "Transfer Requests" on page 5-5 for more information.)

CX — The length (in bytes) of the transfer request.

DX — The length of the transfer request buffer (4096 bytes).

The following values are returned in the output registers by STF.EXE:

AX — The return code indicating whether the requested function was successful. Refer to "DOS Return Code Summary" on page 5-34 for a summary of the return codes set by STF.EXE.

DS:BX — The address in the transfer request buffer of the error message text if the return code in register AX is not zero and register CX is not zero.

CX Contains one of the following:

- A count of the number of templates that can be retrieved if the return code in register AX for the open was zero. Templates can be retrieved for the SELECT, EXTRACT, and REPLACE transfer requests.
- The length of the error message text returned from the system if the return code in register AX contains X'3000', X'3100', or X'0400'.

DX One of the following, depending on the return code in register AX:

- A zero if the return code in register AX is zero
- A count of the number of templates that can be retrieved if the return code in register AX contains X'0400' (a warning message) and the request is a REPLACE transfer request. In this case, register CX contains the length of the warning message where it would normally contain a count of the number of templates.
- The position of the first untranslatable byte in the SELECT, EXTRACT, or REPLACE transfer request if the return code in register AX is X'2002'.
- A value that refers to the position in the transfer request where the error was detected if the return code in register AX is X'3000'.

After this function is successfully performed, the transfer request is considered open and a transfer request buffer is associated with the request. Now your application program can request or send data.

Note: For a REPLACE transfer request, if the file member contains data, the return code for the open is X'0400'. This is a warning that if you continue, the data in the file member will be replaced. If you continue and send records (AL=06), the data is replaced. If you do not continue and you issue function AL=04 (close transfer request) to close the transfer request, data in the file member remains intact.

Retrieve the Templates Function (AL=02)

This function transfers a group of records, called templates, to your application program. This group of records contains a description of the fields within the system database file for SELECT and REPLACE transfer requests. This group of records contain a description of what the lists will look like for an EXTRACT transfer request. Your application program saves the information in the templates for interpreting the actual records that are returned.

Each template contains the name, length, data type, and the maximum size of the number for a field that your application program requests. For binary, zoned decimal, or packed decimal data, the maximum size is expressed in decimal digits. For character or hexadecimal data, the decimal digits are always hexadecimal zero.

A template has the following format:

Byte 1 (field type):

00	Hexadecimal
01	Binary
02	EBCDIC character

03	EBCDIC zoned decimal
04	EBCDIC packed decimal
05	Reserved
06	IGC open
07	IGC only
08	IGC either
0B	Time
0C	Date
0D	Timestamp
0F	Graphic

Bytes 2 and 3 (field length):

- For hexadecimal and character data types, these bytes contain the total length (in bytes) of the field. If the field is variable length, it does not contain the 2-byte length of the field.
- For binary, packed and zoned data types, byte 2 contains the length (in bytes), and byte 3 contains the number of decimal positions to the right of the decimal point.

Bytes 4 through 33 (field name):

Name of the field as defined by the record format for the AS/400 file, left-adjusted, and padded with spaces.

Note: If this is a JOIN request, the file qualifier precedes the field name. For example, Tx.FIELD, where x is 1 through 32.

Byte 34 (digits):

For binary, packed, and zoned data types, this byte contains the maximum number of digits that the field can hold. For example, if the maximum number of digits is 3, the maximum value of the field cannot be greater than 999.

Byte 35

B'1xxxxxxx' Null-capable

B'x1xxxxxx' Variable length

B'xx1xxxxx' Reserved

B'xxx1xxxx' Reserved

B'xxxx1xxx' Reserved

B'xxxxx1xx' Reserved

B'xxxxxx1x' Reserved

B'xxxxxxx1' Reserved

Bytes 36 through 40:

Reserved.

Put the following values in the input registers to STF.EXE:

AL The code for the function you want to perform. Set AL to 02 to retrieve templates from the system.

DS:BX The address of the transfer request buffer associated with the active transfer request.

The following values are returned in the output registers by STF.EXE:

AX The return code indicating whether the requested function was successful. Refer to "DOS Return Code Summary" on page 5-34 for a summary of the return codes set by STF.EXE.

DS:BX One of the following:

- The address in the transfer request buffer of the template if the return code in register AX is zero.
- The address in the transfer request buffer of the error message text if the return code in register AX is not zero and register CX is not zero.

CX One of the following, depending on the return code in register AX:

- The length, in bytes, of the retrieved template if the return code in register AX is zero. This length is 40 (X'28').
- The length of the error message text returned from the system if the return code in register AX is X'3000' or X'3100'.

DX Should always contain zero.

One template returns for each application program request AL=02. After all templates are returned, further requests for templates result in an end-of-file return code (register AX = X'1FFF'). You receive one template for each field you specified in the SELECT or REPLACE statement.

Retrieve the Records Function (AL=03)

This function retrieves records from the system and allows you to specify the format in which your application program receives records when you transfer data from the system to the personal computer.

Put the following values in the input registers to STF.EXE:

AL The code for the function you want to perform. Set AL to 03 to retrieve records from the system.

AH Controls whether the records are changed to a format compatible with personal computer ASCII format or are left in system EBCDIC format. Set this register to one of the following:

X'00' The records do not change and are received from the system exactly as they appear. Null fields contain zeros for numeric fields and blanks for character and hexadecimal fields. Variable-length fields are padded with blanks for character and hexadecimal fields, or zeros for numeric fields.

X'01' The records change in the same way the interactive or automatic transfer function program changes records when writing to a DOS random file.

X'11' The records change in the same way the interactive or automatic transfer function programs change records when writing to a DOS random type 2 file.

X'02' The records change to DOS random file format, with the following exceptions:

- When an untranslatable EBCDIC zoned decimal byte is found, the byte changes to an X'3F'.
- When an untranslatable EBCDIC packed decimal byte is found, the byte changes to an X'FF'.

X'12' The records change to DOS random type 2 file format, with the following exceptions:

- When an untranslatable EBCDIC zoned decimal byte is found, the byte is changed to an X'3F'.
- When an untranslatable EBCDIC packed decimal byte is found, the byte is changed to an X'FF'.

X'80' The records are not converted, and information about null and variable-length fields is retained.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte record length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field does contain a null value.

X'81' The records change in the same way the interactive or automatic transfer function program changes records when writing to a DOS random file.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte record length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field does contain a null value.

X'82' The records change to DOS random file format, with the following exceptions:

- When an untranslatable EBCDIC zoned decimal byte is found, the byte changes to an X'3F'.

- When an untranslatable EBCDIC packed decimal byte is found, the byte changes to an X'FF'.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte record length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field does contain a null value.

X'91' The records change in the same way the interactive or automatic transfer function programs change records when writing to a DOS random type 2 file.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

X'92' The records change to DOS random type 2 file format, with the following exceptions:

- When an untranslatable EBCDIC-zoned decimal byte is found, the byte is changed to an X'3F'.
- When an untranslatable EBCDIC-packed decimal byte is found, the byte is changed to an X'FF'.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte record length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the

field does not contain a null value, or X'F1', which indicates the field does contain a null value.

If this register is set to something other than the above values, the records change as if the register had been set to X'01'.

DS:BX The address of the transfer request buffer associated with the active transfer request.

The following values are returned in the output registers by STF.EXE:

AX The return code indicating whether the requested function was successful. Refer to "DOS Return Code Summary" on page 5-34 for a summary of the return codes set by STF.EXE.

DS:BX One of the following:

- The address in the transfer request buffer of the returned record if the return code in register AX is zero.
- The address in the transfer request buffer of the error message text if the return code in register AX is not zero and register CX is not zero.

CX One of the following, depending on the return code in register AX:

- The length, in bytes, of the retrieved record if the return code in register AX is zero.
- The length of the error message text returned from the system if the return code in register AX is X'3000' or X'3100'.

DX One of the following, depending on the return code in register AX:

- A zero, if the return code in AX is zero.
- The number of untranslatable characters encountered in the record if the return code in register AX is X'0300'.

After all records for the transfer request are returned, the next request for a record results in an end-of-file return code (register AX = X'1FFF'), and the transfer request is automatically closed.

Close the Transfer Request Function (AL=04)

This function tells the system to close the file associated with the transfer request. This function should be used after your application program has finished sending or receiving records to or from a particular file.

Put the following values in the input registers to STF.EXE:

AL The code for the function you want to perform. Set AL to 04 to close the transfer request.

DS:BX The address of the transfer request buffer for an active transfer request.

The following values are returned in the output registers by STF.EXE:

AX The return code indicating whether the requested function was successful. Refer to "DOS Return Code Summary" on page 5-34 for a summary of the return codes set by STF.EXE.

DS:BX The address in the transfer request buffer of the error message text if the return code in register AX is not zero and register CX is not zero.

CX The length of the error message text returned from the system if the return code in register AX is X'3000' or X'3100'.

DX Should always contain zero.

If the request is successful, the transfer request is closed. No more data can transfer.

You can also close a transfer request by retrieving all records (AL=03) from it until an end-of-file condition is returned, or by opening another transfer request using the transfer request buffer associated with the previous transfer request.

Notes:

1. If another transfer request is opened using the same transfer request buffer, the transfer request must be of the same type as the previous transfer request. For example, if a "SELECT . . ." transfer request opened previously, another SELECT transfer request must be opened (an "EXTRACT . . ." is also allowed). If a "REPLACE . . ." transfer request opened previously, another REPLACE transfer request must be opened (an "EXTRACT . . ." is also allowed).
2. Closing a request does not reset any options that have been sent to the host system. An end conversation (X'07' or X'05') must be issued, or another options request must be submitted to reset the options.

End All Transfer Request Conversations Function (AL=05)

This function closes all transfer requests immediately and ends any current requests for data.

Note: In most cases your API program will probably be working with one session at a time. In these cases, use function 07, which ends one transfer request conversation at a time. Use function 05 if you have multiple conversations active and you need to end all of them.

Put the following value in the input register to STF.EXE:

AL The code for the function you want to perform. Set AL to 05 to end all transfer request conversations.

The following values are returned in the output registers by STF.EXE:

AX The return code indicating whether the requested function was successful. Refer to "DOS Return Code Summary" on page 5-34 for a summary of the return codes set by STF.EXE.

DS:BX The address in the transfer request buffer of the error message text if the return code in register AX is not zero and register CX is not zero.

CX The length of the error message text returned from the system if the return code in register AX is X'3000' or X'3100'.

DX Should always contain zero.

Send Records Function (AL=06)

This function sends records to the system and allows you to specify the format in which your application program sends records when you transfer data from the personal computer to the system.

Put the following values in the input registers to STF.EXE:

AL — The code for the function you want to perform. Set AL to 06 to send records to the system.

AH — Controls whether STF.EXE converts the data from personal computer DOS random format to system format. For example, if the transfer request buffer contains an 8-byte ASCII character field, STF.EXE would translate this to an 8-byte EBCDIC field before sending the record to the system. Set this register to one of the following:

X'00' — The records do not change and are sent to the system exactly as they appear on the personal computer. STF.EXE still checks for untranslatable data in zoned decimal and packed decimal fields.

If the AS/400 system is running using Version 2 Release 2 Modification 0, the null map and the 2-byte variable length for each variable-length field are added to each record. This ensures compatibility with the AS/400 database format.

X'01' — The records are converted from personal computer DOS random format to system format.

If the AS/400 system is running using Version 2 Release 2 Modification 0, the null map and the 2-byte variable length for each variable-length field are added to each record. This ensures compatibility with the AS/400 database format.

X'11' — The records are converted from personal computer DOS random type 2 format to system format.

If the AS/400 system is running using Version 2 Release 2 Modification 0, the null map and the 2-byte variable length for each variable-length field are added to each record. This ensures compatibility with the AS/400 database format.

X'80' — The records are not converted, and information about null and variable-length fields is retained and is sent as the fields appear on the personal computer. STF.EXE checks for untranslatable data in zoned-decimal and packed-decimal fields.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each

byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

If the AS/400 system is running using Version 2 Release 1.1 or any other previous release or version, the record is converted to the fixed-length fields without a null map. This conversion ensures compatibility with the AS/400 database variable-length record and null value support.

X'81' The records change in the same way the interactive or automatic transfer function program changes records when read from a DOS random file.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

If the AS/400 system is running using Version 2 Release 1.1 or any other previous release or version, the record is converted to the fixed-length fields without a null map. This conversion ensures compatibility with the AS/400 database variable-length record and null value support.

X'91' The records change in the same way the interactive or automatic transfer function programs change records when read from a DOS random type 2 file.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

If the AS/400 system is running using Version 2 Release 1.1 or any other previous release or version, the record is converted to the fixed-length fields without a null map. This conversion ensures compatibility with the AS/400 database variable-length record and null value support.

DS:BX The address of the transfer request buffer associated with the active transfer request.

The record that your application program sends to the system file must be within this buffer and start at the first byte. The fields within the record to be transferred must be in the same order as, and compatible with, the format of the system file, as described by the templates returned when the transfer request was opened. For example, assume the templates specified that the system file consisted of two fields, and that the first field was an 8-byte character field and the second was a 4-byte packed field. Any record you transfer to the AS/400 system should be formatted the same way in the transfer request buffer.

CX The length of the record to send.

The following values are returned in the output registers by STF.EXE:

AX The return code indicating whether the requested function was successful. Refer to "DOS Return Code Summary" on page 5-34 for a summary of the return codes set by STF.EXE.

Note: If the record to send contains one or more fields containing untranslatable character data, register AX is set to X'0302'.

DS:BX The address in the transfer request buffer of the error message text if the return code in register AX is not zero and register CX is not zero.

CX The length of the error message text returned from the system if the return code in register AX is X'3000' or X'3100'.

DX One of the following, depending on the return code in register AX:

- A zero if the return code in register AX is zero.
- The number of untranslatable bytes of character data if the return code in register AX is X'0302'.
- The position of the first untranslatable byte in a zoned or packed decimal field if the return code in register AX is X'2003'.
- The position of the field in which a digit range error occurred if the return code in register AX is X'2004'.

Before sending a record to the system file, STF.EXE checks the record for untranslatable data. If register AH is set to X'00' (no translation), STF.EXE only checks for untranslatable data in zoned decimal and packed decimal fields. If register AH is set to X'01' or X'11' (translation), STF.EXE checks all fields for untranslatable data.

If STF.EXE finds untranslatable character data in the record, the record transfers to the system after the following occurs:

- The untranslatable bytes are set to X'00'.
- Register DX is set to the number of untranslatable bytes in the record.
- Register AX is set to X'0302', indicating a warning (untranslatable character data found).

If STF.EXE finds untranslatable zoned decimal or packed decimal data in the record, the record is *not* sent to the system. The following occurs:

- Register DX is set to the position of the first untranslatable byte.
- Register AX is set to X'2003' (untranslatable numeric data found).

If STF.EXE does not find any untranslatable zoned decimal or packed decimal data in the record, STF.EXE checks all numeric data (binary, zoned decimal, and packed decimal) to see whether the number of digits fits the range specified by the template for that field.

As soon as STF.EXE detects a digit range error, processing stops. STF.EXE returns the position of the field in which the digit error occurred in register DX and sets register AX to X'2004' (numeric data does not fit in range specified by digits).

For example, assume that a packed decimal field contains X'5103' and the template for that field indicates that the number of digits is 2. STF.EXE would not send the record because the actual number contains 3 digits, 510 (remember that the last half-byte is the sign).

You can send one record to the system file each time you use this function. After you finish sending all records, issue function AL=04 to close the transfer request. The last group of records will be sent to the system and the system database file is closed.

Note: STF.EXE and the router send data records in groups to the system. If an error occurs, the actual number of records that arrive on the system cannot be predicted. For example, if you received an error return code while sending record 100, only 50 records may actually be in the system database file.

End the Transfer Request Conversation Function (AL=07)

This function tells the system to close the file associated with the transfer request after your application program sends or retrieves the records to or from a particular file. It also ends communication between the system and the personal computer for that transfer request. If the transfer request associated with this buffer is open, it automatically closes.

Note: Ending the transfer request is not the same as closing the transfer request. When you end a transfer request, the router conversation ends. When you close a transfer request, the router conversation remains active.

Use this function when you close a transfer request and do not plan to open another transfer request using the same buffer. This releases the personal computer and system resources that supported the communication for that transfer request.

Put the following values in the input registers to STF.EXE:

AL — The code for the function you want to perform. Set AL to 07 to end the transfer request conversation.

DS:BX — The address of the transfer request buffer.

The following values are returned in the output registers by STF.EXE:

AX — The return code indicating whether the requested function was successful. Refer to "DOS Return Code Summary" on page 5-34 for a summary of the return codes set by STF.EXE.

DS:BX — The address in the transfer request buffer of the error message text if the return code in register AX is not zero and register CX is not zero.

CX The length of the error message text returned from the system if the return code in register AX is X'3000' or X'3100'.

DX Should always contain zero.

DOS Application Program Interface Considerations

Consider the following when using the DOS version of the personal computer application program interface:

- The 4096-byte transfer request buffer indicated by register DS:BX cannot cross a segment boundary. Register BX cannot be greater than X'F000'.
- Generally, STF.EXE does not detect whether the data definitions on the system are incorrect or if the data in the file itself is incorrect or damaged. However, STF.EXE does check data sent to the system for errors such as untranslatable characters or too many digits in the field.
- If the application program already knows the format of the records being transferred, it can bypass retrieving templates and go directly from opening a transfer request to requesting or sending the records. However, it cannot go the other way. Once a request is made to retrieve or send records, the application program cannot retrieve templates without opening the transfer request again.

DOS Return Code Summary

Table 5-3 contains a summary of the return codes set by STF.EXE.

Table 5-3 (Page 1 of 2). STF.EXE Return Code Summary

Return Code	Description
0000	The previous operation was successful.
0103	Transfer request is already ended.
0300	Untranslatable data found in transferred record.
0302	Untranslatable data in record to be transferred to AS/400.
0400	Warning detected by the AS/400 system.
1FFF	End-of-file.
2000	Maximum number of transfer requests active.
2001	Transfer request length not valid.
2002	Transfer request character cannot be translated.
2003	Record not transferred to the AS/400 due to untranslatable numeric data.
2004	Numeric data does not fit into range specified by digits.
2005	Record length given by your application program is incorrect.
2007	Null map offset given by your application program is not correct.
2008	Total length of record exceeds 4096 bytes.
2010	Transfer request not opened. One of the causes could be that the 4096 KB buffer address has changed for an active transfer request.
2011	Templates cannot be retrieved now.

Table 5-3 (Page 2 of 2). STF.EXE Return Code Summary	
Return Code	**Description**
2012	Cannot get records on a Personal-Computer-to-AS/400 request.
2013	Cannot send records on a AS/400-to-Personal-Computer request.
2020	Incorrect function requested.
2021	Incorrect buffer length.
2022	Incorrect buffer address.
2023	Transfer request buffer overlaps a previous transfer request buffer.
2029	PCSWIN communications buffer too small.
2030	Options statement not supported on this system.
2031	Options not fully supported on this system.
2032	Conversation format not allowed with STYLE(NEW) in effect.
2033	STYLE keyword not supported on this system.
3000	Error detected by AS/400 — statement offset provided.
3100	Error detected by AS/400 — no statement offset provided.
5048	Connection failed to system &&&&&&&&.
5050	System name is incorrect or inactive.
5052	System &&&&&&&&& &&&&&&&&&&& &&&&&&&&&&&& program not found.
5054	System &&&&&&&&& &&&&&&&&&&& &&&&&&&&&&&& program ended unexpectedly.
5056	Security error occurred for system &&&&&&&&.
5058	System &&&&&&&& is not a supported system.
5060	Cannot continue: &&&&&&&& must be Release &.& Level &&.
5062	Contact with system &&&&&&&& ended.
5064	Contact with system &&&&&&&& temporarily interrupted.
5066	Resource failure on system &&&&&&&&.
5067	Conversation with &&&&&&&& was ended unexpectedly.
9999	Undefined error. Call your IBM service representative.

DOS Program Example

The following pages contain a sample program written in PASCAL. The program links to two subroutines written in assembler language. This program is an example of how you can use the application program interface to create a transfer request. It displays the names of customers for whom accounts receivable are 90 days overdue. The program then adds the overdue amounts to produce the total amount due.

This example does not show all programming considerations or techniques. Review the example before you begin application design and coding.

```
Program   Sample(Output);
Const                                   !Transfer request statement
  TReq      = 'Select LSTNAM,BALDUE from QIWS/QCUSTCDT where BALDUE > 0';
  M1        = 'An error occurred on open';
  M2        = 'An error occurred while getting templates';
  M3        = 'An error occurred while getting records';
  EOF       = #1FFF;                    !End of File return code
Type
  Buffer    = Array[1..4096] of Char; !Buffer needed by STF
  RCType    = Record                  !Return codes from STF
    Main      : Word;                 !Value returned in AX
    Second    : Word;                 !Value returned in DX
    DataLen   : Integer;              !Data or message length (CX)
    End;
```

Figure 5-8 (Part 1 of 7). Application Program Interface Program Example for DOS Users

```
Var
  TRLen     : Integer;              !Transfer request length
  Buf       : Buffer;               !Transfer request buffer
  Start     : Integer;              !Starting position of data in
                                    !buffer
  RC        : RCType;               !Return codes from STF
  Name      : Lstring(255);         !Customer name
  Balance   : Real;                 !Balance due
  Total     : Real;                 !Balance due total
  NameLen   : Integer;              !Length of name field
  BalLen    : Integer;              !Length of Balance due field
  I         : Integer;              !Index counter

!-----------------------------------------------------------------
! Call the Application Program Interface (STF) - Assembler routine
!-----------------------------------------------------------------
Procedure Request (    Op      : Byte;     !Operation code
                       TRLen   : Integer;  !Transfer request length
                   Var Buf     : Buffer;   !Transfer request buffer
                   Var Start   : Integer;  !Starting data position
                   Var RetCode : RCType);  !Return codes
Extern;

!-----------------------------------------------------------------
! Make sure STF has been loaded - Assembler routine
!-----------------------------------------------------------------
Function  STFFound : Boolean;  Extern;

!-----------------------------------------------------------------
! Convert a Zoned decimal field to a real number
!-----------------------------------------------------------------
Procedure ZoneToReal (Pos  : Integer; !Starting buffer position
                      Len  : Integer; !Field length
                  Var Num  : Real);   !Resulting number

Var
  C         : Integer;              !Value of current buffer
                                    !character
  Scale     : Real;                 !Power of 10 for the current
                                    !digit

Begin
  Scale := 0.01;                    !The last digit is the cents
                                    !position
  Num   := 0;                       !Initialize the number to 0
  For I := Len DownTo 1 Do          !Work backward toward start
                                    !of field
    Begin
      C := Ord(Buf[Pos+I-1]);       !Get the byte from the buffer
      Num := Num + (C Mod 16)*Scale;!Convert the digit
      Scale := Scale * 10;          !Set the scale factor
    End;
End;
```

Figure 5-8 (Part 2 of 7). Application Program Interface Program Example for DOS Users

```
!-------------------------------------------------------------------
! Get the templates for each field
!-------------------------------------------------------------------
Procedure GetTemplates;
Begin
  Request (2,TRLen,Buf,Start,RC);     !Get customer name template
  If RC.Main < EOF Then               !Check for an error
    Begin
      NameLen := Ord(Buf[Start+1]);   !Get length of name (integer)
      Name.Len := Wrd(Buf[Start+1]);  !Get length of name (string)
      Request (2,TRLen,Buf,Start,RC); !Get Balance due template
      If RC.Main < EOF Then           !Check for an error
        BalLen := Ord(Buf[Start+1])   !Get length of Balance due
    End;
End;

!-------------------------------------------------------------------
! Get the data records
!-------------------------------------------------------------------
Procedure GetRecords;
Begin
  Repeat
    Request (3,TRLen,Buf,Start,RC);   !Get the next record
    If RC.Main < EOF Then             !Check for error or end of
                                      !file
      Begin
        For I := 1 To NameLen Do      !Get the customer name
          Name[I] := Buf[Start+I-1];
        ZoneToReal (Start+NameLen,BalLen,Balance);
                                      !Get the Balance due
        Total := Total + Balance;     !Total the Balance due
        WriteLn (Name,Balance:Ballen+11:2) !Print the record
      End;
  Until RC.Main >= EOF;
End;

!-------------------------------------------------------------------
! Display error status
!-------------------------------------------------------------------
Procedure ErrorMsg;

Begin
  WriteLn ('Main return code = ',RC.Main:4:16);
  WriteLn ('Secondary return code = ',RC.Second);
  If RC.DataLen > 0 Then
    Begin
      Write ('Message = ');
      For I := 1 to RC.DataLen Do Write (Buf[Start+I-1]);
      WriteLn
    End;
End;
```

Figure 5-8 (Part 3 of 7). Application Program Interface Program Example for DOS Users

```
!-------------------------------------------------------------
! Main procedure
!-------------------------------------------------------------
Begin
  If STFFound=False Then WriteLn ('STF is not loaded')
  Else Begin
    TRLen := Upper (TReq);            !Get length of statement
    UnPack (TReq,Buf,1);              !Put statement into buffer
    Request (1,TRLen,Buf,Start,RC);   !Open the transfer request
    If RC.Main > 0 Then WriteLn (M1)  !Check for an error
    Else Begin
      GetTemplates;                   !Get the field templates
      If RC.Main > EOF Then           !Check for an error
        WriteLn (M2)
      Else Begin
        Write ('Customer',' ':NameLen-8); !Display column headings
        Write ('Balance Due':BalLen+11);
        WriteLn; WriteLn;
        GetRecords;                   !Get the records
        If RC.Main > EOF Then         !Check for an error
          WriteLn (M3)
        Else Begin
          WriteLn;
          Write ('Total',' ':Namelen-5);
          WriteLn (Total:BalLen+11:2)  !Display the total
        End;
      End;
    End;
    If RC.Main <> EOF Then
      ErrorMsg;                       !Display error status
    Request (7,TRLen,Buf,Start,RC)    !Clean up
  End;
End.

PAGE     66,132
NAME     PROCS
TITLE    Assembler routines for calling STF
PUBLIC   REQUEST,STFFOUND
;-------------------------------------------------------------
; Parameters passed to the REQUEST procedure
;-------------------------------------------------------------
RPARMS   STRUC
  RBP    DW      ?             ;Saved frame pointer
  RRET   DW      2 DUP(?)      ;Return address
  RRC    DW      ?             ;Return code address
  RPOS   DW      ?             ;Starting position of data
  RBUF   DW      ?             ;Buffer address
  RLEN   DW      ?             ;Statement length
  RREQ   DB      ?             ;Request code
  RREQ1  DB      ?             ;Unused byte
RPARMS   ENDS
```

Figure 5-8 (Part 4 of 7). Application Program Interface Program Example for DOS Users

```
;-------------------------------------------------------------
; Return code parameter structure
;-------------------------------------------------------------
RCTYPE   STRUC
  RC1    DW      ?             ;Main return code
  RC2    DW      ?             ;Secondary return code
  DLEN   DW      ?             ;Length of data (or message)
RCTYPE   ENDS

;-------------------------------------------------------------
; Save the registers at the start of procedure or function
;-------------------------------------------------------------
INIT     MACRO   INDICATOR
         PUSH    BP            ;Save frame pointer
         MOV     BP,SP         ;Point to parms
         IFDIF  <INDICATOR>,<'FUNCTION'>
           PUSH    AX          ;AX returns value for a function
         ENDIF
         PUSH    BX            ;Save registers
         PUSH    CX
         PUSH    DX
         PUSH    SI
         PUSH    DI
         PUSHF
         PUSH    DS
         PUSH    ES
         ENDM

;-------------------------------------------------------------
; Restore the registers at the start of a procedure or function
;-------------------------------------------------------------
CLEANUP  MACRO   INDICATOR
         POP     ES            ;Restore registers
         POP     DS
         POPF
         POP     DI
         POP     SI
         POP     DX
         POP     CX
         POP     BX
         IFDIF  <INDICATOR>,<'FUNCTION'>
           POP     AX          ;AX returns value for a function
         ENDIF
         POP     BP
         ENDM
```

Figure 5-8 (Part 5 of 7). Application Program Interface Program Example for DOS Users

```
;-------------------------------------------------------------
; The code that can be run begins here
;-------------------------------------------------------------
CODE     SEGMENT 'CODE'
         ASSUME  CS:CODE
SYSNAME  DB       8 DUP(00h)    ;Default system name
;
;
REQUEST  PROC    FAR            ;Make a request of STF
         INIT    'PROCEDURE'    ;Save registers, procedure setup
         MOV     AH,1           ;Set translation flag
         MOV     AL,[BP].RREQ   ;Set request code
         MOV     BX,[BP].RBUF   ;Set buffer address
         MOV     CX,[BP].RLEN   ;Set statement length
         MOV     DX,4096        ;Set buffer length
         PUSH    CS
         POP     ES             ;Set extra segment register
         LEA     DI,SYSNAME     ;Set offset of the system name parm
         DB      0CDH           ;Hex equivalent for INT instr
INTVAL   DB      060H           ;INTVAL is the interrupt at
                                ;which STF is installed
         MOV     DI,[BP].RRC    ;Get return code address
         MOV     [DI].RC1,AX    ;Store main return code
         MOV     [DI].RC2,DX    ;Store secondary return code
         MOV     [DI].DLEN,CX   ;Store length of returned data
         MOV     DI,[BP].RPOS   ;Point to caller's data pointer
         SUB     BX,[BP].RBUF   ;Set BX to offset into array
         INC     BX             ;Set BX to array index
         MOV     [DI],BX        ;Store pointer
         CLEANUP 'PROCEDURE'    ;Restore registers
         RET     6              ;Return
REQUEST  ENDP

;-------------------------------------------------------------
; To find out if STF is loaded, look for 'STF' at offset 0103H
;-------------------------------------------------------------
STFID    DB      'STF'          ;STF identifier
STFLEN   EQU     $ - STFID      ;Length of STF identifier
;
STFFOUND PROC    FAR            ;Determine if STF is loaded
         INIT    'FUNCTION'     ;Save registers, function setup
         CLD                    ;Set direction for string comparison
         MOV     AH,35H         ;DOS function call ID -get interrupt
         PUSH    CS             ;Push CS
         POP     DS             ;Set DS from CS
         MOV     AL,INTVAL      ;Initialize 1st valid interrupt to
                                ;check
```

Figure 5-8 (Part 6 of 7). Application Program Interface Program Example for DOS Users

```
NEXTINT: INT     21H             ;Get the interrupt vector from DOS
         LEA     SI,STFID        ;Offset (CS) of search string
         MOV     DI,0103H        ;Offset (ES) to be compared
         MOV     CX,STFLEN       ;Length of string comparison
                                 ;Is this the STF interrupt?
         REPE    CMPS STFID[SI],[DI]
         JE      FOUND           ;Yes - jump to FOUND
         INC     INTVAL          ;No - look at the next valid
                                 ;interrupt
         MOV     AL,INTVAL       ;Set AL to the next interrupt to
                                 ;check
         CMP     AL,080H         ;Was the last valid interrupt
                                 ;checked?
         JB      NEXTINT         ;No - Check the interrupt line
         MOV     AL,0            ;Yes  - return FALSE to caller
         JMP     FEXIT           ;Exit
FOUND:   MOV     INTVAL,AL       ;Set interrupt value to STF
                                 ;interrupt
         MOV     AL,1            ;Return TRUE to calling program
FEXIT:   CLEANUP 'FUNCTION'      ;Restore registers
         RET                     ;Return
STFFOUND ENDP
;
CODE     ENDS
         END
```

Figure 5-8 (Part 7 of 7). Application Program Interface Program Example for DOS Users

When this program runs, it displays the data records shown in Figure 5-9.

```
Customer                    Balance Due

Henning                           37.00
Jones                            100.00
Vine                             439.00
Johnson                         3987.50
Stevens                           58.75
Alison                            10.00
Doe                              250.00
Williams                          25.00
Lee                              489.50
Abraham                          500.00

Total                           5896.75
```

Figure 5-9. Resulting Data Records for DOS Users

Transfer Function API for OS/2 Users

If you use the OS/2 version of the transfer function, your application program must communicate with the IBM-supplied API, EHNTFSTF.DLL, to send transfer requests from the personal computer to the system and to transfer records. EHNTFSTF.DLL is a dynamic link library with one entry point. This entry point is also named EHNTFSTF. For the PASCAL API program example, the following import statement was used during the link process:

```
EHNTFSTF = EHNTFSTF.EHNTFSTF
```

Refer to "OS/2 Program Example" on page 5-61 for PASCAL program examples.

Note: The format of this import statement may vary depending on the compiler and linker you are using. Refer to the appropriate documentation for your compiler and linker to determine the format of this import statement when linking your application program.

Refer to the OS/2 technical documentation for a description of linking to dynamic link routines.

Your application program must set parameters and supply a buffer for EHNTFSTF.DLL with the following interface:

```
EhnTfStf (function,
          buffer address,
          buffer length,
          request length,
          system address,
          conversion,
          return address,
          return length,
          return code second)
```

Note: EHNTFSTF receives the parameters specified on the call in PASCAL order. The first parameter (function) is pushed on the stack first and the last parameter (return code second) is pushed on the stack last. Input parameters are passed by value. Output parameters, passed by reference, are pointers to where EHNTFSTF places the output values.

The interface to EHNTFSTF.DLL is similar to that of an OS/2 API function call. Your application program must do the following:

1. Declare the EHNTFSTF function as EXTERNAL FAR.
2. Set up the EHNTFSTF function parameters for the function to be performed.
3. Supply a buffer for EHNTFSTF.DLL to use.
4. Perform a FAR call to the EHNTFSTF dynamic link unit.

Then EHNTFSTF does the following:

1. Receives control.
2. Attempts to perform the requested function.
3. Sets register AX to indicate whether the operation is successful.
4. Returns control to your application program with the output parameters set to indicate the status of the request.

The return codes could be as follows:

Return Code (Hex)	Description
0	Operation was unconditionally successful.

1 to 1FFF	Operation was successful, but a warning condition was encountered.
2000 and over	Operation was not successful due to severe errors.

Refer to "OS/2 Return Code Summary" on page 5-59 for a summary of the return codes set by EHNTFSTF.DLL.

The OS/2 Communications Manager uses your application program's stack when your application program performs requests. The Communications Manager requires approximately 2560 bytes. You need additional stack space for your application program's own variables and the variables used by the EHNTFSTF dynamic link library. Therefore, the recommended stack size is at least 3000 bytes. Refer to the APPC programming information for OS/2 for additional information about stack size requirements.

Requesting Functions Using the OS/2 Operating System

To request a function, set the function parameter to the function you want to perform. (Refer to "Function Call Parameters" on page 5-45 for information on the parameters for each function.) You must also provide a 4096-byte buffer, called the transfer request buffer, to be associated with the transfer request while it is open. A transfer request is associated with its transfer request buffer until the transfer request closes. Any function requested for that transfer request must specify that buffer. Your application program and EHNTFSTF use this buffer to pass the data between each other.

You cannot send a "SELECT . . ." transfer request and a "REPLACE . . ." transfer request using the same buffer. If you want to use both types of transfer requests, you must allocate a transfer request buffer for "SELECT . . ." requests and allocate another transfer request buffer for "REPLACE . . ." requests. You can send an "EXTRACT . . ." transfer request in either of the transfer request buffers.

Your application program can request the following functions:

- Open a transfer request
- Retrieve the templates
- Retrieve the records
- Close the transfer request
- End a transfer request conversation
- End all transfer request conversations
- Send records

Refer to "Function Call Parameters" on page 5-45 for more information on these functions.

A transfer request is considered open when an *open a transfer request* function is completed. Then the application program can request or send data.

An application program can have an unlimited number of transfer requests simultaneously open against database files. You may, however, be limited by the number of router sessions available, or by the availability of system resources to run the system transfer function program.

Once a request is open, your application program can perform the *retrieve the templates* function (function parameter = 02) for either a SELECT, EXTRACT, or REPLACE transfer request.

Then, for an AS/400 system-to-personal-computer transfer request, you could receive data using the *retrieve the records* function (function=03). For a personal-computer-to-AS/400 transfer request, you could send data using the *send records* function (function=06).

Function Call Parameters

This section provides information on function calls and their parameters for the OS/2 version of the transfer function application program. The parameters are first discussed in general, and then discussed for each function.

The input parameters to EHNTFSTF generally contain the following values:

function
: A 1-byte value, ranging from 01 to 07, specifying the code for the function you want EHNTFSTF.DLL to perform.

buffer address
: A 4-byte selector:offset address pointing to the transfer request buffer.

buffer length
: A 2-byte value specifying the length of the transfer request buffer. It must be 4096 bytes.

request length
: A 2-byte value specifying the length of the data sent to EHNTFSTF.DLL.

system address
: A 4-byte selector:offset address pointing to the string containing the name of the system in which you want your request to run.

conversion
: A 1-byte value specifying the code for the type of translation you want EHNTFSTF to perform when transferring records.

EHNTFSTF.DLL returns the following values in the output parameters:

register AX
: A return code indicating whether the requested function was successful. Refer to "OS/2 Return Code Summary" on page 5-59 for a summary of the return codes set by EHNTFSTF.DLL.

return address
: A 4-byte selector:offset address pointing to the address within the transfer request buffer where the requested data has been placed.

  Note: If register AX contains X'3000', X'3100', or X'0400', the return address parameter points to the error message text and the return length parameter contains the length of this error message text.

return length
: A 2-byte value which contains one of the following:

  - The length of the data returned from EHNTFSTF.DLL.
  - A count of the number of templates that can be retrieved.

return code second
: A 2-byte value which contains one of the following:

  - A secondary error code.

- The number of templates that may be retrieved if a warning message occurred during the open of a REPLACE transfer request.

Note: The return address, return length, and return code second are passed by reference to EHNTFSTF. In other words, your application program must pass the address (a far pointer) of each parameter to EHNTFSTF. Refer to "OS/2 Program Example" on page 5-61 for an example of this.

Open a Transfer Request Function (Function=01)

This function passes a transfer request to EHNTFSTF.DLL, identifying the records it wants to transfer to or from the requested system.

Use the following values for the input parameters:

function
: Code for the function you want to perform. Set this parameter to 01 to open a transfer request.

buffer address
: Address of a 4096-byte buffer. This buffer contains the transfer request that describes the data your PC application program sends or retrieves.

  The transfer request must be an EXTRACT, REPLACE, OPTIONS, or SELECT transfer request, and must be in ASCII format. (Refer to "Transfer Requests" on page 5-5 for more information.)

request length
: Length (in bytes) of the transfer request.

buffer length
: Length of the transfer request buffer (4096 bytes).

system address
: The address of the following string:

  ```
  SYSTEM(SYSTEM01)
  ```

  In this string, SYSTEM is a keyword and SYSTEM01 is the system name where you want your request to run. The name should be in ASCII format, left-justified, from 1 to 8 bytes long, and padded with spaces. If you want the default system, do one of the following:

  - Set this address to zeros.
  - Set the address to point to the string SYSTEM().
  - Set the address to point to an 8-byte string of hexadecimal zeros.

  Once you supply a system name and that transfer request opens, that name cannot change. If you want a different system, the application program must do one of the following:

  - Open a new transfer request specifying a different buffer.
  - Close and end the current transfer request, then open a new transfer request specifying the same buffer.

EHNTFSTF.DLL returns the following output parameters (passed by reference) and sets register AX:

register AX — Return code in register AX. Refer to "OS/2 Return Code Summary" on page 5-59 for a summary of the return codes set by EHNTFSTF.DLL.

return address — Address of error text if the return code in register AX is unsuccessful and the return length parameter is not zero.

return length — Contains one of the following:

- A count of the number of templates that can be retrieved if the return code in register AX for the open was zero. Templates can be retrieved for the SELECT, EXTRACT, and REPLACE transfer requests.
- The length of the error message text returned from the system if the return code in register AX contains X'3000', X'3100', or X'0400'.

return code second — One of the following, depending on the return code in register AX:

- A zero if the return code in AX is zero.
- A count of the number of templates that can be retrieved if the return code in register AX is X'0400' (a warning message) and the request is a REPLACE transfer request. In this case, the return length parameter is the length of the warning message where it would normally be a count of the number of templates.
- An OS/2 error code if the return code in register AX is X'5400'.
- The position of the first untranslatable byte in the SELECT, EXTRACT, or REPLACE transfer request if the return code in register AX is X'2002'.
- A value that refers to the position in the transfer request where the error was detected if the return code in register AX is X'3000'.

After this function is successfully performed, the transfer request is considered open and a transfer request buffer is associated with the request. Now your application program can request or send data.

Note: For a REPLACE transfer request, if the file member contains data, the return code for the open is X'0400'. This is a warning that, if you continue, the data in the file member will be replaced. If you continue and *send records* (function parameter = 06), the data is replaced. If you do not continue and you *close the transfer request* (function parameter = 04), data in the file remains intact.

Retrieve the Templates Function (Function=02)

This function transfers a group of records, called templates, to your application program. This group of records contains a description of the fields within the system database file for SELECT and REPLACE transfer requests and a description of what the lists look like for an EXTRACT transfer request. Your application program saves the information in the templates for interpreting the actual records that are returned.

Each template contains the name, length, data type, and the maximum size of the number for a field that your application program requests. For binary, zoned decimal, or packed decimal data, the maximum size is expressed in decimal digits. For character or hexadecimal data, the decimal digits are always hexadecimal zero.

A template has the following format:

Byte 1 (field type):

00	Hexadecimal
01	Binary
02	EBCDIC character
03	EBCDIC zoned decimal
04	EBCDIC packed decimal
05	Reserved
06	IGC open
07	IGC only
08	IGC either
0B	Time
0C	Date
0D	Timestamp
0F	Graphic

Bytes 2 and 3 (field length):

- For hexadecimal and character data types, these bytes contain the total length (in bytes) of the field.
- For binary, packed and zoned data types, byte 2 contains the length (in bytes), and byte 3 contains the number of decimal positions to the right of the decimal point.

Bytes 4 through 33 (field name):

Name of the field as defined by the record format for the AS/400 file, left-adjusted, and padded with spaces.

Note: If this is a JOIN request, the file qualifier precedes the field name. For example, Tx.FIELD, where x is 1 through 32.

Byte 34 (digits):

For binary, packed, and zoned data types, this byte contains the maximum number of digits that the field can hold. For example, if the maximum number of digits is 3, the maximum value of the field cannot be greater than 999.

Byte 35

B'1xxxxxxx' Null-capable

B'x1xxxxxx' Variable length

B'xx1xxxxx' Reserved

B'xxx1xxxx' Reserved

B'xxxx1xxx' Reserved

B'xxxxx1xx' Reserved

B'xxxxxx1x' Reserved

B'xxxxxxx1' Reserved

Bytes 36 through 40:

Reserved.

Use the following values for the input parameters:

function — The code for the function you want to perform. Set this parameter to 02 to retrieve templates.

buffer address — Address of the active transfer request buffer.

EHNTFSTF.DLL returns the following output parameters (passed by reference) and sets register AX:

register AX — Return code in register AX. Refer to "OS/2 Return Code Summary" on page 5-59 for a summary of the return codes set by EHNTFSTF.DLL.

return address — One of the following:

- The address in the transfer request buffer of the template if the return code in register AX is zero.
- The address in the transfer request buffer of the error message text if the return code in register AX is not zero and the return length parameter is not zero.

return length — One of the following depending on the return code in register AX:

- The length, in bytes, of the retrieved template if the return code in AX is zero. This length is 40 (X'28').
- The length of the error message text returned from the system if the return code in register AX contains X'3000' or X'3100'.

return code second — One of the following depending on the return code in register AX:

- A zero, if the return code in AX is zero.
- An OS/2 error code if the return code in register AX is X'5400'.

One template returns for each application program request function=02. After all templates are returned, further requests for templates result in an end-of-file return code (register AX=X'1FFF'). You receive one template for each field you specified in the SELECT or REPLACE statement.

Retrieve the Records Function (Function=03)

This function retrieves records from the system and allows you to specify the format in which your application program receives records when you transfer data from the system to the personal computer.

Use the following values for the input parameters:

function — The code for the function you want to perform. Set this parameter to 03 to retrieve records.

buffer address
: Address of the active transfer request buffer.

conversion
: Controls whether the records change to a format compatible with personal computer ASCII format or are left in system EBCDIC format. Specify one of the following:

  X'00'
  : The records do not change and are received from the system exactly as they appear. Null fields contain zeros for numeric and blanks for character and hexadecimal fields. Variable-length fields are padded with blanks or zeros.

  X'01'
  : The records change in the same way the interactive or automatic transfer function program changes records when writing to a DOS random file.

  X'11'
  : The records change in the same way the interactive or automatic transfer function programs change records when writing to a DOS random type 2 file.

  X'02'
  : The records change to DOS random file format, with the following exceptions:

    - When an untranslatable EBCDIC zoned decimal byte is found, the byte is converted to X'3F'.
    - When an untranslatable EBCDIC packed decimal byte is found, the byte is converted to X'FF'.

  X'12'
  : The records change to DOS random type 2 file format, with the following exceptions:

    - When an untranslatable EBCDIC zoned decimal byte is found, the byte is changed to X'3F'.
    - When an untranslatable EBCDIC packed decimal byte is found, the byte is changed to X'FF'.

  X'80'
  : The records are not converted, and information about null and variable-length fields is retained.

    When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

    Two additional fields are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

X'81' The records change in the same way the interactive or automatic transfer function program changes records when writing to a DOS random file.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

X'82' The records change to DOS random file format, with the following exceptions:

- When an untranslatable EBCDIC zoned decimal byte is found, the byte changes to an X'3F'.
- When an untranslatable EBCDIC packed decimal byte is found, the byte changes to an X'FF'.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

When null-capable fields are specified in the format of the file, the record returned is a fixed size. This fixed size has two additional fields that are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

X'91' The records change in the same way the interactive or automatic transfer function programs change records when writing to a DOS random type 2 file.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2

bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

When null-capable fields are specified in the format of the file, the record returned is a fixed size. This fixed size has two additional fields that are included in the 4096-byte record length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

X'92' The records change to DOS random type 2 file format, with the following exceptions:

- When an untranslatable EBCDIC zoned decimal byte is found, the byte is changed to an X'3F'.
- When an untranslatable EBCDIC packed decimal byte is found, the byte is changed to an X'FF'.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

If you specify something other than the above values, the records change as if you had specified X'01'.

EHNTFSTF.DLL returns the following output parameters (passed by reference) and sets register AX:

register AX — Return code in register AX. Refer to "OS/2 Return Code Summary" on page 5-59 for a summary of the return codes set by EHNTFSTF.DLL.

return address — One of the following:

- The address in the transfer request buffer of the returned record if the return code in register AX is zero.

- The address in the transfer request buffer of the error message text if the return code in register AX is not zero and the return length parameter is not zero.

return length — One of the following, depending on the return code in register AX:

- The length, in bytes, of the retrieved record if the return code in register AX is zero.
- The length of the error message text returned from the system if the return code in register AX contains X'3000' or X'3100'.

return code second — One of the following depending on the return code in register AX:

- Zero, if the return code in AX is zero.
- Number of untranslatable characters encountered in the record if the return code in register AX is X'0300'.
- OS/2 error code if the return code in register AX is X'5400'.

After all records for the transfer request are returned, the next request for a record results in an end-of-file return code (register AX=X'1FFF'), and the transfer request is automatically closed.

Close the Transfer Request Function (Function=04)

This function tells the system to close the file associated with the transfer request for this process. This function should be used after your application program has finished sending or retrieving records to or from a particular file.

Use the following values for the input parameters:

function — The code for the function you want to perform. Set this parameter to 04 to close the transfer request.

buffer address — Address of the active transfer request buffer.

EHNTFSTF.DLL returns the following output parameters (passed by reference) and sets register AX:

register AX — Return code in register AX. Refer to "OS/2 Return Code Summary" on page 5-59 for a summary of the return codes set by EHNTFSTF.DLL.

return address — Address of the error text if the return code in register AX is not zero and the return length parameter is not zero.

return length — Length of the error message text returned from the system if the return code in register AX contains X'3000' or X'3100'.

return code second — A 2-byte value which contains an OS/2 error code if the return code in register AX is X'5400', otherwise it should be zero.

If the request is successful, the transfer request is closed. No more data can transfer.

You can also close a transfer request by retrieving all records (function=03) from it until an end-of-file condition is returned, or by opening another transfer request using the transfer request buffer associated with the previous transfer request.

Note: If another transfer request is opened using the same transfer request buffer, the transfer request must be of the same type as the previous transfer request. For example, if a "SELECT . . ." transfer request was opened previously, another SELECT transfer request must be opened (an "EXTRACT . . ." is also allowed). If a "REPLACE . . ." transfer request opened previously, another REPLACE transfer request must be opened (an "EXTRACT . . ." is also allowed).

End All Transfer Request Conversations Function (Function=05)

This function closes all transfer requests immediately and ends any current requests for data.

Note: In most cases, your application programs will probably be working with one session at a time. In these cases, use function 07, which ends one transfer request conversation at a time. Use function 05 if you have multiple conversations active and you need to end all of them.

Use the following input parameter:

function — The code for the function you want to perform. Set this parameter to 05 to end all transfer request conversations.

EHNTFSTF.DLL returns the following output parameters (passed by reference) and sets register AX:

register AX — Return code in register AX. Refer to "OS/2 Return Code Summary" on page 5-59 for a summary of the return codes set by EHNTFSTF.DLL.

return address — Address of error text if the return code in register AX is not zero and the return length parameter is not zero.

return length — The length of the error message text returned from the system if the return code in register AX contains X'3000' or X'3100'.

return code second — A 2-byte value which contains an OS/2 error code if the return code in register AX is X'5400', otherwise it should be zero.

Send Records Function (Function=06)

This function sends records to the system and allows you to specify the format in which your application program sends records when you transfer data from the personal computer to the system.

Use the following values for the input parameters:

function — The code for the function you want to perform. Set this parameter to 06 to send records.

buffer address — The address of the transfer request buffer associated with the active transfer request.

The record that your application program sends to the system

file must be within this buffer and start at the first byte. The fields within the record to be transferred must be in the same order as, and compatible with, the format of the system file, as described by the templates returned when the transfer request was opened. For example, assume the templates specified that the system file consisted of two fields, and that the first field was an 8-byte character field and the second was a 4-byte packed field. Any record you transfer to the AS/400 system should be formatted the same way in the transfer request buffer.

request length — Length of the record to send.

conversion — Controls whether the data is translated from DOS random format to system format. For example, if the transfer request buffer contains an 8-byte ASCII character field, EHNTFSTF.DLL would translate this to an 8-byte EBCDIC field before sending the record to the system.

Specify one of the following:

X'00' — The records do not change and are sent to the system exactly as they appear on the personal computer. EHNTFSTF.DLL still checks for untranslatable data in zoned decimal and packed decimal fields.

If the AS/400 system is running using Version 2 Release 2 Modification 0, the null map and the 2-byte variable length for each variable-length field are added to each record. This ensures compatibility with the AS/400 database format.

X'01' — The records are translated from PC DOS random format to system format.

If the AS/400 system is running using Version 2 Release 2 Modification 0, the null map and the 2-byte variable length for each variable-length field are added to each record. This ensures compatibility with the AS/400 database format.

X'11' — The records are translated from PC DOS random type 2 format to system format.

If the AS/400 system is running using Version 2 Release 2 Modification 0, the null map and the 2-byte variable length for each variable-length field are added to each record. This ensures compatibility with the AS/400 database format.

X'80' — The records are not converted, and information about null and variable-length fields is retained.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field will contain irrelevant data).

There are two additional fields that are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not

contain a null value, or X'F1', which indicates the field contains a null value.

If the AS/400 system is running using Version 2 Release 1.1 or any other previous release or version, the record is converted to the fixed-length fields without a null map. This conversion ensures compatibility with the AS/400 database variable-length record and null value support.

X'81' The records are translated from personal computer DOS random format to the system format.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

If the AS/400 system is running using Version 2 Release 1.1 or any other previous release or version, the record is converted to the fixed-length fields without a null map. This conversion ensures compatibility with the AS/400 database variable-length record and null value support.

X'91' The records are translated from personal computer DOS random type 2 format to the system format.

When variable-length fields are specified in the format of the file, each variable-length field has 2 additional bytes associated with the field. These 2 bytes (preceding the variable-length field) indicate the length of the significant data contained in the field (the remainder of the field contains irrelevant data).

Two additional fields are included in the 4096-byte length restriction. These additional fields are a 2-byte offset to the null map and the null map itself, which is a field containing 1 byte for each field in the data portion of the record. Each byte can have a value of X'F0', which indicates the field does not contain a null value, or X'F1', which indicates the field contains a null value.

If the AS/400 system is running using Version 2 Release 1.1 or any other previous release or version, the record is converted to the fixed-length fields without a null map. This conversion ensures compatibility with the AS/400 database variable-length record and null value support.

EHNTFSTF.DLL returns the following output parameters (passed by reference) and sets register AX:

register AX

Return code in register AX. Refer to "OS/2 Return Code Summary" on page 5-59 for a summary of the return codes set by EHNTFSTF.DLL.

return address

Address of error text if the return code in register AX is not zero and the return length parameter is not zero.

return length

The length of the error message text returned from the system if the return code in register AX contains X'3000' or X'3100'.

return code second

A 2-byte value which contains one of the following:

- A zero if the return code in register AX is zero.
- The number of untranslatable bytes of character data if the return code in register AX contains X'0302'.
- The position of the first untranslatable byte in a zoned decimal or packed decimal field if the return code in register AX is X'2003'.
- The position of the field in which a digit range error occurred if the return code in register AX is X'2004'.
- An OS/2 error code if the return code in register AX is X'5400'.

Before sending a record to the system file, EHNTFSTF.DLL checks the record for untranslatable data. If the value of the conversion parameter is X'00' (no translation), EHNTFSTF.DLL only checks for untranslatable data in zoned decimal and packed decimal fields. If the value of the conversion parameter is X'01' or X'11' (translation), all fields are checked for untranslatable data.

If EHNTFSTF.DLL finds untranslatable character data in the record, the record transfers to the system after the following occurs:

- The untranslatable bytes are set to X'00'.
- The buffer length parameter is set to the number of untranslatable bytes in the record.
- Register AX is set to X'0302', indicating a warning (untranslatable character data found).

If EHNTFSTF.DLL finds untranslatable zoned decimal or packed decimal data in the record, the record is *not* sent to the system. The following occurs:

- The buffer length parameter is set to the position of the first untranslatable byte.
- Register AX is set to X'2003' (untranslatable numeric data found).

If EHNTFSTF.DLL does not find any untranslatable zoned decimal or packed decimal data in the record, it checks all numeric data (binary, zoned decimal, and packed decimal) to see whether the number of digits fits the range specified by the template for that field.

As soon as EHNTFSTF.DLL detects a digit range error, processing stops. EHNTFSTF.DLL returns the position of the field in which the digit error occurred in the buffer length parameter and sets register AX to X'2004' (numeric data does not fit into range specified by digits).

For example, assume that a packed decimal field contains X'5103' and the template for that field indicates that the number of digits is 2. EHNTFSTF.DLL would not send the record because the actual number contains 3 digits, 510 (remember that the last half-byte is the sign).

You can send one record to the system file each time you use this function. After you finish sending all records, issue function 04 to close the transfer request. The last group of records will be sent to the system and the system database file is closed.

Note: EHNTFSTF.DLL and the OS/2 Communications Manager send data records in groups to the system. If an error occurs, the actual number of records that arrive on the system cannot be predicted. For example, if you received an error return code while sending record 100, only 50 records may actually be in the system file or library member.

End the Transfer Request Conversation Function (Function=07)

This function tells the system to close the file associated with the transfer request for this process. This function should be used after your application program has finished sending or receiving records to or from a particular file. It also ends communication between the system and the personal computer for that transfer request. If the transfer request associated with this buffer is open, it automatically closes.

Note: Ending the transfer request is not the same as closing the transfer request. When you end a transfer request, the router conversation ends. When you close a transfer request, the router conversation remains active.

Use this function when you close a transfer request and do not plan to open another transfer request using the same buffer. This releases the personal computer and system resources that supported the communication for that transfer request.

Use the following values for the input parameters:

function — The code for the function you want to perform. Set this parameter to 07 to end the transfer request conversation.

buffer address — Address of the active transfer request buffer.

EHNTFSTF.DLL returns the following output parameters (passed by reference) and sets register AX:

register AX — Return code in register AX. Refer to "OS/2 Return Code Summary" on page 5-59 for a summary of the return codes set by EHNTFSTF.DLL.

return address — Address of error text if the return code in register AX is not zero and the return length parameter is not zero.

return length — The length of the error message text returned from the system if the return code in register AX contains X'3000' or X'3100'.

return code second — A 2-byte value which contains an OS/2 error code if the return code in register AX is X'5400', otherwise it should be zero.

OS/2 Application Program Interface Considerations

Consider the following when using the transfer function personal computer application program interface with OS/2:

- Generally, EHNTFSTF.DLL does not detect whether the data definitions on the system are incorrect or if the data in the file itself is incorrect or damaged. However, EHNTFSTF.DLL does check data sent to the system for errors such as untranslatable characters or too many digits in the field.
- If the application program already knows the format of the records being transferred, it can bypass retrieving templates and go directly from opening a transfer request to requesting or sending the records. However, it cannot go the other way. Once a request is made to retrieve or send records, the application program cannot retrieve templates without opening the transfer request again.

OS/2 Return Code Summary

Table 5-4 contains a summary of the return codes set by EHNTFSTF.DLL.

Table 5-4 (Page 1 of 2). EHNTFSTF.DLL Return Code Summary

Return Code	Description
0000	The previous operation was successful.
0103	Transfer request is already ended.
0300	Untranslatable data found in transferred record.
0302	Untranslatable data in record to be transferred to AS/400.
0400	Warning detected by the AS/400 system.
1FFF	End of file.
2001	Transfer request length not valid.
2002	Transfer request character cannot be translated.
2003	Record not transferred to the AS/400 because of untranslatable numeric data.
2004	Numeric data does not fit into range specified by digits.
2005	Record length given by your application program is incorrect.
2006	Program stack is not large enough.
2007	Null map offset given by your application program is not correct.
2008	Total length of record exceeds 4096 bytes.
2010	Transfer request not opened. One of the causes could be that the 4096 KB buffer address has changed for an active transfer request.
2011	Templates cannot be retrieved now.
2012	Cannot get records on a Personal Computer-to-AS/400 request.

Table 5-4 (Page 2 of 2). EHNTFSTF.DLL Return Code Summary

Return Code	Description
2013	Cannot send records on a AS/400-to-Personal Computer request.
2020	Incorrect function requested.
2021	Incorrect buffer length.
2022	Incorrect buffer address.
2023	Transfer request buffer overlaps a previous transfer request buffer.
2024	Incorrect address specified for the buffer address parameter.
2025	Incorrect address specified for the return address parameter.
2026	Incorrect address specified for the return length parameter.
2027	Incorrect address specified for the secondary return code parameter.
2028	Incorrect address specified for the system address parameter.
2030	Options statement not supported on this system.
2031	Options not fully supported on this system.
2032	Conversation format not allowed with STYLE(NEW) in effect.
2033	STYLE keyword not supported on this system.
3000	Error detected by AS/400 — statement offset provided.
3100	Error detected by AS/400 — no statement offset provided.
5042	PC Support router has not been started.
5048	Connection failed to system &&&&&&&&.
5050	System name is incorrect or inactive.
5052	System &&&&&&&&& &&&&&&&&&& &&&&&&&&&&&& program not found.
5054	System &&&&&&&&& &&&&&&&&&& &&&&&&&&&&&& program ended unexpectedly.
5056	Security error occurred for system &&&&&&&&.
5057	Communications Manager not available.
5058	System &&&&&&&& is not a supported system.
5059	Local Name has not been started.
5060	Cannot continue: &&&&&&&& must be Release &.& Level &&.
5062	Contact with system &&&&&&&& ended.
5063	Too many Communications Manager programs are active.
5064	Contact with system &&&&&&&& temporarily interrupted.
5066	Resource failure on system &&&&&&&&.
5067	Conversation with &&&&&&&& was ended unexpectedly.
5071	Insufficient memory to perform function.
5080	PC memory cannot be used.
5400	Operating system returned error code &&&&.
9999	Undefined error. Call your IBM service representative.

OS/2 Program Example

The following pages contain a sample program written in PASCAL for the OS/2 operating system. The program makes far calls to the EHNTFSTF dynamic link unit. This program is an example of how you can use the application program interface to create a transfer request. It displays the names of customers for whom accounts receivable are 90 days overdue. The program then adds the overdue amounts to produce the total amount due.

This example does not show all programming considerations or techniques. Review the example before you begin application design and coding.

```
Program   Sample(Output);

Const
  TReq     = 'Select LSTNAM,BALDUE from QIWS/QCUSTCDT where BALDUE > 0';
                                    !Transfer request statement
  M1       = 'An error occurred on open';
  M2       = 'An error occurred while getting templates';
  M3       = 'An error occurred while getting records';
  EOF      = #1FFF;                 !End of File return code

Type
  Buffer   = Array[1..4096] of Char;  !Buffer needed by EHNTFSTF
  RCType   = Record                   !Return codes from EHNTFSTF
    Main     : Word;                  !Primary return code
    Second   : Word;                  !Secondary return code
    DataLen  : Integer;               !Data or message length
    End;

Var
  TRLen     : Integer;                !Transfer request length
  Buf       : Buffer;                 !Transfer request buffer
  Start     : Integer;                !Starting position of data in
                                      !buffer
  RC        : RCType;                 !Return codes from EHNTFSTF
  Name      : Lstring(255);           !Customer name
  Balance   : Real;                   !Balance due
  Total     : Real;                   !Balance due total
  NameLen   : Integer;                !Length of name field
  BalLen    : Integer;                !Length of Balance due field
  I         : Integer;                !Index counter
```

Figure 5-10 (Part 1 of 5). Application Program Interface Program Example for OS/2 Users

```
!-------------------------------------------------------------
! Application Program Interface (EHNTFSTF.DLL)
!-------------------------------------------------------------
Function Ehntfstf (Function_ID    : Byte;     !Function code
                   Buffer_Address : Adsmem;   !Buffer address
                   Buffer_Length  : Integer;  !Buffer length
                   Request_Length : Integer;  !Transfer Request length
                   System_Address : Adsmem;   !System Name address
                   Conversion     : Byte;     !Translation Type
              Vars Return_Address : Adsmem;   !Address of returned data
              Vars Return_Length  : Integer;  !Length of returned data
              Vars Ret_Code_Second: Word   )  !Secondary Return code
                                  : Word;     !Primary Return code
                                    Extern;

!-------------------------------------------------------------
! Call the Application Program Interface (EHNTFSTF.DLL)
!-------------------------------------------------------------
Procedure Request (Op      : Byte;    !Operation code
                   TRLen   : Integer; !Transfer request length
               Var Buf     : Buffer;  !Transfer request buffer
               Var Start   : Integer; !Starting data position
               Var RetCode : RCType); !Return codes

Var
  Buf_Adr      : Adsmem;              !Pointer to Transfer Request
                                      !buffer
  Buf_Len      : Integer;             !Transfer Request buffer length
  Sys_Adr      : Adsmem;              !Pointer to System Name
  Rtn_Data_Adr : Adsmem;              !Address of returned data
  Rtn_Len      : Integer;             !Length of data returned
  Rtn_Code_Sec : Word;                !Secondary return code


Begin

  Buf_Adr := Ads Buf;                 !Get address of Transfer Request
                                      !buffer
  Buf_Len := Upper (Buf);             !Set Transfer Request buffer
                                      !length
  Sys_Adr.S := 0;                     !Set System Address to zero
  Sys_Adr.R := 0;                     !(Causes default system to be used)

  Retcode.main := Ehntfstf(Op,Buf_Adr,Buf_Len,TRLen,Sys_Adr,1,
                           Rtn_Data_Adr,Rtn_Len,Rtn_Code_Sec);

  Retcode.Second := Rtn_Code_Sec;     !Put Secondary RC in structure
  Retcode.Datalen := Rtn_Len;         !Put data length in structure
  Start := Ord(Rtn_Data_Adr.R - Buf_Adr.R + 1); !Calculate Start of Data
                                                !in Transfer Request buffer
End;
```

Figure 5-10 (Part 2 of 5). Application Program Interface Program Example for OS/2 Users

```
!-------------------------------------------------------------
! Convert a Zoned decimal field to a real number
!-------------------------------------------------------------
Procedure ZoneToReal (Pos  : Integer; !Starting buffer position
                      Len  : Integer; !Field length
                  Var Num  : Real);   !Resulting number

Var
  C         : Integer;                !Value of current buffer
                                      !character
  Scale     : Real;                   !Power of 10 for the current
                                      !digit

Begin
  Scale := 0.01;                      !The last digit is the cents
                                      !position
  Num   := 0;                         !Initialize the number to 0
  For I := Len DownTo 1 Do            !Work backward toward start
                                      !of field
    Begin
      C := Ord(Buf[Pos+I-1]);         !Get the byte from the buffer
      Num := Num + (C Mod 16)*Scale;  !Convert the digit
      Scale := Scale * 10;            !Set the scale factor
    End;
End;

!-------------------------------------------------------------
! Get the templates for each field
!-------------------------------------------------------------
Procedure GetTemplates;

Begin
  Request (2,TRLen,Buf,Start,RC);     !Get customer name template
  If RC.Main < EOF Then               !Check for an error
    Begin
      NameLen := Ord(Buf[Start+1]);   !Get length of name (integer)
      Name.Len := Wrd(Buf[Start+1]);  !Get length of name (string)
      Request (2,TRLen,Buf,Start,RC); !Get Balance due template
      If RC.Main < EOF Then           !Check for an error
        BalLen := Ord(Buf[Start+1])   !Get length of Balance due
    End;
End;
```

Figure 5-10 (Part 3 of 5). Application Program Interface Program Example for OS/2 Users

```
!-------------------------------------------------------------------
! Get the data records
!-------------------------------------------------------------------
Procedure GetRecords;

Begin
  Repeat
    Request (3,TRLen,Buf,Start,RC);   !Get the next record
    If RC.Main < EOF Then             !Check for error or end of
                                      !file
      Begin
        For I := 1 To NameLen Do      !Get the customer name
          Name[I] := Buf[Start+I-1];
        ZoneToReal (Start+NameLen,BalLen,Balance);
                                      !Get the Balance due
        Total := Total + Balance;     !Total the Balance due
        WriteLn (Name,Balance:Ballen+11:2) !Print the record
      End;
  Until RC.Main >= EOF;
End;

!-------------------------------------------------------------------
! Display error status
!-------------------------------------------------------------------
Procedure ErrorMsg;

Begin
  WriteLn ('Main return code = ',RC.Main:4:16);
  WriteLn ('Secondary return code = ',RC.Second);
  If RC.DataLen > 0 Then
    Begin
      Write ('Message = ');
      For I := 1 to RC.DataLen Do Write (Buf[Start+I-1]);
      WriteLn
    End;
End;
```

Figure 5-10 (Part 4 of 5). Application Program Interface Program Example for OS/2 Users

```
!-------------------------------------------------------------
! Main procedure
!-------------------------------------------------------------
Begin
  TRLen := Upper (TReq);              !Get length of statement
  UnPack (TReq,Buf,1);                !Put statement into buffer
  Request (1,TRLen,Buf,Start,RC);     !Open the transfer request
  If RC.Main > 0 Then WriteLn (M1)    !Check for an error
  Else Begin
    GetTemplates;                     !Get the field templates
    If RC.Main > EOF Then             !Check for an error
      WriteLn (M2)
    Else Begin
      Write ('Customer',' ':NameLen-8); !Display column headings
      Write ('Balance Due':BalLen+11);
      WriteLn; WriteLn;
      GetRecords;                     !Get the records
      If RC.Main > EOF Then           !Check for an error
        WriteLn (M3)
      Else Begin
        WriteLn;
        Write ('Total',' ':Namelen-5);
        WriteLn (Total:BalLen+11:2)   !Display the total
      End;
    End;
  End;
  If RC.Main <> EOF Then
    ErrorMsg;                         !Display error status
  Request (7,TRLen,Buf,Start,RC)      !Clean up
End.
```

Figure 5-10 (Part 5 of 5). Application Program Interface Program Example for OS/2 Users

When this program runs, it displays the data records shown in Figure 5-11.

```
Customer                    Balance Due

Henning                           37.00
Jones                            100.00
Vine                             439.00
Johnson                         3987.50
Stevens                           58.75
Alison                            10.00
Doe                              250.00
Williams                          25.00
Lee                              489.50
Abraham                          500.00

Total                           5896.75
```

Figure 5-11. Resulting Data Records for OS/2 Users

Chapter 6. Work Station Function Low-Level Application Program Interface

This chapter contains information on the assembler language application program interface (API) for the work station function. It includes characteristics of the API and describes the API service requests. It introduces and supplies procedures for using the API available with the work station function of PC Support. The API is based on the control program interface for the IBM 3270 personal computer.

The assembler language API for the work station function allows user-written programs to manipulate the characteristics and the operation of the work station function sessions.

Work Station Function Overview

An API is a programming language interface that allows users to communicate with an IBM system control program or licensed program.

The work station function API (WSF API) allows an application program running in a DOS session to communicate with the host system. The application programmer can use the WSF API to send keystrokes, or copy data, to the work station function EBCDIC (extended binary-coded decimal interchange code) buffer. The EBCDIC buffer is the host display session screen buffer.

Note: This API is meant to be used for logical terminal control (control of the characteristics and operation of display sessions). It is not meant to be used to transfer large amounts of data from the personal computer to the host system, since other programs in PC Support supply functions for bulk data transfer.

The application program starts all interaction between the application program and the work station function. The application program sets up the parameters and registers for a specific service and calls the API using the interrupt vector (X'7A', or the same value specified when loading the WSF API). The interrupt handler starts the work station function that processes the specified service (see Figure 6-1).

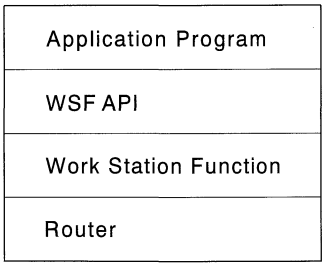


Figure 6-1. Work Station Function Architecture

Installation of the Work Station Function API

The WSF API program WSFAPI.EXE resides on the shared folders QIWSFLR and QIWSFLR2. To use the API, load the work station function, and then load the WSFAPI.EXE program.

The hardware and software requirements for the host system and the personal computer are the same as the requirements defined for the work station function, except for the amount of storage needed on the personal computer. In addition to the storage requirements defined for the work station function, the personal computer requires an additional 10KB (KB equals 1024 bytes) of storage to load the WSF API.

To load the WSFAPI.EXE program, use the following command:

WSFAPI [?] [/I:xx] [/Z]

The following parameters are optional:

Parameter	Description
?	Displays help for the command, but does not load the WSF API.
/:xx	Specifies that interrupt vector X'xx' is to be used to start the WSF API from within the application program. The value **xx** must be a valid hexadecimal number from X'30' to X'FF'. If this parameter is not specified, the default is interrupt vector X'7A'.
/Z	Do not display the WSFAPI copyright banner.

The WSFAPI command loads the WSF API resident in memory and returns control to DOS.

Run the WSFAPI.EXE program only once, since it loads and remains resident in memory. If it is run more than once, message 6203 displays, which indicates that the API is already loaded. See "Work Station Function API Messages" for information on the messages that can display when you run this command.

For convenience, include the WSFAPI.EXE program call in your AUTOEXEC.BAT file after the call to start the work station function (WSF).

Work Station Function API Messages

The application program issues informational and error messages as it runs the WSFAPI command.

Informational Messages

Informational messages appear when you install WSFAPI.EXE. The following message displays the release, version, and level information for the WSF API:

```
PC Support/400
Work Station Function Application Program Interface
(C) Copyright IBM Corp. 1984, 1991.  All rights reserved.
Version 2.0  Release 1.0  Level 00
```

The following message displays when the WSF API is loaded and no errors occur:

```
Work Station Function API loaded
```

Error Messages

The following error messages can appear when you load the WSFAPI.EXE:

5085 Incorrect program option specified

Cause: A value that is not correct was specified when starting a PC Support/400 program.

Recovery: Do one of the following:

- If the option to continue is available, press the Enter key. The program ignores the value that is not correct and continues.
- Press the Esc key to end the program. Check the values of the PC Support/400 command that you entered. Correct the values and start the program again.

6202 Work station function not loaded

Cause: Work station function must be loaded before the work station function API can be loaded.

Recovery: Run the work station function (WSF) command, and then load the work station function API by running the work station function API (WSFAPI) command.

6203 Work station function API is already loaded.

Cause: The work station function API has already been loaded by WSFAPI.EXE.

Recovery: This message is shown for your information only. No action is required.

6205 Incorrect version of work station function running.

Cause: An earlier version of work station function is loaded, but the work station function API runs only with version 1.0 or higher.

Recovery: Start the operating system again, load the correct version of work station function, and then load the work station function API.

Supported Services for the Work Station Function

The WSF API supports the following services:

- Supervisory service
  - Name Resolution
- Session information service
  - Query Session ID
  - Query Session Parameters
  - Query Session Cursor
  - Query Session Status
  - Define Hot-Key Characteristics
- Keyboard service
  - Connect to Keyboard
  - Disconnect from Keyboard
  - Disable Input
  - Enable Input
  - Enable work station function DOS key processing
  - Disable work station function DOS key processing
  - Hot key to work station function
  - Read Input

  - Write Keystroke
- Copy service
  - Copy String
- Operator information area service
  - Read Operator Information Area Group

General Restrictions on Service Requests

All services process synchronously. Once the work station function begins processing a service, it does not return to the application program until the service is complete. On return, the application program examines the return code to determine the results of the service. The description of each service in "API Service Requests" shows which processing is done before the service is complete and control is returned to the application. Other restrictions for each service are also described.

API Service Requests

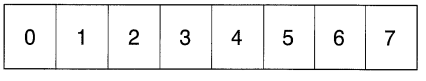
This section describes the correct method for issuing API service requests. Each service supported by the WSF API is explained in detail. This section also shows how to set up the various registers and memory locations needed to successfully complete each service request.

Conventions Used in the API Service Descriptions

The following conventions are used in the descriptions of the API services:

- Hexadecimal numbers are represented in the notation X'nn' for byte values and X'nnnn' for word values.
- Offsets for arranging data used by the API services are given as decimal numbers.
- Bits within a byte are numbered in two different formats:
  - IBM System/360* and System/370* convention

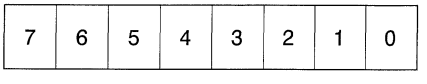
    Bits within a byte are numbered with the high-order (farthest left) bit as bit 0 and the low-order (farthest right) bit as bit 7, as follows:



RV2F954-0

  - Intel 8088 convention

    Bits within a byte are numbered with the low-order (farthest right) bit as bit 0 and the high-order (farthest left) bit as bit 7, as follows:



RV2F955-0

Issuing API Service Requests

Before issuing any service request, the application program determines whether the WSF API is available by doing the following:

- Verifying that the interrupt vector X'7A' (or the interrupt vector specified when loading the WSF API) is not zero. If it is zero, the WSF API function is not available and the application program should not try to use it.
- Issuing a valid API service request and checking that the CH register is X'12' on completion. If the CH register is not X'12', the WSF API is not available and the application program should not try to use it.

If both of the above steps are successful, the WSF API function is available. If either step fails, the API function is not available. The API function can be unavailable for the following reasons:

- The work station function is not loaded.
- The WSFAPI.EXE program is not loaded.

To issue an API service request, load the parameter list and set the registers to the correct values. Then, issue an INT 7AH (or INT XXH, where XX was specified on the WSFAPI command) instruction to signal the WSF API that it has a request to process.

On request completion, verify the CH register first. If the CH register is anything other than X'12', the WSF API function is no longer available.

Note: The ES and DI registers must point to a valid parameter list. If they do not, the results cannot be predicted.

Additional return codes are returned in the CL register and the parameter list.

API Sample Services Flow

This section shows the sequence of API services that the application program issues to perform a function using the WSF API. Initially, a host system program has been started and is waiting for a request from the work station function.

Starting the API Sample Services Flow

To use WSF API, the application program may have to perform some initial steps. Typically, the application program must verify that the WSF API is available, and perform the following services:

- Name Resolution for SESSMGR
- Name Resolution for KEYBOARD
- Name Resolution for COPY
- Name Resolution for OIAM
- Query Session ID for work station function session
- Query Session Parameters of work station function session
- Query Session Status of work station function session
- Connect to Keyboard
- Disable Input from operator or user

Sending Requests

To send information in fields to the host system, the application program should repeat the services in the following list for each request until either a response is received and no longer input-inhibited, or until the session ends:

- Write Keystroke service for each character of a value to be placed into an input field.
- Write Keystroke service to send the Enter keystroke so that data is sent to the host system.

Stopping the API Sample Services Flow

Typically, to return logical display station control to the operator, the application program should perform the following services:

- Enable Input to Operator
- Disconnect from Keyboard

Supervisory Service Requests

This section describes the supervisory service supplied by the API. The results of this service are needed to use the application program services described in other sections of this manual.

Obtaining the Gate Name for the Services Your Application Uses

Use the Name Resolution service to obtain the gate name ID for the specific service that your application program uses.

A **gate** is a grouping of services that performs common function. Each gate is assigned a name when the gate is created. The work station function supplies the following groups of services, or gates:

Gate Name	Meaning
COPY	Copy services
KEYBOARD	Keyboard services
OIAM	Operator information area services
SESSMGR	Session information services

The work station function represents each gate with a 2-byte number called the gate ID. Before your application program can use any of the services in a particular gate, you must obtain the gate ID that the work station function assigned to the application program.

To obtain the correct gate ID, request the name resolution service, specifying an alphanumeric gate name on the request. Work station function returns the gate ID. You must supply the resolved gate ID as input when you request any of the services except name resolution.

Name Resolution Service

Obtains the gate ID of a particular gate name.

Register Values:

On request

AH = X'81'

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

BH = X'07'

CH = X'12'

CL = Return code

DX = Resolved gate ID (numeric representation of the alpha-numeric ASCII gate name)

The contents of registers AX, BL, ES, and DI cannot be predicted.

Parameter List Format: Table 6-1 shows the name resolution service parameter list format.

Table 6-1. Name Resolution Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0–7	8 bytes	Gate name	Unchanged

Parameter Definitions: The gate name must contain ASCII characters and must be padded to the right with spaces if it is less than 8 characters long. See "Obtaining the Gate Name for the Services Your Application Uses" on page 6-6 for a description of possible gate names.

Return Codes: The CH and CL registers contain a return code created by WSF API. System return codes use a function ID of X'12' (found in the CH register). The error code is in the CL register. The error codes that can appear are:

Hex Code	Meaning
00	Successful completion
2E	Specified gate name is not valid
2F	Specified supervisor service number (AH register value) is not valid

Considerations: The name resolution service does not have to be issued more than once for each gate name used by the application program. More than one request for the same gate name always returns the same gate ID during any given work station function operation.

Session Information Service Requests

This section describes the session information services supplied by the API.

The session information services allow your application program to obtain the work station function display session ID, session characteristics, session status, and cursor position.

Requesting the Session Information Services

To request any of the session information services, load the registers and parameter list with the correct values and use the INT 7AH instruction to signal the API that it has a request to process.

Notes:

1. Before your application program can request the session information services, it must request the name resolution service, using SESSMGR as the gate name in the parameter list.
2. You must pad the gate name to the right with spaces if it is less than 8 characters.

Return Codes for the Session Information Services

Each session information service has two return codes: a system return code and a session information services return code. Both types of return codes are 2-byte values made up of a function ID and an error number. The function ID indicates the portion of WSFAPI command in which the error occurred. The error number indicates the specific type of error that occurred. An error number of X'00' always indicates a successful acceptance or completion of the request.

- System return codes

  After your application program requests a session information service, the CH and CL registers contain a return code created by the request processing portion of WSF API. The function ID is in the CH register and the error number is in the CL register. System return codes use a function ID of X'12'. The following error numbers can appear:

Hex Code	Meaning
00	Request accepted
05	Specified name resolution value (DX register value) is not valid
07	Specified reply (BH register value) is not valid
08	Specified wait type (BL register value) is not valid
2F	Specified supervisor service number (AH register value) is not valid
34	Specified service number (AL register value) is not valid
7F	Session not available (started and communicating with host)

  These system return codes apply to all the session information services.

- Session information service return codes

  After a requested session information service is completed, bytes 0 and 1 of the parameter list contain a return code created by the session management portion of the work station function. The function ID is in byte 1, and the error number is in byte 0. Session information services return codes use a function

ID of X'6B'. The error numbers that appear are specific to the service that is requested and are included in the description of each service.

Query Session ID Service

Obtains the session ID of the display or printer session you specify. You can specify a session by its short name, or ask for the IDs of all sessions.

Notes:

1. A short name is a 1-character identifier of a session. Since there is a maximum of 5 work station function sessions, the short names are A, B, C, D, and E. The first session is A, the second is B, and so on.
2. Issue the name resolution service request before issuing this service request.

Register Values:

On request

AH = X'09'

AL = X'01'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for SESSMGR

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-2 shows the Query Session ID service parameter list format.

Table 6-2. Query Session ID Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'6B')
2	1 byte	Option code	Unchanged
3	1 byte	Data code	Unchanged
4	1 word	Offset address of name array	Unchanged
6	1 word	Segment address of name array	Unchanged
8–15	8 bytes	Reserved	Unchanged

Parameter Definitions: The Query Session ID service parameters are described as follows:

- Request parameters

  To obtain the session ID of a session for a specific short name, the following must be true:

  – The option code must be X'01'.
  – The data code must be the 1-character (A through E) ASCII short name of the session.

  To obtain the session ID for all sessions of a specific type, the following must be true:

  – The option code must be X'00'.
  – The data code must be:
    - X'02' for a host system display session
    - X'06' for a host system printer session

Name Array Format: Table 6-3 shows the Name Array Format list.

Table 6-3. Name Array Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Name array length	Unchanged
1	1 byte	Reserved	Number of matching sessions
2	1 byte	Reserved	Short name of session 1
3	1 byte	Reserved	Type of session 1
4	1 byte	Reserved	Session ID of session 1
5	1 byte	Reserved	Reserved
6–13	8 bytes	Reserved	Unchanged

Note:

1 Offset 4 in Table 6-2 on page 6-9 must point to a name array as defined above.

2 The format of name array offsets 2 through 13 must be repeated for as many sessions as can match the Query Session ID service request.

Name Array Parameter Definitions: The name array parameters are described as follows:

- Request parameters

  The name array length is the number of bytes in the name array. The name array must be at least 14 bytes long and no greater than 62 bytes long. In addition, if you are coding this service to obtain the session ID for all sessions of a specific type, the name array must be large enough for all the possible sessions that can be returned for the session type.

- Completion parameters

  – The number of matching sessions contains the number of sessions that matched the request.

  – The session short name is the 1-character uppercase ASCII alphabetic name of the session (A through E).

– The session type is:
  - X'02' for a host system display session
  - X'06' for a host system printer session
– The session ID is the ID that the work station function uses to identify the session. Use the session ID to specify the host session in any subsequent API service requests.

Return Codes: The name array return codes are described as follows:

- System return codes

  Refer to "Return Codes for the Session Information Services" on page 6-8 for a description of the system return codes found in the CH and CL registers.

- Session information services return codes

  Bytes 0 and 1 of Table 6-2 on page 6-9 contain a return code created by the session management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Session information services return codes use a function ID of X'6B'. The error number is in byte 0. You can receive the following session information services error codes for this service:

Hex Code	Meaning
00	Successful completion
06	Specified session ID not in use
09	Session type is not valid
0B	The specified short name is not valid (not A through E)
0C	Byte 0 of the parameter list is not zero on request
0D	Option code is not valid
12	The name array length is not valid
13	The specified short name is not an uppercase ASCII alphabetic character

Considerations: You do not need to issue the Query Session ID service more than once to get the session ID. More than one request always returns the same session ID during the work station function's operation.

Query Session Parameters Service

Obtains the session characteristics of the display or printer session you specify. The application program should use certain values returned by this service when requesting the Copy String service.

Note: Issue the name resolution and Query Session ID service requests before issuing the Query Session Parameters service request.

Register Values:

On request

AH = X'09'

AL = X'02'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for SESSMGR

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-4 shows the Query Session Parameters service parameter list format.

Table 6-4. Query Session Parameters Service List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'6B')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 byte	Reserved	Session type
5	1 byte	Reserved	Reserved
6	1 byte	Reserved	Rows
7	1 byte	Reserved	Columns
8	1 word	Reserved	Reserved
10	1 word	Reserved	Reserved
12	1 byte	Reserved	Device or model ID
13	1 byte	Reserved	Display adapter type
14	1 word	Offset address of area to receive EBCDIC-to-ASCII translation table	Unchanged
16	1 word	Segment address of area to receive EBCDIC-to-ASCII translation table	Unchanged
18	1 word	Offset address of area to receive ASCII-to-EBCDIC translation table	Unchanged
20	1 word	Segment address of area to receive ASCII-to-EBCDIC translation table	Unchanged

Parameter Definitions: The Query Session Parameters service parameters are defined as follows:

- Request parameters
  - The session ID is the ID of the display session whose characteristics you request.
  - An offset and segment address of a 256-byte area may be supplied to receive the contents of the EBCDIC-to-ASCII translation table. If both the offset and segment addresses are X'0000', the EBCDIC-to-ASCII table is not retrieved.
  - An offset and segment address of a 256-byte area may be supplied to receive the contents of the ASCII-to-EBCDIC translation table. If both the offset and segment addresses are X'0000', the ASCII-to-EBCDIC table is not retrieved.
- Completion parameters
  - The session type byte is:
    - X'02' for a display session
    - X'06' for a printer session
  - *Rows* is the hexadecimal number of rows in the session's presentation space. The *Rows* byte returns X'18' (decimal 24) or X'1B' (decimal 27) for the display session and X'00' for the printer session.
  - *Columns* is the hexadecimal number of columns in the session's presentation space. The *Columns* byte returns X'50' (decimal 80) or X'84' (decimal 132) for the work station function display session and X'00' for the work station function printer session.

    Note: The Rows and Columns values will be 27 and 132 only when the host application forces the display session into 27 X 132. Normally, the values are 24 X 80.
  - The model ID for a display session is:
    - X'18' for 3196 Model A1 emulation
    - X'58' for 3197 Model C2 emulation
    - X'49' for 5292 Model 2 emulation
    - X'28' for 3180 Model 2 emulation

    Note: Only the 3180 emulation session supports 27 rows by 132 columns.
  - The device ID for a printer session is:
    - X'2D' for 5219 emulation
    - X'24' for 5224 emulation
    - X'60' for 5256 emulation
    - X'AD' for 3812 emulation
  - The display adapter type indicates whether a monochrome or color adapter is in use:
    - X'00' for monochrome display adapter
    - X'01' for color display adapter

Return Codes: The Query Session Parameters service return codes are defined as follows:

- System return codes

  See "Return Codes for the Session Information Services" on page 6-8 for a description of the system return codes found in the CH and CL registers.

- Session information services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the session management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Session information services return codes use a function ID of X'6B'. You can receive the following session information services error codes for this service:

Hex Code	Meaning
00	Successful completion
02	Session ID is not valid
06	Specified session ID is not in use
0C	Byte 0 of the parameter list is not zero on request

- You do not need to issue the Query Session Parameters service more than once to get the session type and characteristics. More than one request always returns the same session type and characteristics during work station function's operation.
- Use the row and column values returned by the Query Session Parameters service to calculate the maximum offset ending character that you can specify for the Copy String service. For example:

```
MAXIMUM OFFSET = ROWS * COLUMNS - 1
```

Query Session Cursor Service

Obtains the row and column addresses of the specified session's cursor and the cursor type.

Note: Issue the name resolution and Query Session ID service requests before issuing the Query Session Cursor service request.

Register Values:

On request

AH = X'09'

AL = X'0B'

BH = X'80'

BL = X'20'

CX = X'00FF'

DX = Resolved gate ID for SESSMGR

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-5 shows the Query Session Cursor service parameter list format.

Table 6-5. Query Session Cursor Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'6B')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Cursor type
4	1 byte	Reserved	Row address
5	1 byte	Reserved	Column address

Parameter Definitions: The Query Session Cursor service parameters are described as follows:

- Request parameters

  The session ID is the ID of the display session whose cursor information you are requesting.

- Completion parameters
  - The cursor type byte is as shown in Table 6-6.

Table 6-6. Cursor Type Byte

IBM Bit #	Intel Bit #	Bit Description
0	7	Reserved
1	6	Reserved
2	5	Reserved
3	4	Reserved
4	3	Reserved
5	2	Reserved
6	1	Blinking underline cursor
7	0	Blinking box cursor

  - The row address is the address in the session's presentation space representing the cursor's row position. The row address is always a value from 1 to 27 for a display session. This address is always X'00' for a printer session.

– The column address is the address in the session's presentation space representing the cursor's column position. The column address value is always from 1 to 132 for a display session. This address is always X'00' for a printer session.

Return Codes: The Query Session Cursor service return codes are described as follows:

- System return codes

  See "Return Codes for the Session Information Services" on page 6-8 for a description of the system return codes found in the CH and CL registers.

- Session information services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the session management portion of work station function. The function ID is in byte 1, and the error number is in byte 0. Session information services return codes use a function ID of X'6B'. You can receive the following session information services error codes for this service:

Hex Code	Meaning
00	Successful completion
02	Session ID is not valid
06	Specified session ID is not in use
0C	Byte 0 of the parameter list is not zero on request

Query System Level Service

Obtains the system level information for the current release of the WSF API. WSF API maintains an area required to change the release, the program temporary fix (PTF) level, or both. This verb returns the information to the person who called for it.

Note: Issue the name resolution service request before issuing the Query System Level service request.

Register Values:

On request

AH = X'09'

AL = X'0C'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for SESSMGR

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-7 shows the Query System Level service parameter list format.

Table 6-7. Query System Level Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'6B')
2	1 byte	Reserved	Unchanged
3–6	4 bytes	Must be zero	Product ID
7–8	2 bytes	Must be zero	Unchanged
9–10	2 bytes	Must be zero	API version
11–12	2 bytes	Must be zero	API release
13–14	2 bytes	Must be zero	API PTF level
15	1 byte	Must be zero	Reserved
16	1 byte	Must be zero	Control program type
17	1 byte	Must be zero	Reserved
18–19	2 bytes	Must be zero	Reserved

Parameter Definitions: The Query System Level service parameters are described as follows:

- Completion parameters
  - Product ID is the ID of the given product. It is a 4-byte ASCII string. The value is WSF for work station function.
  - API version is a 2-byte ASCII API version number (20 for API version 2.0, 00 for API version 1.0).
  - API release is a 2-byte ASCII API release number (10 for API release 1.0).
  - API PTF level is a 2-byte ASCII API (01 for API level 01) PTF level number.
  - The control program types are as follows:

Types	Meaning
C	3270 personal computer control program
F	5250 work station function
W	3270 work station program

Return Codes: The Query System Level service return codes are described as follows:

- System return codes

  See "Return Codes for the Session Information Services" on page 6-8 for a description of the system return codes found in the CH and CL registers.

- Session information services return codes

Bytes 0 and 1 of the parameter list contain a return code created by the session management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Session information services return codes use a function ID of X'6B'. You can receive the following session information services error codes for this service:

Hex Code	Meaning
00	Successful completion
0C	Byte 0 of the parameter list not zero on request

Query Session Status Service

Determines whether a display or printer session is active and whether the contents of the display session EBCDIC buffer has changed since the last time the work station function changed the presentation space (display adapter buffer). This also indicates whether the Sys Req line or the message line is active. The message line is 25 on a 25x80 display session or line 28 on a 29x132 display session.

Note: Issue the name resolution and Query Session ID service requests before issuing the Query Session Status service request.

Register Values:

On request

AH = X'09'

AL = X'0D'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for SESSMGR

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-8 shows the Query Session Status service parameter list format.

Table 6-8 (Page 1 of 2). Query Station Status Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'6B')
2	1 byte	Session ID	Unchanged

Table 6-8 (Page 2 of 2). Query Station Status Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
3	1 byte	Reserved	Reserved
4	1 byte	Not used	Session status

Parameter Definitions: The Query Session Status service parameters are defined as follows:

- Request parameters

  The session ID is the ID of the display session returned by the Query Session ID service.

- Completion parameters

  The session status byte is defined as shown in Table 6-9.

Table 6-9. Session Status Byte

IBM Bit #	Intel Bit #	Bit Description
0	7	Session activity: • 0 = Inactive session. An inactive session is one that is configured but is not communicating with the host. • 1 = Active session. An active session is one that is configured and is communicating with the host.
1	6	Buffer change status: • 0 = EBCDIC buffer has not been changed. • 1 = EBCDIC buffer has been changed.
2	5	Application program session use: • 0 = The session is not being used by an application program for keyboard services. • 1 = The session is already being used by an application program for keyboard services.
3	4	Sys Req status: • 0 = Sys Req is not active for this session. • 1 = Sys Req is active for this session.
4	3	Message line status: • 0 = There is no message on the message line. The message line is line 25 for a 25x80 display session or line 28 for a 29x132 display session. • 1 = There is a message on the message line. The message line is line 25 for a 25x80 display session or line 28 for a 29x132 display session.
5-7	0-2	Reserved.

Return Codes: The Query Session Status service return codes are defined as follows:

- System return codes

See "Return Codes for the Session Information Services" on page 6-8 for a description of the system return codes found in the CH and CL registers.

- Session information services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the session management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Session information services return codes use a function ID of X'6B'. You can receive the following session information services error codes for this service:

Hex Code	Meaning
00	Successful completion
02	Session ID is not valid
06	Specified session ID is not in use
0C	Byte 0 of the parameter list is not zero on request

Considerations: When using the Query Session Status service, keep in mind the following:

- The session must be active before you can run any of the keyboard, copy, or operator information services.
- An application program can use the EBCDIC buffer change status to determine when the data in the EBCDIC buffer changed. This flag is set when the Copy String service is used to copy data into the 5250 EBCDIC buffer, when the 5250 EBCDIC buffer is changed by the host system, and so on.
- The buffer change status flag is set to 0 after it returns its status. This way the flag always reflects any changes made to the 5250 EBCDIC buffer since the last time the Query Session Status service was run.

Define Hot-Key Characteristics Service

Defines the hot-key characteristics of a session.

Note: Issue the name resolution and Query Session ID service requests before you issue the Define Hot-Key Characteristics service request.

Register Values:

On request

AH = X'09'

AL = X'0F'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for SESSMGR

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-10 shows the Define Hot-Key Characteristics service parameter list format.

Table 6-10. Define Hot-Key Characteristics Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'6B')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 byte	Hot-key control	Unchanged
5	1 byte	Hot-key character row	Unchanged
6	1 byte	Hot-key character column	Unchanged
7	1 byte	Hot-key character	Unchanged

Parameter Definitions: The Define Hot-Key Characteristics service parameters are described as follows:

- Request parameters
  - The session ID is the ID of the display session whose characteristics you are defining.
  - Use the hot-key control flag to specify whether you should make the operator's ability to hot-key into a session effective or ineffective.
    - X'01' makes hot-keying effective.
    - X'00' makes hot-keying ineffective.
  - The work station function automatically goes to the DOS session from a display session when the specified EBCDIC character appears in the selected row and column. Following are the hot-key characteristics:
    - The hot-key character is the ASCII character which, when found at the specified row and column on the display, automatically causes the work station function to switch (hot-key) to the DOS session.
    - The hot-key row is the row number (1–27) that the specified hot-key character must be in to have the work station function start an automatic hot-key to the DOS session.
    - The hot-key column is the number of the column (1–132) in which the specified hot-key character must exist to have the work station function start an automatic hot-key to the DOS session. If you specify zero for the column value, then automatic hot-keying to the DOS session on the specified character stops.

Return Codes: The Define Hot-Key Characteristics service return codes are described as follows:

- System return codes

See "Return Codes for the Session Information Services" on page 6-8 for a description of the system return codes found in the CH and CL registers.

- Session information services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the session management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Session information services return codes use a function ID of X'6B'. You can receive the following session information services error codes for this service:

Hex Code	Meaning
00	Successful completion
02	Session ID is not valid
06	Specified session ID is not in use
0C	Byte 0 of the parameter list is not zero on request
14	Input parameter is not valid

Keyboard Services Requests

This section describes the keyboard services supplied by the API.

The keyboard services allow the application program to write keystroke data to the work station function display session and to stop and start operator input from the keyboard of the work station function display session.

To request any of the keyboard services, load the registers and the parameter list with the correct values and use the INT 7AH instruction to signal the API that it has a request to process.

Note: Before your application program can request the keyboard services, the program must request the name resolution service, using KEYBOARD as the gate name in the parameter list. Your application program must also request the Query Session ID service.

Attention Identifier Keys

A description of attention identifier (AID) keys is found in the description of keyboard services. In general, those keys causing immediate host system interaction are treated as AID keys. The following are AID keys:

- Cmd
- F1 through F24
- Enter
- Clear
- Sys Req
- Attn
- Print (host print)
- Help
- Page Up
- Page Down
- Record Backspace
- Home

Note: You can automatically create an AID key by sending a series of non-AID keys using the Write Keystroke service. For example, the Home key is

usually a non-AID key. However, if the Home key is written using the Write Keystroke service and the cursor was already in the home position, a record backspace indicator is returned to the host system program that is controlling the work station.

If you are entering data into an automatic enter field, the system assumes that you pressed the Enter key after the cursor left the input field because you pressed the Field Exit, Field +, or Field - keys, or entered the farthest right character into the field.

Return Codes for the Keyboard Services

Each keyboard service has two return codes — a system return code and a keyboard services return code. Both types of return codes are 2-byte values made up of a function ID and an error number. The function ID indicates the portion of WSF API in which the error occurred. The error number indicates the specific type of error that occurred. An error number of X'00' always indicates a successful acceptance or completion of the request.

- System return codes

  After your application program has requested a keyboard service, the CH and CL registers contain a return code created by the request processing portion of WSF API. The function ID is in the CH register, and the error number is in the CL register. System return codes use a function ID of X'12'. The following error numbers can appear:

Hex Code	Meaning
00	Request accepted
05	Specified name resolution value (DX register value) is not valid
07	Specified reply (BH register value) is not valid
08	Specified wait type (BL register value) is not valid
2F	Specified supervisor service number (AH register value) is not valid
34	Specified service number (AL register value) is not valid
7F	Session not available (started and communicating with host)

  These system return codes apply to all the keyboard services.

- Keyboard service return codes

  After a requested keyboard service is completed, bytes 0 and 1 of the parameter list contain a return code created by the keyboard management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Keyboard services return codes use a function ID of X'62'. The error numbers that can appear are specific to the service that is requested and are included in the description of each service.

Connect to Keyboard Service

Connects to the display session for keyboard services.

Before your application program issues any other keyboard service, the program must issue the Connect to Keyboard service request.

Note: Issue the name resolution and Query Session ID service requests before issuing the Connect to Keyboard service request.

Register Values:

On request

AH = X'09'

AL = X'01'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for KEYBOARD

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-11 shows the Connect to Keyboard service parameter list format.

Table 6-11. Connect to Keyboard Parameter Service List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'62')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Must be zero	Unchanged
6	1 word	Must be zero	Unchanged
8	1 byte	Must be zero	Unchanged
9	1 byte	Reserved	Reserved

Parameter Definitions: The Connect to Keyboard service parameters are defined as follows:

- Request parameters

  The session ID is the ID of the display session to which you want to connect.

Return Codes: The Connect to Keyboard service return codes are defined as follows:

- System return codes

  See "Return Codes for the Keyboard Services" on page 6-23 for a description of the system return codes found in the CH and CL registers.

- Keyboard services return codes

Bytes 0 and 1 of the parameter list contain a return code created by the keyboard management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Keyboard services return codes use a function ID of X'62'. You can receive the following keyboard services error codes for this service:

Hex Code	Meaning
00	Successful completion
01	Byte 8 of the parameter list is not zero on request
02	Session ID is not valid
04	Session is already connected for keyboard services
0C	Byte 0 of the parameter list is not zero on request
15	Session must be host display session

Considerations: When using the Connect to Keyboard service, keep in mind the following:

- Use the Connect to Keyboard service, in conjunction with the Disconnect from Keyboard service, to serialize access to the keyboard services. This provides protection against two or more application programs using the API at the same time in a multiple application environment.

  If you want this protection, do the following when using the Connect to Keyboard service:

  - Use the Connect to Keyboard service before issuing the first keyboard service request.
  - Verify the return code from the Connect to Keyboard service. If the return code is X'04' (Session already connected), do not issue any other keyboard service until the Connect to Keyboard service is issued again and the return code is X'00' (Successful completion).

Disconnect from Keyboard Service

Disconnects from the display session when you are finished using the keyboard services. This function also enables input if it was previously disabled.

Note: Issue the name resolution, Query Session ID, and Connect to Keyboard service requests before issuing the Disconnect from Keyboard service request.

Register Values:

On request

AH = X'09'

AL = X'02'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for KEYBOARD

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-12 shows the Disconnect from Keyboard service parameter list format.

Table 6-12. Disconnect from Keyboard Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'62')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Must be zero	Unchanged

Parameter Definitions: The Disconnect from Keyboard service parameters are defined as follows:

- Request parameters

  The session ID is the ID of the display session from which you want to disconnect.

Return Codes: The Disconnect from Keyboard service return codes are defined as follows:

- System return codes

  See "Return Codes for the Keyboard Services" on page 6-23 for a description of the system return codes found in the CH and CL registers.

- Keyboard services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the keyboard management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0.

  Keyboard services return codes use a function ID of X'62'. You can receive the following keyboard services error codes for this service:

Hex Code	Meaning
00	Successful completion
02	Session ID is not valid
04	Session is not connected for keyboard services
0C	Byte 0 of the parameter list is not zero on request
15	Session must be host display session

Considerations: When using the Disconnect from Keyboard service, consider the following:

- This service enables operator input to the session if the service was previously made ineffective through a Disable Input service request.
- Used in conjunction with the Connect to Keyboard service, the Disconnect from Keyboard service serializes access to the keyboard services. This can provide protection from two or more application programs using the API at the same time in a multiple application environment.

  If you want such protection, do the following:

  – When you finish using the keyboard services, use the Disconnect from Keyboard service to allow the session to be used by other application programs.

  – Before your application program exits, it must use the Disconnect from Keyboard service for the display session if the application program previously issued a successful Connect to Keyboard service request for the same session. Request the Disconnect from Keyboard service at all error exit points as well as during normal processing.

Read Input Service

Reads intercepted keystroke data destined for the display session. This service returns the ASCII short form of operation name for one keystroke for each request made for a keystroke residing in the keystroke buffer or playback string.

Note: Issue the name resolution, Query Session ID, and Connect to Keyboard service requests before issuing the Read Input service request.

Register Values:

On request

AH = X'09'

AL = X'03'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for KEYBOARD

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-13 on page 6-28 shows the Read Input service parameter list format.

Table 6-13. Read Input Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'62')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Reserved	Unchanged
6	1 byte	Options byte (X'80')	Unchanged
7	1 byte	Reserved	Length of the ASCII/ASCII short form name mentioned in bytes 8–13
8–13	6 bytes	Reserved	ASCII/ASCII short form of operation name

Parameter Definitions: The Read Input service parameters are defined as follows:

- Request parameters
  - The session ID is the ID of the display session from which keystroke data is intercepted.
  - The options byte must have the byte value of X'80', indicating that an ASCII/ASCII short form of the keystroke name is being received.
- Completion parameters
  - ASCII/ASCII short form of operation names are from 1 to 6 bytes long. ASCII short form names start with @.
  - The length of the field is the length of the ASCII code or ASCII short form name. Any bytes left unused remain unchanged.

Return Codes: The Read Input service return codes are defined as follows:

- System return codes

  Refer to "Return Codes for the Keyboard Services" on page 6-23 for a description of the system return codes found in the CH and CL registers.

- Keyboard services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the keyboard management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Keyboard services return codes use a function ID of X'62'. You can receive the following keyboard services error codes for this service:

Hex Code	Meaning
00	Successful completion
01	Option byte value is not valid
02	Session ID is not valid
04	Session is not connected for keyboard services
09	There were no keystrokes to process
0C	Byte 0 of the parameter list is not zero on request
15	Session must be host display session

Write Keystroke Service

Sends keystroke data to the display session just as an operator would type on a 5250 keyboard. This service sends the ASCII/ASCII short form of the operation name of one keystroke for each request made. The work station function processes the keystroke immediately. See "Character Code Table" on page 6-49 for a list of the ASCII/ASCII shortened names that can be sent.

Note: Issue the name resolution, Query Session ID, and Connect to Keyboard service requests before issuing this service request.

Register Values:

On request

AH = X'09'

AL = X'04'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for KEYBOARD

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-14 shows the Write Keystroke service parameter list format.

Table 6-14. Write Keystroke Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'62')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Must be zero	Unchanged
6	1 byte	Options byte (X'90')	Unchanged
7	1 byte	Number of bytes of ASCII/ASCII short form of operation names to send	Number of bytes of ASCII/ASCII short form of operation names sent
8–13	6 bytes	ASCII/ASCII short form of operation name	Unchanged

Parameter Definitions: The Write Keystroke service parameters are defined as follows:

- Request parameters
  - The session ID is the ID of the display session to which you write keystrokes.
  - The options byte value must be X'90', indicating that a single ASCII/ASCII short form of operation name of the keystroke is being sent.
  - The keystroke types and values are defined in "Character Code Table" on page 6-49.
- Completion parameters

  Number of bytes sent is the number of bytes in the ASCII/ASCII short form name that is sent to the session.

Return Codes: The Write Keystroke service return codes are defined as follows:

- System return codes

  See "Return Codes for the Keyboard Services" on page 6-23 for a description of the system return codes found in the CH and CL registers.
- Keyboard services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the keyboard management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Keyboard services return codes use a function ID of X'62'. You can receive the following keyboard services error codes for this service:

Hex Code	Meaning
00	Successful completion
01	Option byte value is not valid
02	Session ID is not valid
04	Session is not connected for keyboard services
0C	Byte 0 of the parameter list is not zero on request
10	Processing stopped because a keystroke value was sent that was not valid
15	Session must be host system display session

Considerations: When using the Write Keystroke service, consider the following:

- The keystroke is sent to the work station function for processing, and control returns to the application program. The application program must use the Read Operator Information Area Group service to determine whether any of the indicators are altered as a result of processing the keystroke.
- A successful completion return code from a Write Keystroke service request to a host system session does not mean that the host system processed the keystroke. It only indicates that the keystroke has been successfully sent to the host system session.

Disable Input Service

Stops operator input to the display session.

Note: Issue the name resolution, Query Session ID, and Connect to Keyboard service requests before you issue the Disable Input service request.

Register Values:

On request

AH = X'09'

AL = X'05'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for KEYBOARD

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-15 shows the Disable Input service parameter list format.

Table 6-15. Disable Input Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'62')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Must be zero	Unchanged

Parameter Definitions: The Disable Input service parameters are defined as follows:

- Request parameters

  The session ID is the ID of the display session whose keyboard you want to stop.

Return Codes: The Disable Input service return codes are defined as follows:

- System return codes

  See "Return Codes for the Keyboard Services" on page 6-23 for a description of the system return codes found in the CH and CL registers.

- Keyboard services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the keyboard management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Keyboard services return codes use a function ID of X'62'. You can receive the following keyboard services error codes for the Disable Input service:

Hex Code	Meaning
00	Successful completion
02	Session ID is not valid
04	Session is not connected for keyboard services
0C	Byte 0 of the parameter list is not zero on request
15	Session must be host system display session

Considerations: Keystrokes typed on the keyboard of a session can become intermixed with the keystroke data that your application program is sending to that session. To prevent this, use the Disable Input service to stop the processing of keystrokes typed on the session keyboard.

When you use this service, any keystrokes you type are put into a keyboard buffer so that the application program can process them through the Read Input service. A maximum of 32 keystrokes is saved. When the buffer is full, a 1-second beep sounds.

Enable Input Service

Starts operator input to the display session again.

Note: Issue the name resolution, Query Session ID, and Connect to Keyboard service requests before you issue the Enable Input service request.

Register Values:

On request

AH = X'09'

AL = X'06'

BH = X'80'

BL = X'20'

CX = X'0000'

DX = Resolved gate ID for KEYBOARD

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-16 on page 6-33 shows the Enable Input service parameter list format.

Table 6-16. Enable Input Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'62')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Must be zero	Unchanged

Parameter Definitions: The Enable Input service parameters are defined as follows:

- Request parameters

  The session ID is the ID of the display whose keyboard you want to start.

Return Codes: The Enable Input service return codes are defined as follows:

- System return codes

  See "Return Codes for the Keyboard Services" on page 6-23 for a description of the system return codes found in the CH and CL registers.

- Keyboard services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the keyboard management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Keyboard services return codes use a function ID of X'62'.

  You can receive the following keyboard services error codes for this service:

Hex Code	Meaning
00	Successful completion
02	Session ID is not valid
04	Session is not connected for keyboard services
0C	Byte 0 of the parameter list is not zero on request
15	Session must be host system display session

Considerations: When using the Enable Input service, the Disconnect from Keyboard service also starts operator input to a display session if the operator input previously was stopped. Any keystrokes left in the keystroke buffer while input was stopped are deleted.

Enable Work Station Function DOS Key processing

After this call has been successfully issued, work station function processes all keys that are typed in the DOS session.

Note: Issue the name resolution, Query Session ID, and Connect to Keyboard service requests before you issue the Enable Work Station Function DOS key processing service request.

Register Values:

On request

AH = X'09'
AL = X'07'
BH = X'80'
BL = X'20'
CX = X'0000'
DX = Resolved gate ID for KEYBOARD
ES = Segment address of the parameter list
DI = Offset address of the parameter list

On completion

CH = X'12'
CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-17 shows the Enable Work Station Function DOS key processing service parameter list format.

Table 6-17. Enable WSF DOS Key Processing Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'62')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Must be zero	Unchanged

Parameter Definitions: The Enable Work Station Function DOS key processing service parameters are defined as follows:

- Request parameters

  The session ID is the ID of the display session to process the DOS keys.

Return Codes: The Enable Work Station Function DOS key processing service return codes are defined as follows:

- System return codes

  See "Return Codes for the Keyboard Services" on page 6-23 for a description of the system return codes found in the CH and CL registers.

- Keyboard services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the keyboard management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Keyboard services return codes use a function ID of X'62'.

  You can receive the following keyboard services error codes for this service:

Hex Code	Meaning
00	Successful completion.
01	Option byte is not valid.
02	Session ID is not valid.
04	Session is not connected for keyboard services.
0C	Byte 0 of the parameter list is not zero on request.
15	Session must be host system display session.
16	Work station function DOS key processing is active for another session.

Disable Work Station Function DOS Key Processing

After this call has been successfully issued, the DOS application processes the DOS keys. (Work station function stops processing the keys.)

Note: Issue the name resolution, Query Session ID, and Connect to Keyboard service requests before you issue the request to disable the work station function DOS key processing service.

Register Values:

On request

AH = X'09'
AL = X'08'
BH = X'80'
BL = X'20'
CX = X'0000'
DX = Resolved gate ID for KEYBOARD
ES = Segment address of the parameter list
DI = Offset address of the parameter list

On completion

CH = X'12'
CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-18 shows the parameter list for the disable work station function DOS key processing API.

Table 6-18. Disable WSF DOS Key Processing Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'62')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Must be zero	Unchanged

Parameter Definitions: The parameters for the Disable Work Station Function DOS key processing API are defined as follows:

- Request parameters

The session ID is the ID of the display session to stop processing the DOS keys.

Return Codes: The return codes for the Disable Work Station Function DOS key processing API are defined as follows:

- System return codes

  See "Return Codes for the Keyboard Services" on page 6-23 for a description of the system return codes found in the CH and CL registers.

- Keyboard services return codes

  You can receive the following keyboard services error codes for this service:

Hex Code	Meaning
00	Successful completion.
01	Option byte is not valid.
02	Session ID is not valid.
04	Session is not connected for keyboard services.
0C	Byte 0 of the parameter list is not zero on request.
15	Session must be host system display session.

Hot Key to Work Station Function

This API will allow a PC application to execute a hot key to a specified work station function session.

Register Values:

On request

AH = X'09'
AL = X'09'
BH = X'80'
BL = X'20'
CX = X'0000'
DX = Resolved gate ID for KEYBOARD
ES = Segment address of the parameter list
DI = Offset address of the parameter list

On completion

CH = X'12'
CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-19 on page 6-37 shows the format for the Hot Key to Work Station Function API parameter list.

Table 6-19. Format for the Hot Key to Work Station Function API Parameter List

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'62')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Reserved	Reserved

Parameter Definitions: The parameters for the Hot Key to Work Station Function API are defined as follows:

- Request parameters

  The session ID is the ID of the display session to which to hot key. The available options are 1 - 5 for the five work station function sessions, or X'FF' to hot key to the DOS session.

Return Codes: The Hot Key to Work Station Function API return codes are defined as follows:

- System return codes

  See "Return Codes for the Keyboard Services" on page 6-23 for a description of the system return codes found in the CH and CL registers.

- Keyboard services return codes

  Bytes 0 and 1 of the parameter list contain a return code generated by the keyboard management portion of the WSF API. The function ID is in byte 1, and the error number is in byte 0. Keyboard services return codes use a function ID of X'62'.

  You can receive the following keyboard services error codes for this service:

Hex Code	Meaning
00	Successful completion
02	Session ID is not valid
7F	Session is not available

Considerations: Hot keying from DOS to WSF suspends the execution of the calling program until a subsequent hot key to DOS.

Copy Service Requests

This section describes the copy service supplied by the API.

To request the copy service, load the registers and parameter list with the correct values, and use the INT 7AH instruction to signal the API that it has a request to process.

Note: Before your application program requests the copy service, it must request the name resolution service, using COPY as the gate name in the parameter list. (Remember that the gate name must be padded to the right with spaces if it has less than eight characters.)

Return Codes for the Copy Service

The copy service has two return codes — a system return code and a copy service return code. Both types of return codes are 2-byte values made up of a function ID and an error number.

The function ID indicates the portion of WSF API in which the error occurred. The error number indicates the specific type of error that has occurred. An error number of X'00' always indicates a successful acceptance or completion of the request.

The copy service return codes are defined as follows:

- System return codes

  After your application program has requested the copy service, the CH and CL registers contain a return code created by the request processing portion of WSF API. The function ID is in the CH register, and the error number is in the CL register. System return codes use a function ID of X'12'. The following error numbers can appear:

Hex Code	Meaning
00	Request accepted
05	Name resolution value (DX register value) specified not valid
07	Reply (BH register value) specified not valid
08	Wait type (BL register value) specified not valid
2F	Supervisor service member (AH register value) specified not valid
34	Service number (AL register value) specified not valid
7F	Session not available (started and communicating with host)

- Copy service return codes

  After the copy service has completed, bytes 0 and 1 of the parameter list contain a return code created by the copy management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. The copy service return codes use a function ID of X'64'.

Copy String

Copies a string from an application buffer to the work station function 5250 EBCDIC buffer, or from the 5250 EBCDIC buffer to the application buffer. The service is complete when the data is physically in the 5250 EBCDIC buffer or the application buffer, depending on the direction of the copy. The data copied is not translated in any way during the copy unless the target of the copy is a presentation space buffer. An exact duplicate is made of the string in the target buffer regardless of the direction in which the copy is performed.

The Copy String service does not affect the cursor position.

Note: Issue the name resolution and Query Session ID service requests before you issue the Copy String service request.

Register Values:

On request

AH = X'09'

AL = X'01'

BH = X'80'

BL = X'20'

CX = X'00FF'

DX = Resolved gate ID for COPY

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-20 shows the Copy String service parameter list format.

Table 6-20. Copy String Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Must be zero	Return code
1	1 byte	Must be zero	Function ID (X'64')
2	1 byte	Source session ID or zero[1]	Unchanged
3	1 byte	Reserved - must be X'00'	Reserved
4	1 word	Offset address of source buffer[1]	Unchanged
6	1 word	Segment address of source buffer[1]	Unchanged
8	1 byte	Reserved - must be X'00'	Unchanged
9	1 byte	Source session type (X'02')[1]	Unchanged
10	1 word	Offset of source-starting character	Unchanged
12	1 word	Offset of source-ending character	Unchanged
14	1 byte	Target session ID or zero[1]	Unchanged
15	1 byte	Reserved - must be X'00'	Reserved
16	1 word	Offset address of target buffer[1]	Unchanged
18	1 word	Segment address of target buffer[1]	Unchanged
20	1 byte	Reserved - must be (X'00')[1]	Unchanged
21	1 byte	Target session type (X'02')[1]	Unchanged
22	1 word	Offset of target starting character	Unchanged
24	1 byte	Reserved - must be X'00'	Unchanged
25	1 byte	Reserved	Reserved
26	1 byte	Target type	Unchanged

Note:

1 The content of this offset depends on whether you are using an application buffer or a 5250 EBCDIC buffer as the copy source/target. See "Parameter Definitions" for more information.

Parameter Definitions: The Copy String service parameters are defined as follows:

- If the copy source is the 5250 EBCDIC buffer:
  - The source session ID is the ID of the session containing the string to copy.
  - The offset and segment address of the source 5250 EBCDIC buffer must be zero.
  - The source session type is X'02' for the 5250 device buffer format.
- If the copy source is the application buffer:
  - The source session ID must be zero.
  - The offset and segment address of the source application buffer contains the source string of the copy.
  - The source session type is X'02' for the 5250 device buffer format.
- The offset of the source-starting character is the offset into the application buffer or the 5250 EBCDIC buffer of the first character in the string to be copied.
- The offset of the source-ending character is the offset into the application buffer or the 5250 EBCDIC buffer of the last character in the string to be copied.
- If the copy target is the 5250 EBCDIC buffer:
  - The target session ID is the ID of the display session to receive the copied string.
  - The offset and segment address of the target buffer must be zero.
  - The offset of the target starting character is the offset into the 5250 EBCDIC buffer where the copy of the string should begin. The character offset values are from 0 to 1919 for the 24 X 80 EBCDIC buffer and from 0 to 3563 for the 27 X 132 EBCDIC buffer.
  - The copy target type must be X'00' to indicate that the target is an EBCDIC buffer. No field attributes are translated.
- If the copy target is the application buffer:
  - The target session ID must be zero.
  - The offset and address of the target application buffer contains the target string of the copy.
  - The target session type is X'02' for the 5250 device buffer format.
  - The offset of the target starting character is the offset into the application buffer where the copy of the string should begin. The character offsets range from 0 to the length of the application buffer.
  - The copy target type must be X'00' to indicate that the target is an EBCDIC buffer. No field attributes are translated.
- If the copy target is the presentation space:
  - The target session ID must be zero.
  - The offset and segment address of the target buffer must not be zero. The presentation space buffer must be allocated by the application program, and its address must be set in the parameter list. The target presentation

space buffer is not the buffer used for the PC display. This is to allow data to be copied and translated in the application program or 5250 EBCDIC buffer to the application program's presentation space buffer.

- The offset of the target starting character is the offset into the presentation space where the copy of the string should begin. The character offsets range from 0 to the length of the presentation space.
- The copy target type must be X'01' to indicate that the target is a presentation space buffer. All field attributes are translated.

Return Codes: The Copy String service return codes are defined as follows:

- System return codes

  See "Return Codes for the Copy Service" on page 6-38 for a description of the system return codes found in the CH and CL registers.

- Copy services return codes

  Bytes 0 and 1 of the parameter list contain a return code created by the copy management portion of WSF API. The function ID is in byte 1, and the error number is in byte 0. Copy services return codes use a function ID of X'64'.

  You can receive the following copy services error codes for this service:

Hex Code	Meaning
00	Successful completion
02	Session ID not valid
03	Target window is input inhibited
05	Source and target area overlap (copy performed)
06	Missing or invalid parameter on source definition
07	Missing or invalid parameter on target definition
09	Truncation occurred (copy performed)
0C	Byte 0 of the parameter list not zero on request
0D	Target not allowed
15	Session must be host display session

- When the 5250 EBCDIC buffer is the target of a copy, if the starting and ending character offsets in the source causes the copy to extend beyond the end of the 5250 EBCDIC buffer, the data up to the end of this buffer is copied. Then the Copy String service return code of X'09' is returned to the application program.
- For any given source, if the starting character offset is greater than the ending character offset, a copy is not performed, and the Copy String service return code of X'06' is returned to the application program.
- When the 5250 EBCDIC buffer is the source of the copy, if the starting or ending character offsets are values that are out of range, the data is not copied, and the Copy String service return code of X'06' is returned to the application.
- The buffer change status flag is set when the 5250 EBCDIC buffer is the target of the Copy String service.

Considerations: Consider the following when using the Copy String service:

- Direction of data flow in the copy service

  The direction of the data flow in the copy service is determined by the source

and target session ID and the source and target buffer address values in the parameter list. For example, if the source session buffer address is the application buffer address, and the target session ID is the work station function session ID, the direction of the copy is from the application buffer to the 5250 EBCDIC buffer.

- 5250 EBCDIC buffer

  The 5250 EBCDIC buffer is in the 5250 device buffer format. Each byte of data in the 5250 EBCDIC buffer is either a 5250 EBCDIC character code (X'40' to X'FE') or a 5250 field attribute (X'20' to X'3F'). Each character offset in an EBCDIC buffer consists of 1 byte of data.

  The 5250 EBCDIC buffer is a logical equivalent of a 5250 screen as seen by an operator, except that the operator information area is not included. There is a one-to-one relationship between each character storage location in the 5250 EBCDIC buffer and each character position in the screen. The 5250 EBCDIC buffer is mapped to the display adapter buffer by the WSF. Direct access to the display adapter buffer is not supported by the API.

- Application buffer format

  The application buffer is in the 5250 device buffer format. Each byte of data in the application buffer is either a 5250 EBCDIC character code (X'40' to X'FE') or a 5250 field attribute (X'20' to X'3F'). Each character offset in an application buffer consists of 1 byte of data.

- Presentation space format

  The presentation space is a logical equivalent of a 5250 screen as seen by an operator, except that the operator information area is not included. There is a constant one-to-one relationship between each character storage location (2 bytes — data value and attribute) in the presentation space buffer and the 5250 screen.

- PC character attributes and 5250 field attributes

  Your IBM Personal Computer's display I/O card uses PC character attributes to define the individual characteristics of each character on your display. Each byte on the physical display is mapped to 2 bytes in the display adapter buffer: the character data byte followed by the display attribute byte. Since 5250 field attributes supply a similar function as PC character attributes, the work station function maps the 5250 field attributes to the PC character attributes.

- Offset of starting and ending characters

  The offset of the source-starting character and the offset of the target-starting character are values in the parameter list that determine the offset into the 5250 EBCDIC buffer and the application buffer where the copy starts. The offset of the source-starting character and the offset of the source-ending character are values in the parameter list that determine the length of the string to be copied.

  When the presentation space is the target of the Copy String service, the offsets are doubled to take into account the attribute bytes in the presentation space buffer.

Operator Information Area Service Requests

This section describes the operator information area service supplied by the API.

The operator information area service allows your application program to determine the current settings of the indicators in the operator information area for the display session.

To request the operator information area service, load the registers and parameter list with the correct values, and use the INT 7AH instruction to signal the API that it has a request to process.

Notes:

1. Before your application program can request the operator information area service, it must request the name resolution service, using OIAM as the gate name in the parameter list.
2. The gate name must be padded to the right with spaces if it is less than 8 characters.

Return Codes for the Operator Information Area Service

The operator information area service has two return codes: a system return code and an operator information area service return code. Both types of return codes are 2-byte values that contain a function ID and an error number.

The function ID indicates the portion of the WSF API in which the error occurred. The error number indicates the specific type of error that occurred. An error number of X'00' always indicates a successful acceptance or completion of the request.

The operator information area service return codes are defined as follows:

- System return codes

  After your application program requests the operator information area service, the CH and CL registers contain a return code created by the request processing portion of WSF API. The function ID is in the CH register and the error number is in the CL register. System return codes use a function ID of X'12'. You can receive the following error codes:

Hex Code	Meaning
00	Request accepted
05	Name resolution value (DX register value) specified not valid
07	Reply (BH register value) specified not valid
08	Wait type (BL register value) specified not valid
2F	Supervisor service member (AH register value) specified not valid
34	Service number (AL register value) specified not valid
7F	Session not available (started)

  These system return codes apply to the operator information area service

- Operator information area service return codes

  After a requested operator information area service completes,
  bytes 0 and 1 of the parameter list contain a return code created by the operator information area management portion of WSF API. The function ID is in

byte 1, and the error number is in byte 0. Operator information area service return codes use a function ID of X'6D'. The error numbers that appear are defined in the description of the Read Operator Information Area Group service.

Read Operator Information Area Group Service

Obtains a bit string that indicates the current settings of a group of indicators in the operator information area of the specified display session and returns the indicator history data.

The indicator's current settings represent the current status of the 5250 display session as follows:

- Diacritic mode (DM)
- Input mode (IM)
- Input inhibited (II)
- Keyboard shift (KS)
- Language or Latin layer status
- Message waiting (MW)
- Right to left cursor direction
- Right to left screen direction
- System available (SA)

Additional indicators for bidirectional languages:

- Right to left cursor direction
- Language layer
  - Used for all language or Latin layer switching keyboards.
- Right to left screen direction
- Flag to indicate a language or Latin layer switching keyboard is present.

Use the indicator history data to determine whether specific indicators have been altered since the last time this service was run.

Note: Issue the name resolution and Query Session ID service requests before you issue the Read Operator Information Area Group service request.

Register Values:

On request

AH = X'09'

AL = X'02'

BH = X'80'

BL = X'20'

CX = X'00FF'

DX = Resolved gate ID for OIAM

ES = Segment address of the parameter list

DI = Offset address of the parameter list

On completion

CH = X'12'

CL = Return code

The contents of registers AX, BX, DX, ES, and DI cannot be predicted.

Parameter List Format: Table 6-21 shows the Read Operator Information Area Group service parameter list format.

Table 6-21. Read Operator Information Area Group Service Parameter List Format

Offset	Length	Contents on Request	Contents on Completion
0	1 byte	Specifies the size of the OIA buffer, where: • 1 = 4-byte OIA buffer • 0 = 3-byte OIA buffer **Note:** A 3-byte OIA buffer is used with Version 1 of PC Support/400.	Return code
1	1 byte	Must be zero	Function ID (X'6D')
2	1 byte	Session ID	Unchanged
3	1 byte	Reserved	Reserved
4	1 word	Offset address of OIA buffer	Unchanged
6	1 word	Segment address of OIA buffer	Unchanged

Parameter Definitions: The Read Operator Information Area Group service parameters are defined as follows:

- Request parameters
  - The session ID is the ID of the display session returned by the Query Session ID service from which operator information area (OIA) data is retrieved.
  - The OIA buffer is the buffer where the OIA data is returned. The buffer must be 4 bytes long. Version 1 of PC Support/400 uses a 3-byte OIA buffer. Offset 0 of the *Read Operator Information Area Group Service Parameter List* specifies the size of the OIA buffer (see Table 6-21).
- Completion parameters

  The OIA buffer is organized as follows:
  - Byte 1, current indicator status, is shown in Table 6-22.

Table 6-22. Byte 1 Format of the OIA Buffer Completion Parameter

IBM Bit Number	Intel Bit Number	Bit Description
0	7	Message waiting light (MW): • 0 = No messages are waiting. • 1 = One or more messages are waiting.
1	6	Reserved.
2	5	Keyboard shift light (KS): • 0 = Keyboard not shifted. • 1 = Keyboard shifted.
3	4	Reserved.
4	3	Insert mode light (IM): • 0 = Not in insert mode. • 1 = In insert mode.
5	2	Diacritic mode light (DM): • 0 = Not in diacritic mode. • 1 = In diacritic mode.
6	1	Input inhibited light (II): • 0 = Input is allowed. • 1 = Input is inhibited.
7	0	System available light (SA): • 0 = System is not available. • 1 = System is available.

– Byte 2, indicator set (alternated from 0 to 1) history is shown in Table 6-23.

Table 6-23. Byte 2 Format of the OIA Buffer Completion Parameter

IBM Bit Number	Intel Bit Number	Bit Description
0	7	Message waiting light (MW): • 0 = Indicator not alternated from 0 to 1. • 1 = Indicator alternated from 0 to 1.
1	6	Reserved.
2	5	Keyboard shift light (KS): • 0 = Indicator not alternated from 0 to 1. • 1 = Indicator alternated from 0 to 1.
3	4	Reserved.
4	3	Insert mode light (IM): • 0 = Indicator not alternated from 0 to 1. • 1 = Indicator alternated from 0 to 1.
5	2	Diacritic mode light (DM): • 0 = Indicator not alternated from 0 to 1. • 1 = Indicator alternated from 0 to 1.
6	1	Input inhibited light (II): • 0 = Indicator not alternated from 0 to 1. • 1 = Indicator alternated from 0 to 1.
7	0	Reserved.

– Byte 3, indicator reset (alternated from 1 to 0) history, is shown in Table 6-24.

Table 6-24. Byte 3 Format of the OIA Buffer Completion Parameter

IBM Bit Number	Intel Bit Number	Bit Description
0	7	Message waiting light (MW): • 0 = Indicator not alternated from 0 to 1. • 1 = Indicator alternated from 0 to 1.
1	6	Reserved.
2	5	Keyboard shift light (KS): • 0 = Indicator not alternated from 0 to 1. • 1 = Indicator alternated from 0 to 1.
3	4	Reserved.
4	3	Insert mode light (IM): • 0 = Indicator not alternated from 1 to 0. • 1 = Indicator alternated from 1 to 0.
5	2	Diacritic mode indicator (DM): • 0 = Indicator not alternated from 1 to 0. • 1 = Indicator alternated from 1 to 0.
6	1	Input inhibited light (II): • 0 = Indicator not alternated from 1 to 0. • 1 = Indicator alternated from 1 to 0.
7	0	Reserved.

– Byte 4

Note: WSFAPI only changes this byte if the value for offset 0 of the *Read Operator Information Area Group Service Parameter List* is 1.

Table 6-25. Byte 4 Format of the OIA Buffer Completion Parameter

IBM Bit Number	Intel Bit Number	Bit Description
0	7	Language or Latin layer capability: • 0 = WSF keyboard does not support Latin layer or language layer switching. • 1 = WSF keyboard supports Latin layer or language layer switching.
1	6	Cursor movement direction: • 0 = Cursor direction is left to right in input field. • 1 = Cursor direction is right to left in input field.
2	5	Orientation of host screen: • 0 = Screen direction is left to right. • 1 = Screen direction is right to left.
3	4	Keyboard layer status: • 0 = Latin layer is active. • 1 = Language layer is active.
4	3	Reserved.
5	2	Reserved.
6	1	Reserved.
7	0	Reserved.

Return Codes: The Read Operator Information Area Group service return codes are defined as follows:

- System return codes

  See "Return Codes for the Operator Information Area Service" on page 6-43 for a description of the system return codes found in the CH and CL registers.

- Operator information area service return codes

  Bytes 0 and 1 of the parameter list contain a return code created by WSF API. The function ID is in byte 1, and the error number is in byte 0. Operator information area service return codes use a function ID of X'6D'. You can receive the following keyboard services error codes for this service:

Hex Code	Meaning
00	Successful completion
02	Session ID not valid
0C	Byte 0 of the parameter list not zero on request
15	Session must be host display session

Note: Run this service once, initially, to establish the time from which the indicator history can be taken. The internal indicator history data returns to zero each time this service is run. The indicator history represents the indicator alternating since the last time this service was run.

Character Code Table

This section lists the valid character codes for the Read Input and Write Keystroke services. The ASCII values that you can send or receive include:

- All standard ASCII characters representing keys that can be received from the keyboard, except the at sign (@), which must be sent as @@.
- ASCII short form of operation names representing host keystrokes that do not have ASCII codes. These shortened names are 2 to 6 bytes long and start with @.

Keystrokes represented by an ASCII value from the 256 ASCII character set are sent or received as that ASCII value. (For example, **a** = X'61'.)

These values are based on the active code page (ACP). This means that any translation is done using the ACP. EBCDIC-to-ASCII translates from the host code page EBCDIC to the active PC ASCII code page and from the active PC ASCII code page to the host code page EBCDIC.

The first column of Table 6-26 lists a description of the key, and the second column lists the ASCII short form of operation name value of the keystroke.

Note: Code values not found in this table are not valid.

Table 6-26 (Page 1 of 4). Character Codes

Character or Function Key Description	ASCII Short Form of Operation Name
Alt cursor	@A@o
Alt key down[1]	@A@d
Alt key up[1]	@A@u
At sign (@)	@@
ASCII characters (except @)[2]	X'00'–X'3F' X'40'–X'FE'
Attention	@A@Q
Backspace	@<
Base	@B@B
Caps Lock[1]	@Y
Clear	@C
Close	@B@C
Cmd	@G
Corrective backspace	@A@9
Cursor down	@V
Cursor down fast	@A@V
Cursor left	@L
Cursor left fast	@A@L
Cursor right	@Z
Cursor right fast	@A@Z
Cursor select	@A@J

Table 6-26 (Page 2 of 4). Character Codes

Character or Function Key Description	ASCII Short Form of Operation Name
Cursor up	@U
Cursor up fast	@A@U
Delete	@D
Dup	@S@x
End	@q
Enter	@E
Erase end of field	@F
Erase end of line	@A@E
Erase input	@A@F
Error reset	@R
Field exit	@X
Field mark	@S@y
Field -	@A@-
Field +	@A@+
F1	@1
F2	@2
F3	@3
F4	@4
F5	@5
F6	@6
F7	@7
F8	@8
F9	@9
F10	@a
F11	@b
F12	@c
F13	@d
F14	@e
F15	@f
F16	@g
F17	@h
F18	@i
F19	@j
F20	@k
F21	@l
F22	@m
F23	@n
F24	@o

Table 6-26 (Page 3 of 4). Character Codes

Character or Function Key Description	ASCII Short Form of Operation Name
Help	@A@p
Hexadecimal	@A@X
Home	@0
Hot-key to DOS	@A@0
Hot-key round robin	@A@N
Hot-key to session 1	@A@1
Hot-key to session 2	@A@2
Hot-key to session 3	@A@3
Hot-key to session 4	@A@4
Hot-key to session 5	@A@5
Insert	@I
Insert toggle	@A@I
Keyboard overrun	@A@K
Local select mode	@A@M
Mode	@M
New line	@N
Num lock[1]	@t
Pad 0	@S@0
Pad 1	@S@1
Pad 2	@S@2
Pad 3	@S@3
Pad 4	@S@4
Pad 5	@S@5
Pad 6	@S@6
Pad 7	@S@7
Pad 8	@S@8
Pad 9	@S@9
Pad .	@S@.
Pad ,	@S@,
Page Down	@v
Page Up	@u
PA1	@x
PA2	@y
PA3	@z
Print, host	@P
Print, PC	@K
Record backspace	@A@<
Reverse	@B@R

Table 6-26 (Page 4 of 4). Character Codes

Character or Function Key Description	ASCII Short Form of Operation Name
Shift key down[1]	@S@d
Shift key up[1]	@S@u
Shift lock[1]	@S@l
Space	@O
System request	@A@H
Tab Left (Backtab)	@B
Tab Right (Tab)	@T
Test request	@A@C

Notes:

1 The Read Input service may return these function keys. They are accepted as input to the Write Keystroke service but are not needed to input keystrokes to the Write Keystroke service. For example, the application programs can use the Write Keystroke service to write an ASCII uppercase A by sending the uppercase A (X'41') directly. It is not necessary to send an @S@d, ASCII lowercase A, and @S@u. An application program can use these shift keys to keep track of the different shift conditions (Alt State, Num Lock, Caps Lock, and Shift keys, to name a few).

2 The ASCII data keys are mapped to EBCDIC through the ASCII-to-EBCDIC translation table. Not all of these ASCII values may have an equivalent EBCDIC value on translation. They may be mapped to a common EBCDIC character (such as EBCDIC X'1F').

BASIC Application Program Interface Access

Sample BASIC programs are provided with PC Support to access functions of the work station function. (These programs are identified by the extension BAS.) You can use these programs when you write personal computer application programs to interface with the work station function. The sample programs use the method described in "BASIC Application Program Interface Access" to determine if the work station function is loaded and can be studied for additional information. The following is a description of the programs:

WSFASO.BAS This sample program allows you to sign on automatically to the host from a BASIC application program.

WSFSS.BAS This program copies the contents of the session screen to a diskette file without displaying the screen as it copies.

WSFSS2.BAS This program copies the contents of the session screen to a diskette file, displaying the screen as it copies.

WSFSUBS.BAS This program checks if the work station function is loaded.

WSFSBPAI.ASM This is the source for the assembler language program which is used to provide an interface between the BASIC programs and the work station function program.

WSFSBPAI.BIN This program is an assembler language program which is used to provide an interface between the BASIC programs and the work station function program.

Commented lines in the source files contain information about the BASIC programs. Source code is provided if you want to change these programs to meet your requirements. The programs are distributed with PC Support on the QIWSFLR folder. Use shared folder support on your personal computer to read, copy, or change them. Become familiar with their content and function before you attempt to change or use them.

In addition to the sample BASIC programs, an application programming interface (API) is available to access the work station function from assembler language programs.

Figure 6-2 on page 6-53 is an excerpt from WFSUBS.BAS to provide an example of using BASIC to identify whether or not the work station function is loaded. This example does not show all programming considerations or techniques. Review the example before you begin application design and coding.

```
6 CLEAR ,&HF000 'Reserve 4K bytes of BASIC's work area for API interface code.
2610 '
2620 ' This subroutine checks that the WSF API is loaded.
2640 '
2650 DEF SEG=0                          'Point to interrupt vector table.
2652 TSEG = PEEK(&H1EA)+256*PEEK(&H1EB) 'Read int 7A segment address.
2654 IF TSEG=0 THEN 2726                'If zero, then not loaded.
2656 '
2658 ' Retrieve BASIC's data segment address.
2660 '
2662 DIM GBS%(3)                        'Array to receive assembler code.
2664 RESTORE 2676                       'Point to DATA statement.
2666 FOR I=0 TO 3: READ GBS%(I): NEXT   'Read in assembler code.
2668 BS%=0                              'BASIC segment address.
2670 DEF SEG                            'Use default BASIC segment address.
2672 DEF USR0 = VARPTR(GBS%(0))         'Point to assembler code.
2674 BS%=USR0(0)                        'Retrieve actual BASIC segment address.
2676 DATA &HDB8C,&H079A,&H0000,&HCBF6:  'Assembler code.
2678 AS% = BS% + &H0F00 'Calc addr of seg where API interface code is loaded.
2680 DEF SEG = AS%               'Current segment is API interface code segment.
2682 '
2684 ' Put API interface assembler code into memory.
2686 '
2688 AO% = 0                         'Offset of next byte of API interface code.
2690 CF$ = "WSFSBPAI.BIN"               'Input file name.
2692 O*EN CF$ FOR INPUT AS #1           'Open input file.
2694 WHILE NOT EOF(1)                   'Quit on end of file.found.
2696 POKE AO%,ASC(INPUT$(1,#1))         'Poke API code into memory.
2698 AO% = AO% + 1                      'Increment offset.
2700 WEND                               'Loop until done.
2702 APII% = 0               'Save address of API interface CALL handling code.
2706 FUNCNO% = 0:APIREL$="   ":APIVER$="   "
2708 CALL APII%(RC%,FUNCCODE%,APIREL$,APIVER$,FUNCNO%)
2710 IF FUNCCODE%<>&H0012 OR RC%<>0 THEN 2726 'API version not retrieved?
2712 PRINT "WSF API version ";APIVER$;", release ";APIREL$;" is running."
2713 SYSTEM
2724 '----- Error exits -----
2726 BEEP: LOCATE 2,10: PRINT "WSF API not loaded.": SYSTEM
```

Figure 6-2. BASIC Application Program Interface Access Example

Chapter 7. Shared Folders Function Low-Level Application Program Interface

This chapter describes the shared folders application program interface (API).

Shared Folders Function Overview

The shared folders function API provides the ability to assign and release shared folders drives, query the drive assignments, and check out or check in files in folders.

The following sections describe in detail:

- Purpose
- Procedure and data structure declarations
- User-supplied values
- Returned values
- Return codes

Assign and Release Drives

DOS Environment

The shared folders function for extended DOS support is required for this API.

Purpose

Assigns or releases drives using the shared folders function for extended DOS support.

Procedure Declaration

The API is provided through INT 21 function call 44H. Input consists of a function code and parameter data. For the assign API, there is also a required input signature ID. Output consists of two return codes and a signature ID. The first return code is from the I/O Control (IOCTL) call and is returned in AX. The second return code is returned in the IOCTL parameter data area. It gives the results of the specific function requested by the IOCTL call (the assign or release request.) The signature ID tells the caller whether or not shared folders was the function that replied to the request. The calling sequence is as follows:

```
L    BL   Drive Number
L    AL   Read Bytes From Drive Function
LDS  DX   IOCTL Data Area
L    CX   IOCTL Data Area Length
L    AH   IOCTL Function Code
INT  21H  Interrupt 21
```

Parameters

Drive Number
: The number of the drive to be assigned or released. Numbers are mapped to drive letters as follows: 0 = the default drive, 1 = drive A, 2 = drive B, and so on.

Read Bytes From Drive Function
: X'04'.

IOCTL Data Area
: The area that contains the IOCTL information.

IOCTL Data Area Length
: The length of the IOCTL data area.

IOCTL Function Code
: X'44'.

Data Area Structures

Table 7-1. Structure of IOCTL Data Area for the Assign Request

Offset	Length	Value	Description
0	4	CSFS	CSFS on input; if shared folders function answers the request, it is SFLR on output.
4	2		Reserved.
6	1		Return code from the assign request.
7	1	X'20'	Function ID X'20' for assign.
8	10		Reserved.
18	9		Name of the host system to which the drive is to be assigned. This is an ASCIIZ string.
27	64		Name of the folder to be assigned. This is an ASCIIZ string. If the first character is X'00', the drive is assigned to all folders. A leading backslash should not be included in the folder name. If a leading backslash is included, the folder path will not be found.

Table 7-2. Structure of IOCTL Data Area for the Release Request

Offset	Length	Value	Description
0	4		ID of calling program on input; if the shared folders function answers the request, it is SFLR on output.
4	2		Reserved.
6	1		Return code from the release request.
7	1	X'21'	Function ID X'21' for release.
8	10		Reserved.

Return Codes

Possible values of the return code in AX on the IOCTL call:

Table 7-3. Return Codes in AX on the IOCTL Call

Return Code	Description
X'00'	The request completed successfully. If the program ID in the data area is SFLR on output, the shared folders function handled the IOCTL call successfully. To find out if the specified assign or release request completed successfully, the return code in the data area must be checked. If the program ID in the data area is not SFLR on output, the shared folders function did not handle this IOCTL call. This happens in the case where the specified drive number on the IOCTL call is for a drive that is not owned by the shared folders function, but by some other driver that supports the DOS IOCTL call.
X'01'	The function code is not valid. This return code is returned if the IOCTL call was sent for a valid drive letter, but the driver owning the drive does not support the IOCTL call.
X'03'	The shared folders function type being used does not support this function. The shared folders function for extended DOS support must be used.
X'0F'	A drive number that is not valid was specified on the IOCTL call.

Possible values of the assign return code in the data area:

Note: This return code is only valid if the return code in AX is X'00' and the program ID in the data area is SFLR on output.

Table 7-4. Assign Return Codes

Return Code	Description
X'00'	No error.
X'03'	Path not found.
X'04'	Folder name too long.
X'05'	Access denied.
X'07'	Incorrect user ID.
X'1F'	General failure.
X'3B'	Network error. An error occurred in the communications layer.
X'55'	Drive already assigned.
X'5B'	Host system must be at Version 1 Release 3 or later.
X'5C'	System name not correct or not active.
X'5E'	Router not active.
X'61'	System name not specified or too long.
X'62'	Contact with system ended. The data link was lost or there was an error on the host system.
X'63'	Resource failure on host system.

Possible values of the release return code in the data area:

Note: This return code is only valid if the return code in AX is X'00' and the program ID in the data area is SFLR on output.

Table 7-5. Release Return Codes

Return Code	Description
X'00'	No error.
X'03'	Open files on drive.
X'1F'	General failure.
X'3B'	Network error. An error occurred in the communications layer.
X'55'	Drive not assigned.
X'62'	Contact with system ended. The data link was lost or there was an error on the host system.

OS/2 Environment

Purpose

Assigns and releases drives using the shared folders function for the OS/2 program.

Procedure Declaration

The API is provided through the OS/2 function call DOSFSATTACH. The return code is in the AX register on return from the call. The format of the call is:

```
PUSH@     ASCIIZ       Device Name
PUSH@     ASCIIZ       FSD Name
PUSH@     OTHER        Data Buffer
PUSH      WORD         Data Buffer Length
PUSH      WORD         Operation Flag
PUSH      DWORD        0
Call      DOSFSATTACH
```

Parameters

Device Name
: Pointer to a drive letter followed by a colon in ASCIIZ format.

FSD Name
: Pointer to the ASCIIZ string EHNSFL0.

Data Buffer
: Pointer to the data buffer.

Data Buffer Length
: Length of the data buffer.

Operation Flag
: The operation to be performed. 0 = Assign, 1 = Release.

0
: Reserved parameter; must be set to 0. This is a double word.

Data Area Structures

Table 7-6. Assign Buffer Format

Offset	Length	Value	Description
0	2	3	Number of parameters; must be set to 3.
2	3		Return code qualifier. This is an ASCIIZ string. The value in this field is only valid if the return code in AX is X'58'.
5	9		Name of host system. This is an ASCIIZ string.
14	*		Path name (maximum 63 characters). This is an ASCIIZ string.

Table 7-7. Release Buffer Format

Offset	Length	Value	Description
0	2		Number of parameters.
2	3		Return code. This is an ASCIIZ string.

Return Codes

Possible values of the return code in AX on the DOSFSATTACH call:

Table 7-8. Return Codes in AX on the DOSFSATTACH Call

Return Code	Description
X'00'	NO_ERROR - No error
X'03'	ERROR_PATH_NOT_FOUND - The specified path was not found.
X'05'	ERROR_ACCESS_DENIED - Not authorized to specified path.
X'08'	ERROR_NOT_ENOUGH_MEMORY - Not enough PC resources to assign drive.
X'0F'	ERROR_INVALID_DRIVE - The specified drive is not valid.
X'15'	ERROR_NOT_READY - Shared folders function has not been started.
X'1F'	ERROR_GENERAL_FAILURE - Communications error occurred.
X'55'	ERROR_ALREADY_ASSIGNED - Drive is already assigned.
X'58'	ERROR_NET_WRITE_FAULT - See Table 7-9 on page 7-6 for the values of the return code qualifiers.
X'7C'	ERROR_INVALID_LEVEL - The specified operation flag is not valid.
X'8E'	ERROR_BUSY_DRIVE - Drive is being used by another program.
X'FC'	ERROR_INVALID_FSD_NAME - The FSD name is not correct or not found.
X'FD'	ERROR_INVALID_PATH - The specified path is not valid.

Possible values of the return code qualifier area:

Note: This return code is only valid if the return code in AX is X'58' (ERROR_NET_WRITE_FAULT).

Table 7-9. Return Code Qualifiers	
Return Code	**Description**
X'5A'	Cannot allocate adequate resources to attach drive.
X'5B'	Version levels for host and PC program do not match.
X'5C'	System name is not correct or system is not active.
X'5D'	Communications manager is not active.
X'5E'	PC Support router is not active.
X'5F'	Local LU specified in CONFIG.PCS file is not correct.

Get Assigned Drive List

Purpose

Allows user applications to query the shared folders drive assignments.

DOS Environment

Procedure Declaration

In the DOS environment, the I/O Control (IOCTL) function call gives the caller the ability to retrieve all drive letters currently assigned to the shared folders function and the corresponding drive assignment information (system name and folder name). Input consists of a function code and parameter data. Output consists of a return code and drive information (drive letter, system name, and folder).

The format of the calling sequence is:

```
L     BL     Drive Number
L     AL     Read Bytes from Drive Function
LDS   DX     IOCTL Data Area
L     CX     IOCTL Data Area Length
L     AH     IOCTL Function Code
INT   21H    Interrupt 21
```

Parameters

Drive Number
: Numbers are mapped to drive letters as follows: 0 = the default drive, 1 = drive A, 2 = drive B, and so on.

Read Bytes From Drive Function
: X'04'.

IOCTL Data Area
: The area that contains the IOCTL information.

IOCTL Data Area Length
: The length of the IOCTL data area.

IOCTL Function Code
: X'44'.

Data Area Structures

Table 7-10. Structure of IOCTL Data Area

Offset	Length	Value	Description
0	4		ID of calling program on input; SFLR on output
4	2		Reserved
6	1		Return code from the Get Assigned Drive List request
7	1	X'14'	Function ID
8	10		Reserved
18	4		Far pointer - Address of data area to receive the assigned drive list
22	2		Length of the data area (must be at least 602 bytes)
24	2		Length of data returned in the data area

Table 7-11. Structure of Data Returned in the Data Area

Offset	Length	Value	Description
0	2		Number of drives assigned
2	1		Drive letter assigned
3	8		System name
11	2		Length of assigned folder name. This field has a value of zero if the drive assigned is a system drive.
13	Variable		Name of assigned folder (if length is not 0)

Additional Information

If more than one drive is assigned, the data returned in the data area contains information for each of the assigned drives. The format of the information is: number of drives assigned, first drive letter, system name, length of folder, name of assigned folder, second drive letter, system name, length of folder, name of assigned folder, and so on. The number of drives assigned can be used to loop through the list of drives and their assignment information.

The user program should issue this IOCTL call to the first nonphysical drive letter and, on return, check the ID field in the IOCTL data area for the SFLR characters. Loop through all drive letters until either SFLR is returned in the ID field along with an AX return code of 0, or until the function has been issued against all non-physical drive letters.

Note: The user program should set this field to something other than SFLR before making the function call.

Return Codes

Return Code	Description
X'57'	One or more of the input parameters were not valid.

OS/2 Environment

Procedure Declaration

In the OS/2 environment, the DOSFSCTL function call gives the caller the ability to retrieve all drive letters currently assigned to the shared folders function and the corresponding drive assignment information (system name and folder name). Input consists of a function code and a data area. Output consists of a return code (in AX) and drive information (drive letter, system name, and folder) in the data area.

The format of the calling sequence is:

```
EXTRN     DOSFSCTL     Far

PUSH@     OTHER        Data Area
PUSH      WORD         Data Area Length
PUSH@     WORD         Length of Data Area on Return
PUSH@     OTHER        Parameter List
PUSH      WORD         Parameter List Length
PUSH@     WORD         Length of Parameter List on Return
PUSH      WORD         Function Code

PUSH@     ASCIIZ       Route Name
PUSH      WORD         File Handle
PUSH      WORD         Route Method
PUSH      DWORD        Reserved

Call      DOSFSCTL
```

Parameters

Data Area
: Address of the area to receive drive information.

Data Area Length
: Length of the data area. This must be at least 602 bytes in length.

Length of Data Area on Return
: Address of a word in the caller's storage that contains the length of the drive information returned in the data area.

Parameter List
: Should be zeros. Because there is no parameter list for this function, it is ignored.

Parameter List Length
: Should be zeros.

Length of parameter list
: Should be zeros.

Function Code
: X'8100'.

Route Name
: EHNSFL0 followed by X'00' (end of ASCIIZ string)

File Handle
: -1.

Route Method
: 3.

Reserved
: A reserved field set to zeros.

Data Area Structures

Table 7-12. Structure of Data Returned in the Data Area

Offset	Length	Value	Description
0	2		Number of drives assigned.
2	1		Drive letter assigned.
3	8		System name.
11	2		Length of assigned folder name. This field has a value of zero if the drive assigned is a system drive.
13	Variable		Name of assigned folder (if length is not zero).

Additional Information

If more than one drive is assigned, the data returned to the data area contains information for each of the assigned drives. The format of the information is: number of drives assigned, first drive letter, system name, length of folder, name of assigned folder, second drive letter, system name, length of folder, name of assigned folder, and so on. The number of drives assigned can be used to loop through the list of drives and their assignment information.

Return Codes

Return Code	Description
X'57'	ERROR_INVALID_PARAMETER - One or more of the input parameters was not valid.
X'15'	ERROR_NOT_READY - Shared folders function is not ready to accept this function.
X'73'	ERROR_PROTECTION_VIOLATION - The data buffer is not in a segment that can use shared folders function.

Check Out and Check In Files in Folders

Purpose

Allows a personal computer user to mark a file in a folder on the AS/400 system as checked out to that user. No other personal computer user or AS/400 office user can change the file while it is checked out. When the personal computer user is finished updating the file, it may be unmarked or checked back in. This API also provides some list capabilities of checked out files. In the DOS environment, the API is provided through an INT 21 Function Call 44 (IOCTL). In the OS/2 environment, the API is provided through a DOSFSCTL call.

Notes:

1. This function applies only to documents and files residing in folders on the AS/400 system. It does not apply to any other objects on the AS/400 system. Folders, system files, and hidden files may not be checked out.
2. A user must have at least *CHANGE authority to a file in order to check it out.

Check file functions are performed using a combination of IOCTL (DOSFSCTL) calls and file system calls. Sometimes the IOCTL (DOSFSCTL) call must precede a file system call. These pairings are:

IOCTL (DOSFSCTL) Call	File System Call
Open for check-out (ID X'16') - Specifies which file is to be checked out.	Open - Opens the file to be checked out (must be for exclusive read/write access).
Open for check-in (ID X'18') - Specifies which file is to be checked in.	Open - Opens the file to be checked in (must be for exclusive read/write access).
Do not check file out (ID X'17') - Specifies that a file previously opened for check-out should not be checked out.	Close - Closes the file, but does not check it out. **Note:** To actually check the file out, do not precede the close with IOCTL (DOSFSCTL) ID X'17'.
Do not check file in (ID X'19') - Specifies that a file previously opened for check-in should not be checked in.	Close - Closes the file, but does not check it in. **Note:** To actually check the file in, do not precede the close with IOCTL (DOSFSCTL) ID X'19'.
List check-out status of a directory (ID X'1B') - Specifies the directory for which check-out status is requested.	Search first - Begins the search of the directory for which check-out status is requested.
List files checked out to a user (ID X'1C') - Specifies the directory to search and the user ID to use.	Search first - Begins the search of the directory for files checked out to the user.
List directories only (ID X'1D') - Specifies the directory to search.	Search first - Begins the search of the directory for any subdirectories it may contain.

At other times, the IOCTL (DOSFSCTL) call follows a file system call. These pairings are:

File System Call	IOCTL (DOSFSCTL) Call
Open - An open for check-out or check-in which failed with a bad return code.	Get additional check-out or check-in information (ID X'1A') - Queries for a more detailed description of the failure.
Search first or search next - A successful search first or search next that was issued for check-out information.	Get additional check-out list information (ID X'1E') - Queries for any check-out information associated with the file returned on the search first or the search next call.

The file name passed on the IOCTL (DOSFSCTL) calls must be identical to the file name passed on system calls. The file name is not changed on an IOCTL (DOSFSCTL) call, but the operating system may change the file name on a system call. For example, the operating system will remove the combination of backslash and period characters (\.) on a function call.

In the OS/2 operating system, a handle is returned on the first DOSFSCTL call of a series. The handle is returned on DOSFSCTL calls with a function ID of X'16', X'18', X'1B', X'1C', or X'1D'. This handle must be used on all following DOSFSCTL calls in the series.

The following is an example algorithm of how an application might make use of this API.

```
To check a file out:

    IOCTL (DOSFSCTL) call - Open for CheckOut

    OPEN

    If error Then

        IOCTL (DOSFSCTL) call - Get additional check information.
        Returns any additional information associated with the error.

    Else

        Optionally read from file

        If there was an unsuccessful attempt to copy the file Then

            IOCTL (DOSFSCTL) call - Close for CheckOut but don't
            mark the file as checked out.

        CLOSE

To check a file back in:

    IOCTL (DOSFSCTL) call - Open for CheckIn

    OPEN

    If error Then

        IOCTL (DOSFSCTL) call - Get additional check information.
```

```
        Returns any additional information associated with the error.

    Else

        Optionally write to the file

        If there was an unsuccessful attempt to copy the file Then

            IOCTL (DOSFSCTL) call - Close for CheckIn but don't
            mark the file as checked in.

        CLOSE

To get directory information:

    IOCTL (DOSFSCTL) call - List Directory Information

    SEARCH FIRST

    While more files or directories

        If a file was returned Then

            IOCTL (DOSFSCTL) call - Get additional list information.
            This would be done to see if there was any check
            information associated with the file.

        SEARCH NEXT

To get user information:

    IOCTL (DOSFSCTL) call - List user information

    SEARCH FIRST

    While more files

        IOCTL (DOSFSCTL) call - Get additional list information.
        This would be done to find the user id (if needed) and
        the date and time the file was checked out.

    SEARCH NEXT
```

DOS Environment

The shared folders function for extended DOS support is required for this API.

Procedure Declaration

In the DOS environment, the API is provided through INT 21 function call 44H. Input consists of a function code and parameter data. Output consists of a return code and user information. The calling sequence is as follows:

```
L    BL   Drive Number
L    AL   Read Bytes From Drive Function
LDS  DX   IOCTL Data Area
L    CX   IOCTL Data Area Length
L    AH   IOCTL Function Code
INT  21H  Interrupt 21
```

Note: All reserved words in the data structures must be initialized to zero.

Parameters

Drive Number
: A valid shared folders drive number. This is the drive that contains the file or files to be checked out, checked in, or listed. Numbers are mapped to drive letters as follows: 0 = the default drive, 1 = drive A, 2 = drive B, and so on.

Read Bytes From Drive Function
: X'04'.

IOCTL Data Area
: The area that contains the IOCTL information.

IOCTL Data Area Length
: The length of the IOCTL data area.

IOCTL Function Code
: X'44'.

Data Area Structures

Table 7-13 (Page 1 of 2). Structure of IOCTL Data Area

Offset	Length	Value	Description
0	4		ID of calling program on input; SFLR on output.
4	2		Reserved.
6	1		Return code from the check-out or check-in request.
7	1		Function ID.
		X'16'	Open for check-out.
		X'17'	Close for check-out but do not check it out. This is used when a copy operation is requested but cannot be performed (for example, there is not enough disk space to store the file). The file still must be closed, but it should not be marked as checked out. If a file has been opened for check-out and a close request comes in for the file, but an IOCTL call with this function ID did not come in, the file is closed for check-out and is marked as checked out.
		X'18'	Open for check-in.

Table 7-13 (Page 2 of 2). Structure of IOCTL Data Area

Offset	Length	Value	Description
		X'19'	Close for check-in but do not check it in. This is used when a copy operation is requested but cannot be performed (for example, there is not enough disk space to store the file). The file must still be closed, but it should not be marked as checked in. If a file has been opened for check-in and a close request comes in for the file, but an IOCTL call with this function ID did not come in, the file is closed for check-in and is marked as checked in.
		X'1A'	Get additional information for check-in or check-out operation.
		X'1B'	Search for check-out directory information.
		X'1C'	Search for check-out user information.
		X'1D'	Search for directories only.
		X'1E'	Get additional information for list operation.
8	10		Reserved.
18	4		Address of the return data area. This pointer is valid only for function ID X'1A' or X'1E'. For all other function ID values, this pointer is ignored. **Note:** It is assumed that all pointers passed in DOS for the IOCTL calls are passed as real mode pointers.
22	2		Length of the return data area. This must be at least 26 bytes.
24	10		User ID. This field is only valid for function ID X'1C'. If the function ID is X'1C' and the first character is X'00', the user ID used to sign on to the router is used.
34	1		User ID length. This field is valid only for function ID X'1C'.
35	4		Reserved.
39	127		Fully qualified file name of the form d:\path\filename.
166	1		File name length.

Table 7-14 (Page 1 of 2). Structure of Return Data Area

Offset	Length	Value	Description
0	10		User ID. If the first character is X'00', this field is not applicable for this request.
10	1		User ID length. If the length is 0, the user ID is not applicable for this request.
11	2		Check-out date.
13	2		Check-out time.

Table 7-14 (Page 2 of 2). Structure of Return Data Area			
Offset	**Length**	**Value**	**Description**
15	11		Reserved.

Return Codes

Possible values of the return code in AX on the IOCTL call:

Table 7-15. Return Codes in AX on the IOCTL Call	
Return Code	**Description**
X'00'	The request completed successfully. If the program ID in the data area is SFLR on output, shared folders handled the IOCTL call successfully. To find out if the specified check file request completed successfully, the return code in the data area must be checked. If the program ID in the data area is not SFLR on output, the shared folders function did not handle this IOCTL call. This happens in the case where the specified drive number on the IOCTL call is for a drive that is not owned by the shared folders function, but by some other driver that supports the DOS IOCTL call.
X'01'	The function code is not valid. This return code is returned if the IOCTL call was sent for a valid drive letter, but the driver owning the drive does not support the IOCTL call.
X'03'	The shared folders function type being used does not support this function. The shared folders function for extended DOS support must be used.
X'0F'	The specified drive number is not valid.

Possible values of the return code in the data area:

Note: This return code is only valid if the return code in AX is X'00'.

Table 7-16 (Page 1 of 2). Return Codes in the Data Area	
Return Code	**Description**
X'00'	No additional information is available.
X'01'	File is already checked out to another user.
X'02'	File is in use.
X'03'	File cannot be checked in. The file is in use or has not been checked out to you.
X'04'	File is not checked out.
X'05'	Drive is not assigned.
X'06'	The sequence of IOCTL calls is not correct. You must finish the sequence of calls for one check out or check in request before starting another check out or check in request. (For example, you may have made an IOCTL call to open a file for check-out, opened the file, then made an IOCTL call to not check in the file when it is closed.)
X'07'	The drive is not assigned to an AS/400 system with the correct operating system level. Version 2 Release 1 Modification 0 or later must be used.

Table 7-16 (Page 2 of 2). Return Codes in the Data Area	
Return Code	**Description**
X'08'	User ID is not correct.
X'09'	Not enough resource to perform function. The pointer to the return data area is zero or the length of the return data area is not large enough.
X'0B'	Path or file name is not correct.
X'0D'	Sharing mode or access intent is not correct. File sharing mode must be deny all and access intent must be read/write.

OS/2 Environment

Procedure Declaration

In the OS/2 environment, the API is provided through a DOSFSCTL call. Input consists of a function code, parameter data, and the name of the FSD. Output consists of a return code and user information. The calling sequence is as follows:

```
EXTRN    DosFsctl:Far

PUSH@    OTHER    Data Area
PUSH     WORD     Data Area Length
PUSH@    WORD     Data Area Length On Return
PUSH@    OTHER    Parameter List
PUSH     WORD     Parameter List Length
PUSH@    WORD     Parameter List Length On Return
PUSH     WORD     Function Code
PUSH@    ASCIIZ   Route Name
PUSH     WORD     File Handle
PUSH     WORD     Route Method
PUSH     DWORD    0
CALL     DOSFSCTL
```

Note: All reserved words in the data structures must be initialized to zero.

Parameters

Data Area
: Address of the area where the user information is returned.

Data Area Length
: Length of the data area. The length must be at least 26 bytes.

Data Area Length On Return
: Address of a word that contains the length of the user information returned in the data area.

Parameter List
: Address of the caller's parameter data area.

Parameter List Length
: Length of the parameter data area.

Parameter List Length On Return
: Address of a word that contains the length of the parameter information stored in this area passed on input and returned by the FSD on output.

Function Code
: X'8202'.

Route Name
: Address of the ASCIIZ string containing the name of the FSD. In this case, the name is EHNSFL0.

File Handle
: -1.

Route Method
: 3.

Zero
: A double word of zeros.

Data Area Structures

Table 7-17 (Page 1 of 2). Structure of Parameter List

Offset	Length	Value	Description
0	7		Reserved.
7	1		Function ID.
		X'16'	Open for check-out.
		X'17'	Close for check-out but do not check it out. This is used when a copy operation is requested but cannot be performed (for example, there is not enough disk space to store the file). The file still must be closed, but it should not be marked as checked out. If a file has been opened for check-out and a close request comes in for the file, but a DOSFSCTL call with this function ID did not come in, the file is closed for check-out and is marked as checked out.
		X'18'	Open for check-in.
		X'19'	Close for check-in but do not check it in. This is used when a copy operation is requested but could not be performed (for example, there is not enough disk space to store the file). The file must still be closed, but it should not be marked as checked in. If a file has been opened for check-in and a close request comes in for the file, but a DOSFSCTL call with this function ID did not come in, the file is closed for check-in and is marked as checked in.
		X'1A'	Get additional information for check-in or check-out operation.
		X'1B'	Search for check-out directory information.
		X'1C'	Search for check-out user information.
		X'1D'	Search for directories only.

Table 7-17 (Page 2 of 2). Structure of Parameter List			
Offset	**Length**	**Value**	**Description**
		X'1E'	Get additional information for list operation.
8	16		Reserved.
24	11		ASCIIZ user ID. This field is only valid for function ID X'1C'. If the function ID is X'1C' and the first character is X'00', the user ID used to sign on to the router is used.
35	4		File handle.
39	128		ASCIIZ file name including drive and path.

Table 7-18. Structure of Data Area			
Offset	**Length**	**Value**	**Description**
0	11		ASCIIZ user ID. If the first character is X'00', this field is not applicable for this request.
11	2		Check-out date.
13	2		Check-out time.
15	1		Extended return code.
16	4		Check-out handle.
20	6		Reserved.

Return Codes

Possible values of the return code in AX on the FSCTL call:

Table 7-19. Return Codes in AX on the FSCTL Call	
Return Code	**Description**
X'00'	NO_ERROR - The command completed successfully.
X'01'	ERROR_INVALID_FUNCTION - Either the function value is not valid or there was an attempt to do a check file request to a shared folders function drive assigned to a System/36 or AS/400 system with a version previous to Version 2 Release 1 Modification 0.
X'06'	ERROR_INVALID_HANDLE - File handle was not -1.
X'57'	ERROR_INVALID_PARAMETER - One or more of the input parameters were not valid.
X'6F'	ERROR_BUFFER_OVERFLOW - The user's data area was not large enough to hold the information being returned by the FSD.
X'73'	ERROR_PROTECTION_VIOLATION - The data buffer was not in a segment that could be used by shared folders.
X'7C'	ERROR_INVALID_LEVEL - The level was not 3.
X'FC'	ERROR_INVALID_FSD_NAME - The FSD name was not EHNSFL0.

Possible values of the return code in the data area:

Note: This return code is only valid if the return code in AX is X'00'.

Table 7-20. Return Codes in the Data Area

Return Code	Description
X'00'	No additional information is available.
X'01'	File is already checked out to another user.
X'02'	File is in use.
X'03'	File cannot be checked in. The file is in use or has not been checked out to you.
X'04'	File is not checked out.
X'05'	Drive is not assigned.
X'06'	The sequence of DOSFSCTL calls is not correct. You must finish the sequence of calls for one check out or check in request before starting another check out or check in request. (For example, you may have made a DOSFSCTL call to open a file for check-out, opened the file, then made a DOSFSCTL call to not check in the file when it is closed.)
X'07'	The drive is not assigned to an AS/400 system with the correct operating system level. Version 2 Release 1 Modification 0 or later must be used.
X'08'	User ID is not correct.
X'09'	Not enough resource to perform function. The shared folders function supports only 10 concurrent check file operations. Try the function request again when one of the current functions completes.
X'0A'	Function is not correct.
X'0B'	Path or file name is not correct.
X'0C'	Handle is not correct. Some DOSFSCTL calls require a handle from the DOSFSCTL call that started the CHKFIL request. The handle passed on this DOSFSCTL call is not correct.
X'0D'	Sharing mode or access intent is not correct. File sharing mode must be deny all and access intent must be read/write.

Chapter 8. Virtual Printer Function Low-Level Application Program Interface

This chapter describes the virtual printer application program interface (API).

The virtual printer function API provides the ability to work with virtual printers. This chapter describes the APIs provided by the Extended DOS version of PC Support/400.

Virtual print provides the following APIs:

- The virtual printer program VPRT.EXE provides an API through the PC Support/400 shared interrupt.
- The virtual printer program VPRT.EXE provides an API through DOS interrupt 21 to support industry-standard redirection calls.
- The virtual printer dynamic link library (DLL) for Windows** provides routines using the virtual printer API in Windows. For more information, see Chapter 24, "Virtual Printer Windows Application Program Interface."
- The OS/2 virtual printer DLL provides routines for using the virtual printer APIs in the OS/2 environment. For additional information, see Chapter 18, "OS/2 Virtual Printer Application Program Interface."

The PC Support tools folder contains a sample program that demonstrates how to use the virtual printer API. For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

Redirection Support

The VPRT program supports the following redirector calls via interrupt 21.

5F02H	Get redirection list entry–request to provide the name of a redirected virtual printer
5F03H	Redirect device–request to redirect an LPT
5F04H	Cancel redirection–request to redirect an LPT
3CH	Create a file–request to create a file or device
3DH	Open a file–request to open a file or device
3EH	Close a file–request to close a file or device
40H	Write to a file or device–request to write to a file or device

Functions

The VPRT.EXE API is accessed through the PC Support SI support with the registers set as follows:

AL = X'82'
ES:BX = Pointer to SI request block

The request block must reside in real memory and the address passed in ES:BX must be a real address. The address fields contained in the request block must also contain real addresses in segment:offset format.

The length of the request block field is defined in each of the API structures and must be set by the caller before making the API call.

On return to the caller, the AH register is set as follows:

X'09'	Busy
X'12'	End-of-list
X'13'	Corrupted list (caller must perform a Get_List to recover)
X'14'	Invalid virtual printer ID
X'15'	Virtual printer already assigned to local device name
X'90'	Function successful
X'FD'	Programming error (message buffer contains 32-bit pointer to parameter list field in error)
X'FE'	Function not successful (message buffer contains displayable message)
X'FF'	Invalid virtual print function

All text strings are ASCIIZ strings, including the messages returned in the 514-byte message buffer.

Suspend_VP

This function suspends VPRT. When VPRT is suspended, it returns BUSY to all requests to print data but processes all other requests. Register ES:BX should contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Reserved

The *Type of Request* field of the request block must be set to X'10'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

Unsuspend_VP

This function returns VPRT to its normal operating state. Register ES:BX should contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Reserved

The *Type of Request* field of the request block must be set to X'11'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

Qry_VP_Status

This function returns the current status of VPRT. Register ES:BX should contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of Qry_VP_Status data structure

The *Type of Request* field of the request block must be set to X'12'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Qry_VP_Status Data Structure Address* field must contain the address of the Qry_VP_Status data structure.

The *Qry_VP_Status Data Structure* is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
1	Number of assigned virtual printers
1	API function level
6	Version_release_level

The *Number of Assigned Virtual Printers* field contains a hexadecimal value for the number of virtual printers assigned to file names.

The *API Function Level* field contains a hexadecimal value representing the current functional level of the VPRT program that is loaded.

The *Version_Release_Level* field contains 6 characters representing the version, release, and level of the VPRT program that is loaded. (For example, Version 2 Release 2 Modification 0 would be represented as X'323032303030'.)

Get_Redir_List

This function returns a list of the devices that are redirected by the VPRT program. This function should be called repeatedly until register AH = X'12'. (end of list). Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of get_redir_list data structure

The *Type of Request* field of the request block must be set to X'13'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Get_Redir_List Data Structure Address* field must contain the address of the Get_Redir_List data structure. The Get_Redir_List data structure is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
2	Index
9	Local device name
1	Virtual printer ID
9	AS/400 system name
1	Reserved
11	AS/400 printer device name
1	Reserved
11	AS/400 printer file library name
1	Reserved
11	AS/400 printer file name
1	Reserved
11	AS/400 output queue library name
1	Reserved
11	AS/400 output queue name
1	Device status

The caller fills in the *Index* field with the index value of the local device (index is zero-based).

The *Local Device Name* field returns an ASCIIZ string representing the local name of the device. Valid local device names are LPT1 through LPT9.

The *Virtual Printer ID* field returns a hexadecimal value that is to be used to reference a specific assigned virtual printer.

The *AS/400 System Name, AS/400 Printer Device Name, AS/400 Printer File Library Name*, and *AS/400 Printer File Name* fields of the structure contain the ASCIIZ names the virtual printer was assigned with. The name fields contain zeros if the name is not specified.

The *AS/400 output queue name* and *AS/400 output queue library name* fields indicate the output queue and output queue library name into which the caller's spooled data is placed. The names are ASCIIZ strings.

The *Device Status* field contains the current status of the redirected device. The returned byte contains bit flags as follows:

X'00'	Device not assigned
X'01'	Device assigned
X'02'	Device assigned–printer file open
X'04'	Device assigned–error pending
X'08'	Device assigned–suspended
X'10'	Device assigned–datapass mode

Qry_Asn_Status

This function returns the current assign status of a specific redirected device. Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of Qry_Asn_Status data structure

The *Type of Request* field of the request block must be set to X'14'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Qry_Asn_Status Data Structure Address* field must contain the address of the Qry_Asn_Status data structure. The Qry_Asn_Status data structure is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
9	Local device name
1	Virtual printer ID
1	Device status
1	Reserved

The *Local Device Name* field identifies the virtual printer for which assign status is requested. The field should contain an ASCIIZ representing the local name of the device. Valid local device names are LPT1 through LPT9.

The *Virtual Printer ID* field returns a hexadecimal value that is to be used to reference a specific assigned virtual printer.

The *Device Status* field contains the current status of the virtual printer. The returned byte contains bit flags as follows:

X'00'	Device not assigned
X'01'	Device assigned
X'02'	Device assigned–printer file open
X'04'	Device assigned–error pending
X'08'	Device assigned–suspended
X'10'	Device assigned–datapass mode

Qry_Asn_Parms

This function returns the current assign parameters for a specific redirected device. Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of Qry_Asn_Parms data structure

The *Type of Request* field of the request block must be set to X'15'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Qry_Asn_Parms Data Structure Address* field must contain the address of the Qry_Asn_Parms data structure. The Qry_Asn_Parms data structure is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
1	Virtual printer ID
1	Device status

9	AS/400 system name
1	Reserved
11	AS/400 printer device name
1	Reserved
5	AS/400 printer type
1	Reserved
11	AS/400 printer file library name
1	Reserved
11	AS/400 printer file name
1	Reserved
11	AS/400 output queue library name
1	Reserved
11	AS/400 output queue name
1	Reserved
1	Data type
1	Copies
1	Timeout value
1	Defer printing (Schedule-*Immed)
2	Characters per inch
2	Characters per line
1	Lines per inch
1	Page length in lines
1	Lines per page (overflow line)
1	Application formatted data
1	ASCII character set
1	Untranslatable character
2	Number of printed files
2	Number of pages
2	Number of untranslatable characters
1	Target operating system
1	Append form feed

Note: The *Characters per line, Lines per page, and ASCII character set* parameters are ignored by the DBCS version of the virtual printer API. The *Target operating system* parameter is ignored by the SBCS version of the API. See Table 24-1 on page 24-16 for a list of the valid parameters for each printer data type.

The *Virtual Printer ID* field identifies the virtual printer for the request. This ID is returned by the QRY_ASN_STATUS call.

The *Device status* field contains the current status of the virtual printer. The returned byte contains bit flags as follows:

X'00'	Device not assigned
X'01'	Device assigned
X'02'	Device assigned–printer file open
X'04'	Device assigned–error pending
X'08'	Device assigned–suspended
X'10'	Device assigned–datapass mode

The *AS/400 System Name, AS/400 Printer Device Name, AS/400 Printer Name, AS/400 Printer File Library Name*, and *AS/400 Printer File Name* fields of the structure contain the ASCIIZ string names the virtual printer was assigned with. The name fields contain zeros if the name was not specified.

The *AS/400 Output Queue Name* and *AS/400 Output Queue Library Name* fields indicate the output queue and output queue library name into which the caller's spooled data is placed. The names are ASCIIZ strings.

The value for the *AS/400 Printer Type* field is 4 ASCII characters followed by a null describing the printer type (such as 3812).

The *Data type* field contains one of the following hexadecimal values:

01 = SCS data
02 = ASCII-to-SCS transform
03 = FFT data
04 = ASCII data
05 = AFPDS data

The *Copies* field contains a hexadecimal value for the number of copies requested. Valid values are 1 to 99 for Data Type 03 and 1 to 255 for all other data types.

The *Timeout Value* field contains a hexadecimal value for the number of seconds the virtual printer is to wait after it stops receiving data and before it closes the output file on the AS/400 system.

The *Defer Printing* field contains X'01' for Yes and X'02' for No.

The *Characters per Inch* field contains the hexadecimal value representing the CPI.

The *Characters per Line* field contains the hexadecimal value representing the number of characters to be printed on each line.

The *Lines per Inch* field contains the hexadecimal value representing the LPI.

The *Page Length in Lines* field contains the hexadecimal value representing the page length in lines.

The *Lines per Page* field contains the hexadecimal value representing the number of lines of data to be printed on each page.

The *Application Formatted Data* field contains X'01' for Yes if the data being printed to the virtual printer has been formatted by an application such as DW/4. The field contains X'02' if the data has not been formatted. If X'02' is specified, virtual print ignores the PC commands found in the data which change cpi, cpl, lpi, lpp and pl, and instead uses the values chosen when the virtual printer was assigned.

The *ASCII Character Set* field contains X'01' for Character Set 1 and X'02' for Character Set 2.

The *Untranslatable Character* field contains the hexadecimal value for an EBCDIC character. This character is used to replace characters in the data which cannot be found in the ASCII-to-EBCDIC translation table.

The *Number of Printed Files* field contains the hexadecimal value for the number of files that have been printed since the virtual printer was assigned.

The *Number of Pages* field contains the hexadecimal value for the number of pages contained in the last file that was printed.

The *Number of Untranslatable Characters* field contains the hexadecimal value for the number of untranslatable characters found in the last file that was printed.

The *Target Operating System* field contains one of the following hexadecimal values:

Hex Value	Condition
X'01'	Data will be printed using the print emulation in the DOS operating system.
X'02'	Data will be printed using the print emulation in the OS/2 operating system.
X'03'	Data will be printed using the print emulation in the DOS/V operating system.

If you do not specify a value for this field, the virtual printer function uses the default value of the current operating system.

The *Append Form Feed* field contains one of the following hexadecimal values:

Hex Value	Condition
X'01'	A form feed will be added to the end of each ASCII transparent print job.
X'02'	A form feed will not be added to ASCII transparent jobs.
X'03'	A form feed will be added to the end of ASCII transparent jobs only if the last byte in the job is not a form feed.

Get_List

This function initiates list request processing. This function must be called before Qry_List_Head or Get_List_Item. It returns the length of the buffer the caller is required to provide for the Qry_List_Head and Get_List_Item functions.

Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of Get_List data structure

The *Type of Request* field of the request block must be set to X'16'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Get_List Data Structure Address* field must contain the address of the Get_List data structure. The Get_List data structure is defined as follows:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	List request ID
1	List type
1	Reserved
9	AS/400 system name
1	Reserved
11	AS/400 printer file library name
1	Reserved

11	AS/400 printer file name
1	Reserved
2	Buffer length

The *List Request ID* field of the request block returns a hexadecimal value to be used with the Qry_List_Head and Get_List_Item function calls.

The *List Type* field of the request block must be set as follows:

X'00'	Printer devices
X'01'	Printer file libraries
X'02'	Printer files

The *AS/400 System Name* field of the request block must contain an ASCIIZ string indicating the name of the AS/400 system.

The *AS/400 Printer File Library Name* field of the request block should contain zeros to get a list of all libraries in the user portion (*USRLIBL) of the AS/400 jobs library list. It may contain characters followed by an asterisk. This partial name is used to list only those libraries which start with the letters specified before the asterisk. If the request is for a printer file list, then only printer files in the fully named library are listed.

The *AS/400 Printer File Name* field of the request block should contain zeros to get a list of all printer files contained in the libraries in the user portion (*USRLIBL) of the AS/400 jobs library list. It may contain characters followed by an asterisk. This partial name is used to list only those printer files which start with the letters specified before the asterisk.

The *Buffer Length* field of the request block is set by VPRT. The buffer length is set to that of the buffer the caller is required to provide for the Qry_List_Head and Get_List_Item requests. The length is the same for the header record and list item records.

Qry_List_Head

This function retrieves the heading record used to display a list of printer devices, libraries, or print files. This request must be made after the Get_List request but may be made either before or after the Get_List_Item requests.

Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of list heading data structure

The *Type of Request* field of the request block must be set to X'17'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *List Heading Data Structure Address* field must contain the address of the list heading data structure. The list heading data structure is defined as follows:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	List request ID
4	Address of list heading buffer

The *List Request ID* field of the request block must contain the hexadecimal value returned by the Get_List request.

The *Address of List Heading Buffer* field of the request block must contain the address of a buffer the caller provides for the list header. The field should be at least as large as the value VPRT returned in the *Buffer Length* field of Get_List command.

Get_List_Item

This function retrieves an item of the list. This request must be made after the Get_List request but may be made either before or after the Qry_List_Head request.

Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of list item data structure

The *Type of Request* field of the request block must be set to X'18'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

Register AH contains X'12' when no more list items are available.

The *List Item Data Structure Address* field must contain the address of the list item data structure. The List Item Data Structure is defined as follows:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	List request ID
4	Address of list item buffer

The *List Request ID* field of the request block must contain the hexadecimal value returned by the Get_List request.

The *Address of List Item Buffer* field of the request block must contain the address of a buffer the caller provides for the list item. The field should be at least as large as the value VPRT returned in the *Buffer Length* field of Get_List command.

Verify_Assign_Device

This function verifies the AS/400 device name or AS/400 printer file name supplied by the user, and retrieves the supported assign parameters for a given AS/400 device or printer file. If an AS/400 device name or AS/400 printer file name is not specified, then the supported assign parameters for printer file QPVPRINT in library QIWS is returned. Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of Verify_Assign_Device data structure

The *Type of Request* field of the request block must be set to X'19'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Verify_Assign_Device Data Structure Address* field must contain the address of the verify data structure. The Verify_Assign_Device Data Structure is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
9	AS/400 system name
1	Reserved
11	AS/400 printer device name
1	Reserved
11	AS/400 printer file library name
1	Reserved
11	AS/400 printer file name
1	Reserved
11	AS/400 output queue library name
1	Reserved
11	AS/400 output queue name
1	Reserved
2	Default value for characters per inch
2	Default value for characters per line
1	Default value for lines per inch
1	Default value for page length in lines
1	Default value for lines per page (overflow line)
1	Default value for copies
1	Default value for defer printing (Schedule-*Immed)
1	Reserved
5	AS/400 printer type
1	Reserved
20	List of CPI values supported (max = 10 x 2 bytes)
10	List of LPI values supported (max = 10 x 1 byte)
10	List of data types supported (max = 10 x 1 byte)

The *AS/400 System Name, AS/400 Printer Device Name, AS/400 Printer File Library Name*, and *AS/400 Printer File Name* fields of the structure are supplied by the caller when making the verify request. The remaining fields in the structure are supplied by VPRT. The name fields must contain zeros if the name is not specified. The name fields should be padded with zeros for names which do not fill the entire field.

If an AS/400 system name is not specified, the default system name is used and the *AS/400 System Name* field contains the default system name on return. If neither an AS/400 printer device name nor a printer file name is specified, the printer file name QPVPRINT and the printer file library name QIWS are used. The

fields contain these names on return. Both an AS/400 printer device name and a printer file name may be specified.

The *AS/400 Output Queue Name* and *AS/400 Output Queue Library Name* fields can contain the name of the output queue and the output queue library name into which the spooled data of the caller is placed. If the names are not specified, they are returned containing the default output queue name and the output queue library name for the printer or printer file specified.

The following are returned values and are not supplied by the caller:

The returned values for CPI and LPI in the *Default Value* fields are the hexadecimal representation of the decimal number multiplied by ten. For example, the CPI value for 16.7 characters per inch would be returned as the hexadecimal value for 167, or X'A7'.

The returned *Default Value for Defer Printing* is X'01' for Yes and X'02' for No.

The returned *Default Value for Printer Type* is 4 ASCII characters describing the printer type (such as 3812).

The returned *List of Supported LPI and CPI Values* are padded with zeros if the list is not completely filled. The values are represented as described above.

The returned *List of Data Types Supported* are returned as the values 01, 02, 03, 04, or 05. If not filled, the list is padded with zeros.

01 = SCS data
02 = ASCII-to-SCS transform
03 = FFT data
04 = ASCII data
05 = AFPDS data

Assign_VP

This function assigns the virtual printer. This call must be preceded by the *Verify_assign_device* function call. Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of assign data structure

The *Type of Request* field of the request block is to be set to X'1A'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Assign Data Structure Address* field must contain the address of the assign data structure. The *Assign Data Structure for Data Type 1 (SCS Data)* is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
9	Local device name

1	Virtual printer ID
9	AS/400 system name
1	Reserved
11	AS/400 printer device name
1	Reserved
11	AS/400 printer file library name
1	Reserved
11	AS/400 printer file name
1	Reserved
11	AS/400 output queue library name
1	Reserved
11	AS/400 output queue name
1	Reserved
5	AS/400 printer type
1	Reserved
1	Data type
1	Copies
1	Timeout value
1	Defer printing (Schedule-*Immed)
2	Characters per inch
2	Characters per line
1	Lines per inch
1	Page length in lines
1	Lines per page (overflow line)
1	Reserved

The *Assign Data Structure for Data Type 2 (ASCII-to-SCS)* is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
9	Local device name
1	Virtual printer ID
9	AS/400 system name
1	Reserved
11	AS/400 printer device name
1	Reserved
11	AS/400 printer file library name
1	Reserved
11	AS/400 printer file name
1	Reserved
11	AS/400 output queue library name
1	Reserved
11	AS/400 output queue name
1	Reserved
5	AS/400 printer type
1	Reserved
1	Data type
1	Copies
1	Timeout value
1	Defer printing (Schedule-*Immed)
2	Characters per inch
2	Characters per line
1	Lines per inch
1	Page length in lines
1	Lines per page (overflow line)

1	Application formatted data
1	ASCII character set
1	Untranslatable character

The *Assign Data Structure for Data Type 3 (FFT Data)* is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
9	Local device name
1	Virtual printer ID
9	AS/400 system name
1	Reserved
11	AS/400 printer device name
1	Reserved
11	AS/400 printer file library name
1	Reserved
11	AS/400 printer file name
1	Reserved
11	AS/400 output queue library name
1	Reserved
11	AS/400 output queue name
1	Reserved
5	AS/400 printer type
1	Reserved
1	Data type
1	Copies
1	Timeout value
1	Reserved

The *Assign Data Structure for Data Type 4 (ASCII Data)* is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
9	Local device name
1	Virtual printer ID
9	AS/400 system name
1	Reserved
11	AS/400 printer device name
1	Reserved
11	AS/400 printer file library name
1	Reserved
11	AS/400 printer file name
1	Reserved
11	AS/400 output queue library name
1	Reserved
11	AS/400 output queue name
1	Reserved
5	AS/400 printer type
1	Reserved
1	Data type
1	Copies
1	Timeout value
1	Defer printing (Schedule-*Immed)
1	Target operating system
1	Append form feed

The *Assign Data Structure for Data Type 5 (AFPDS Data)* is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
9	Local device name
1	Virtual printer ID
9	AS/400 system name
1	Reserved
11	AS/400 printer device name
1	Reserved
11	AS/400 printer file library name
1	Reserved
11	AS/400 printer file name
1	Reserved
11	AS/400 output queue library name
1	Reserved
11	AS/400 output queue name
1	Reserved
5	AS/400 printer type
1	Reserved
1	Data type
1	Copies
1	Timeout value
1	Defer printing (Schedule-*Immed)

The *Local Device Name* field identifies the device to which the virtual printer is assigned. It must contain an ASCIIZ string name containing the name of the local device being assigned. The possible valid names are: LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9.

The *Virtual Printer ID* field returns a hexadecimal value to be used to reference a specific assigned virtual printer.

The *AS/400 System Name, AS/400 Printer Device Name, AS/400 Printer File Library Name,* and *AS/400 Printer File Name* fields must contain zeros if the name is not specified.

If an *AS/400 System Name* is not specified, the default system name is used and the AS/400 system name field contains the default system on return. If neither an AS/400 printer device name nor a printer file name are specified, the printer file name QPVPRINT and the printer file library QIWS are used. The fields contain these names on return. Both an AS/400 printer device name and a printer file name may be specified.

The *AS/400 Output Queue Name* and *AS/400 Output Queue Library Name* fields indicate the output queue and output queue library name into which the caller's spooled data is placed. If the names are not specified, the output queue default and library name for the specified printer or printer file are used. The value used for the AS/400 printer type is 4 ASCII characters describing the printer type (for example, 3812). This value is returned from the VERIFY_ASSIGN_DEVICE function.

The *Data Type* field must contain one of the following values:

01 = SCS data

02 = ASCII-to-SCS transform
03 = FFT data
04 = ASCII data
05 AFPDS data

The *Copies* field must contain a hexadecimal value for the number of copies requested. Valid values are 1 to 99 for Data Type 03 and 1 to 255 for all other data types.

The *Timeout Value* field must contain a hexadecimal value for the number of seconds the virtual printer is to wait after it stops receiving data and before it closes the AS/400 output file. For example, X'0A' means the virtual printer waits 10 seconds.

The *Defer Printing* field must contain X'01' for Yes and X'02' for No.

The *Characters per Inch* field must contain the hexadecimal value representing the CPI. For example, X'0064' means 10 characters per inch.

The *Characters per Line* field must contain the hexadecimal value representing the number of characters to be printed on each line. For example, X'0050' means 80 characters per line.

The *Lines per Inch* field must contain the hexadecimal value representing the LPI. For example, X'3C' means 6 lines per inch.

The *Page Length in Lines* field must contain the hexadecimal value representing the page length in lines. For example, X'42' means a page length of 66 lines.

The *Lines per Page* field must contain the hexadecimal value representing the number of lines of data to be printed on each page. For example, X'66' means 66 lines printed per page.

The *Application Formatted Data* field contains X'01' for Yes if the data being printed to the virtual printer has been formatted by an application such as DW/4. The field contains X'02' if the data has not been formatted. If X'02' is specified, virtual print ignores the PC commands found in the data which change cpi, lpi, lpp and pl, and instead uses the values chosen when the virtual printer was assigned.

The *ASCII Character Set* field must contain X'01' for Character Set 1 and '02'X for Character Set 2.

The *Untranslatable Character* field must contain the hexadecimal value for an EBCDIC character. This character is be used to replace characters in the data which cannot be found in the ASCII-to-EBCDIC translation table.

The *Target Operating System* field contains one of the following hexadecimal values:

Hex Value	Condition
X'01'	Data will be printed using print emulation in the DOS operating system.
X'02'	Data will be printed using print emulation in the OS/2 operating system.

| X'03' | Data will be printed using print emulation in the DOS/V operating system.

If you do not specify a value for this field, the virtual printer function uses the default value of the current operating system. This field applies only for DBCS.

The *Append Form Feed* field contains one of the following hexadecimal values:

Hex Value	Condition
X'01'	A form feed will be added to the end of each ASCII transparent print job.
X'02'	A form feed will not be added to ASCII transparent jobs.
X'03'	A form feed will be added to the end of ASCII transparent jobs only if the last byte in the job is not a form feed.

Reset_Parms

This function resets one or more of the assign parameters. All parameter fields must be filled in by the caller, even if they are not changing. Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of reset_parms data structure

The *Type of Request* field of the request block must be set to X'1B'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Reset_Parms Data Structure Address* field must contain the address of the Reset_parms data structure. The *Reset_Parms Data Structure* is defined as follows:

Size (Bytes)	Field Description
2	Length of data structure (including this field)
1	Virtual Printer ID
1	Copies
1	Timeout value
1	Defer Printing (Schedule-*Immed)

The *Virtual Printer ID* field must contain the ID previously returned by the assign request.

The *Copies* field must contain a hexadecimal value for the number of copies. Valid values are 1 to 255.

The *Timeout Value* field must contain a hexadecimal value for the number of seconds the virtual printer is to wait after it stops receiving data and before it closes the AS/400 output file.

The *Defer Printing* field must contain X'01' for Yes and X'02' for No.

Write_Data

This function is called to send blocks of data to the virtual printer. Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of Write_data data structure

The *Type of Request* field of the request block must be set to X'1D'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Write_Data Data Structure Address* field must contain the address of the write_data data structure. The *Write_Data Data Structure* is defined as follows:

Size (Bytes)	Field Description
2	Length of request block (including this field)
1	Virtual printer ID
1	Reserved
4	Address of data buffer
2	Data length

The *Virtual Printer ID* field must contain the ID previously returned by the assign request.

The *Data Buffer Address* field must contain the address of the buffer of the data to be written.

The *Data Length* field must contain the length of the data in the buffer. On return, it contains the length of data actually written to the virtual printer.

Release_VP

This function releases a virtual printer. Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of release data structure

The *Type of Request* field of the request block must be set to X'1E'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Release Data Structure Address* field must contain the address of the release data structure. The *Release Data Structure* is defined as follows:

Size (Bytes)	Field Description
2	Length of request block (including this field)
1	Virtual printer ID
1	Reserved

The *Virtual Printer ID* field must contain the ID previously returned by the assign request.

Close_VP

This function closes an open output file. The AS/400 spool file is closed and the virtual printer reset to the assign values. Register ES:BX must contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (including this field)
2	Type of request
4	Address of error message text buffer
4	Address of close data structure

The *Type of Request* field of the request block must be set to X'1F'.

The *Message Buffer Address* field of the request block must contain the address of a 514-byte buffer.

The *Close Data Structure Address* field must contain the address of the close data structure. The *Close Data Structure Address* is defined as follows:

Size (Bytes)	Field Description
2	Length of request block (including this field)
1	Virtual printer ID
1	Reserved

The *Virtual Printer ID* field must contain the ID previously returned by the assign request.

Load_TTable

This function is called to request that VPRT load a new translation table for a given virtual printer. Register ES:BX should contain the address of the following structure:

Size (Bytes)	Field Description
2	Length of request block (Including this field)
2	Type of request
4	Address of error message text buffer
4	Address of Load translation table data structure

The *Type of Request* field of the request block is to be set to X'22'.

The *Message Buffer Address* field of the request block must contain the address of a 514 byte buffer.

The *Load Translation Table Data Structure* field must contain the address of the load translation table data structure.

The *Load Translation Table Data Structure* is defined as follows:

Size (Bytes)	Field Description
2	Length of request block (Including this field)
1	Virtual Printer ID
1	Reserved
4	Address of translation table

The *Virtual Printer ID* field should contain the ID previously returned by the assign request or the query assign parms request.

The *Address of Translation Table* field should contain the address of a valid ASCII-to-EBCDIC translation table.

Part 2. PC Support/400 High-Level Application Program Interfaces for OS/2 and DOS

Chapter 9. PC Support/400 High-Level API Overview for OS/2 and DOS

The PC Support/400 high-level APIs provide a programming interface for client applications running in the DOS and OS/2 environments. These APIs provide access to the low-level PC Support APIs without having to directly manipulate registers and generate software interrupts. Use Part 1, "PC Support Low-Level Application Program Interfaces" in conjunction with this section when developing applications. conjunction with this part when developing applications. Several high-level APIs are available which do not have an associated low interface.

For a description of the PC Support APIs for the Windows environment, see Part 3, "PC Support Windows Application Program Interfaces" on page 18-23.

There are a number of requirements for using these APIs in the DOS environment.

- Applications written to the following APIs must be linked with the EHNALAPI.LIB library found in the QIWSFL2 folder:

  Data transform
  Router
  Shared folders
  Transfer function
  Work station function

- Applications written to the submit remote command API must be linked with the EHNSRAPI.LIB library found in the QIWSFL2 folder for extended DOS and in the QIWSFLR folder for basic DOS.
- Applications written to the remote SQL API must be linked with the EHNRQAPI.LIB library found in the QIWSFL2 folder.
- Applications written to the data queues API must be linked with the EHNDQAPI.LIB library found in the QIWSFL2 folder.
- The DOS high-level APIs can be linked only to real mode applications.

To use these high-level PC Support/400 APIs in a OS/2 environment the following must be done:

- Applications written to the following APIs must be linked with the EHNALAPI.LIB library found in the QIWSOS2 folder:

  Data transform
  Router
  Shared folders
  Transfer function

- Applications written to the submit remote command API must be linked with the EHNSRAPI.LIB library found in the QIWSOS2 folder.
- Applications written to the remote SQL API must be linked with the EHNRQAPI.LIB library found in the QIWSOS2 folder.
- Applications written to the data queues API must be linked with the EHNDQ.LIB library found in the QIWSOS2 folder.
- Applications written to the router API must also be linked with the ACS.LIB library supplied with the OS/2 communications manager.

- Applications written to the work station function API must import the ACS3EHAP.HLLAPI entry point of the EHLLAPI DLL (ACS3EHAP.DLL) supplied with the OS/2 communications manager.
- Applications written to the router API require that the name of the directory that contains the APPC.DLL file be in the user's LIBPATH statement. This DLL is supplied with the OS/2 communications manager.
- The directory name where the following PC Support/400 DLLs (dynamic link libraries) are located must be in the user's LIBPATH statement. These DLLs can be found in the QIWSOS2 folder:

  EHNALAPI.DLL
  EHNCM00.DLL
  EHNDQAPI.DLL
  EHNRQAPI.DLL
  EHNSRAPI.DLL
  EHNTFSTF.DLL

- For applications written to the work station function API, the directory name where the EHLLAPI DLL (ACS3EHAP.DLL) is located must be in the user's LIBPATH statement.

The OS/2 APIs may be called from 16-bit and 32-bit applications. The C header files provided in the PC Support/400 tools folder can be included in both types of applications. Also, most of the sample programs in the tools folder may be built as 16-bit or 32-bit applications. See Appendix C, "PC Support Tools Folder" for additional information.

These APIs support applications are written in the following languages:

IBM C/2
IBM Pascal/2
IBM Macro Assembler/2

Chapter 10. Router High-Level Application Program Interface

This chapter describes the high-level language application program interface (API) for the PC Support router. This API allows you to access the PC Support router verbs from a high-level language. Refer to Chapter 4, "Router Service Verbs and Basic Conversation Verbs" for a description of the router verb structures. These functions may be used in a DOS or an OS/2 environment.

The API supports the following languages:

- IBM Pascal/2
- IBM C/2
- IBM Macro Assembler/2

The library routines described in this chapter are written to support large memory models.

Sample statements, procedure definitions, and data structure definitions for C and Pascal are provided in this chapter. An assembler language program could call the functions by sequence, so no such definitions are given for assembler language.

Sample programs, include files, and other information for using APIs are provided in the PC Support tools folder (QIWSTOOL). See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

DOS Environment

This part defines the PC Support router API for PC programs running in a DOS environment. The PC Support router API is a call and return interface that is designed by masking the software interrupt interface of the PC Support router.

Because the OS/2 EE Communications Manager provides an APPC interface as the standard API, the PC Support router API in a DOS environment is designed with the following characteristics:

- The calling method of the PC Support router API is the same as the OS/2 APPC interface.
- The structure definitions of parameter lists are defined with little difference other than that caused by APPC and PC.
- The symbolic names for constants are identical.

A PC program running on DOS that calls the PC Support router API can be easily migrated to OS/2 operating system with little change.

Note: For information describing the high-level API for the router function for programs running in the Windows environment, see Chapter 20, "Router Windows Application Program Interface."

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the router API:

RTRAPI.H	C language include file
RTRAPI_A.INC	Macro Assembler include file
RTRAPI_T.INC	Pascal language type section
RTRAPI_C.INC	Pascal language constant section
RTRAPI_E.INC	Pascal language external procedure declarations

Functions

The PC Support router API is based on the current verbs of the PC Support router, and it is consistent with the OS/2 APPC programming interface that is provided in the OS/2 EE Communications Manager. These functions are:

Allocate	Allocates a conversation between the PC program and a partner program.
Confirm	Sends a confirmation request to a remote program and waits for a reply.
Confirmed	Sends a confirmation reply to the remote program.
Deallocate	Issued when the program has finished the conversation.
Flush	Flushes the send buffer of the AS/400 router by sending all buffered data to the remote program.
Get Attributes	Returns information about the specified conversation.
Prepare To Receive	Changes the conversation state from send to receive in preparation to receive data.
Receive and Wait	Waits for information to be available for the specified conversation and receives the information.
Receive Immediate	Receives information that is available from the specified conversation, but does not wait for any information to arrive.
Request To Send	Notifies the remote program that the local program is requesting to enter send state.
Send Data	Sends data to the remote program.
Send Error	Detects a remote transaction program error or flush buffer.

The functions are called by giving the operation code to the parameter list while calling the PC Support router API. See Chapter 4, "Router Service Verbs and Basic Conversation Verbs" for a description of the verb structures.

Router Buffer

The PC Support router uses an application supplied buffer to accumulate data received from and sent by the application. Each application has its own buffer, which is supplied to the router on the ALLOCATE request. The buffer must be a minimum of 271 bytes long and cannot cross the 64K boundary.

The router buffer supplied by the application must be in conventional storage; it cannot be in expanded or extended memory. The parameter list for the ALLO-

CATE verb has been extended to provide a method of specifying a buffer address and length.

Formats

With the exception of the Query System Names (QRYSYS) routine, all PC Support router APIs are accessed through one entry point, which is the same as that of the OS/2 APPC Interface. A pointer to a parameter list must be passed to the API, and the requested operation code should be specified in the parameter list.

The format for each supported language is as follows:

C

```
void pascal far router( params_ptr );
char far *params_ptr;
```

PASCAL

```
PROCEDURE router( params_ptr : ADSMEM ) : EXTERN;
```

The ***params_ptr*** is a far address pointing to a function-specific structure called ***parameter list***, which is defined in the following function definitions.

Input and Output

The PC Support router API receives the parameters on call and passes the parameters to the PC Support router. After calling the router, the router API returns to the caller with the values that were returned by the router.

Stepwise, the PC Support router API does the following:

1. Receives control
2. Checks if router is loaded
3. Sets values of parameters to be passed
4. Calls PC Support router by interrupt
5. Returns to caller with returned values

There are two return codes returned by the PC Support router API in the verb record:

- Primary return code: The value returned to the caller. The primary return code is also returned as a value of the function.
- Secondary return code: A value returned to the caller with the primary return code.

Note: All return codes for the router are shown in the same format in which they appear in memory (not Intel format). The constants declared in the include files can be used to test for return code values in the corresponding high-level languages. The constants are coded to take into account the fact that high-level languages expect numbers stored in the Intel format.

The sections on each function describe the procedure and data structure declarations, supplied and returned parameters, and return codes for each function.

For more information on the return codes, see "Verb Return Codes" on page 4-45.

OS/2 Environment

IBM OS/2 EE Communication Manager (CM) provides an APPC interface for PC programs running in an OS/2 environment through dynamic link library (DLL) routines. This part describes the router DLL routines. For more information on using PC Support for OS/2, see the *PC Support/400 OS/2 Installation and Administration Guide*.

Include Files

The following files are provided by OS/2 CM:

APPC_C.H	C language include file
APPC_A.INC	Macro Assembler include file
APPC_TYP.INC	Pascal language type section
APPC_CON.INC	Pascal language constant section
APPC_EXT.INC	Pascal language external procedure declarations

These includes are in path \CMLIB after OS/2 CM is installed.

Functions

The following are the APPC verbs supported in OS/2 CM for both Basic and Mapped conversation mode:

- Allocate
- Confirm
- Confirmed
- Deallocate
- Flush
- Get Attributes
- Prepare To Receive
- Receive and Wait
- Receive Immediate
- Receive and Post
- Request To Send
- Send Data
- Send Error
- Test RTS

OS/2 APPC Format

MACRO ASSEMBLER

```
EXTRN APPC : FAR
```

C

```
extern void pascal far APPC_C(long);
```

PASCAL

```
PROCEDURE APPC_P(CCBPTR : ADSMEM) : EXTERN;
```

OS/2 APPC interface provides all the required verbs in APPC/PC.

Router Library Routines

The following sections for router library routines describe in detail:

- Purpose
- Procedure and data structure declarations
- User-supplied values
- Returned values
- Return codes

ALLOCATE

Purpose

Allocates a conversation between the PC program and a partner program.

On successful completion of the ALLOCATE request, the API returns control to the caller with the Conversation Identifier, which is unique and must be used for subsequent requests to the router.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct allocate
    {
    unsigned char   res1[12];          /* 0  Reserved                 */
    unsigned char   opcode;            /* 12 Operation code           */
    unsigned char   res2[7];           /* 13 Reserved                 */
    unsigned short  primary_rc;        /* 20 Primary return code      */
    unsigned long   secondary_rc;      /* 22 Secondary return code    */
    unsigned char   res3[8];           /* 26 Reserved                 */
    unsigned long   conv_id;           /* 34 Conversation identifier  */
    unsigned char   conv_type;         /* Conversation type           */
                                       /*    AP_BASIC_CONVERSATION    */
                                       /*    AP_MAPPED_CONVERSATION   */
    unsigned char   sync_level;        /* Sync level                  */
                                       /*    AP_NONE                  */
                                       /*    AP_CONFIRM               */
    unsigned char   res4[11];          /* Reserved                    */
    unsigned char   plu_alias[8];      /* Partner LU name             */
    unsigned char   mode_name[8];      /* Mode name                   */
    unsigned char   tpn_length;        /* Length of tp name           */
    unsigned char   tp_name[64];       /* TP name                     */
    unsigned char   res5[34];          /* reserved                    */
    unsigned short  pip_dlen;          /* PIP data length             */
                                       /*   (0 if no pip data)        */
    unsigned char   far *pip_dptr;     /* PIP data                    */
    };

struct allocateinput
    {
    unsigned char      appcid[2];      /* Allocate record identifier */
    unsigned long      *bufferaddress; /* Address of router buffer   */
    unsigned short     bufferlength;   /* Length of router buffer    */
    struct   allocate  alloc;
    };
```

PASCAL

```
TYPE
   unslong = ADS of integer4;

   allocate = RECORD
     res1          [00]: STRING(12);   { 0  Reserved                 }
     opcode        [12]: BYTE;         { 12 Operation code           }
     res2          [13]: STRING(7);    { 13 Reserved                 }
     primary_rc    [20]: WORD;         { 20 Primary return code      }
     secondary_rc  [22]: INTEGER4;     { 22 Secondary return code    }
     res3          [26]: STRING(8);    { 26 Reserved                 }
     conv_id       [34]: INTEGER4;     { 34 Conversation identifier  }
     conv_type     [38]: BYTE;         { Conversation type           }
                                       {    AP_BASIC_CONVERSATION    }
                                       {    AP_MAPPED_CONVERSATION   }
     sync_level    [39]: BYTE;         { Sync level                  }
                                       {    AP_NONE                  }
                                       {    AP_CONFIRM               }
     res4          [40]: STRING(11);   { Reserved                    }
     plu_alias     [51]: STRING(8);    { Partner LU name             }
     mode_name     [59]: STRING(8);    { Mode name                   }
     tpn_length    [67]: BYTE;         { Length of tp name           }
     tp_name       [68]: STRING(64);   { TP name                     }
     res5         [132]: STRING(34);   { Reserved                    }
     pip_dlen     [166]: BYTE;         { PIP data length            }
                                       {   (0 if no pip data)       }
     pip_dptr     [168]: ADSMEM;       { PIP data                   }
   END;

   allocate = RECORD
     appcid        [00]: STRING(2);    { Allocate record identifier  }
     bufferaddress[02]: UNSLONG;       { Address of router buffer    }
     bufferlength [06]: WORD;          { Length of router buffer     }
     alloc        [08]: ALLOCATE
   END;
```

Parameters

opcode — AP_B_ALLOCATE (X'01')

conv_type — One of the following:

AP_BASIC_CONVERSATION (X'00')
AP_MAPPED_CONVERSATION (X'01')

sync_level — One of the following:

AP_NONE (X'00')
AP_CONFIRM (X'01')

plu_alias — Partner LU name (in EBCDIC). If set to all hexadecimal zeros, the default partner name is used.

tp_name — Name of the partner program to start; must be in EBCDIC format.

tpn_length — Length of tp_name.

pip_dlen — 0 if no PIP data. Limited to 128 bytes.

pip_dptr — Address of PIP data if previous field not 0.

appcid	Router allocate record identifier; must be X'AB'
bufferaddress	Address of router buffer.
bufferlength	Length of router buffer; must be at least 271 bytes.
primary_rc	Primary return code
secondary_rc	Secondary return code
conv_id	Conversation identifier.

Return Codes

Return Code	Description
X'0000'	AP_OK - Allocated successfully.
X'0001'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return codes:
X'00000006'	AP_INVALID_DATA_SEGMENT
X'00000010'	AP_TP_NAME_LENGTH_ERROR
X'00000011'	AP_BAD_CONV_TYPE
X'00000012'	AP_BAD_SYNC_LEVEL
X'00000014'	AP_RETURN_CONTROL_NOT_SUPPORTED
X'00000016'	AP_PIP_LEN_INCORRECT
X'00000018'	AP_UNKNOWN_PARTNER_SYSTEM
X'0003'	AP_ALLOCATION_ERROR - Host allocation error. Check the following secondary return codes:
X'00000004'	AP_ALLOCATION_FAILURE_NO_RETRY
X'00000005'	AP_ALLOCATION_FAILURE_RETRY
X'080F6051'	AP_SECURITY_NOT_VALID
X'084B6031'	AP_TRANS_PGM_NOT_AVAIL_RETRY
X'084C0000'	AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
X'10086021'	AP_TP_NAME_NOT_RECOGNIZED
X'10086031'	AP_PIP_NOT_ALLOWED
X'10086032'	AP_PIP_NOT_SPECIFIED_CORRECTLY
X'10086034'	AP_CONVERSATION_TYPE_MISMATCH
X'10086041'	AP_SYNC_LEVEL_NOT_SUPPORTED
X'0014'	AP_UNSUCCESSFUL
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

CONFIRM

Purpose

Sends a confirmation request to a remote program and waits for a reply.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct confirm
    {
    unsigned char   res1[12];          /* 0  Reserved                */
    unsigned char   opcode;            /* 12 Operation code          */
    unsigned char   res2[7];           /* 13 Reserved                */
    unsigned short  primary_rc;        /* 20 Primary return code     */
    unsigned long   secondary_rc;      /* 22 Secondary return code   */
    unsigned char   res3[8];           /* 26 Reserved                */
    unsigned long   conv_id;           /* 34 Conversation identifier */
    unsigned char   rts_rcvd;          /* Request to send received   */
                                       /*    AP_NO                   */
                                       /*    AP_YES                  */

    };
```

PASCAL

```
TYPE
  confirm = RECORD
    res1         [00]: STRING(12);    { 0  Reserved                }
    opcode       [12]: BYTE;          { 12 Operation code          }
    res2         [13]: STRING(7);     { 13 Reserved                }
    primary_rc   [20]: WORD;          { 20 Primary return code     }
    secondary_rc [22]: INTEGER4;      { 22 Secondary return code   }
    res3         [26]: STRING(8);     { 26 Reserved                }
    conv_id      [34]: INTEGER4;      { 34 Conversation identifier }
    rts_rcvd     [38]: BYTE;          { Request to send received   }
                                      {    AP_NO                   }
                                      {    AP_YES                  }

 END;
```

Parameters

opcode	AP_B_CONFIRM (X'03')
conv_id	Conversation identifier returned by the ALLOCATE request
primary_rc	Primary return code
secondary_rc	Secondary return code
rts_rcvd	One of the following: AP_NO (X'00') AP_YES (X'01')

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.

Return Code	Description
X'0001'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return codes:
X'00000002'	AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'00000031'	AP_CONFIRM_ON_SYNC_LEVEL_NONE - The conversation was allocated with AP_SYNC_LEVEL(NONE).
X'0002'	AP_STATE_CHECK - Check the following secondary return code:
X'000000F2'	AP_SEND_DATA_NOT_SEND_STATE - The conversation is not in send state.
X'0003'	AP_ALLOCATION_ERROR - Host allocation error. Check the following secondary return codes:
X'080F6051'	AP_SECURITY_NOT_VALID
X'084B6031'	AP_TRANS_PGM_NOT_AVAIL_RETRY
X'084C0000'	AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
X'10086021'	AP_TP_NAME_NOT_RECOGNIZED
X'10086031'	AP_PIP_NOT_ALLOWED
X'10086032'	AP_PIP_NOT_SPECIFIED_CORRECTLY
X'10086034'	AP_CONVERSATION_TYPE_MISMATCH
X'10086041'	AP_SYNC_LEVEL_NOT_SUPPORTED
X'0006'	AP_DEALLOC_ABEND_PROG
X'0008'	AP_DEALLOC_ABEND_TIMER
X'000E'	AP_PROG_ERROR_PURGING
X'0010'	AP_CONV_FAILURE_NO_RETRY
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

CONFIRMED

Purpose

Sends a confirmation reply to the remote program.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct confirmed
    {
    unsigned char   res1[12];          /* 0  Reserved                */
    unsigned char   opcode;            /* 12 Operation code          */
    unsigned char   res2[7];           /* 13 Reserved                */
    unsigned short  primary_rc;        /* 20 Primary return code     */
    unsigned long   secondary_rc;      /* 22 Secondary return code   */
    unsigned char   res3[8];           /* 26 Reserved                */
    unsigned long   conv_id;           /* 34 Conversation identifier */
    };
```

PASCAL

```
TYPE
  confirmed = RECORD
    res1         [00]: STRING(12);   { 0  Reserved                }
    opcode       [12]: BYTE;         { 12 Operation code          }
    res2         [13]: STRING(7);    { 13 Reserved                }
    primary_rc   [20]: WORD;         { 20 Primary return code     }
    secondary_rc [22]: INTEGER4;     { 22 Secondary return code   }
    res3         [26]: STRING(8);    { 26 Reserved                }
    conv_id      [34]: INTEGER4;     { 34 Conversation identifier }
  END;
```

Parameters

opcode	AP_B_CONFIRMED (X'04')
conv_id	Conversation identifier returned by the ALLOCATE request
primary_rc	Primary return code
secondary_rc	Secondary return code

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001' X'00000002'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return code: AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'0002' X'00000041'	AP_STATE_CHECK - Check the following secondary return code: AP_CONFIRMED_BAD_STATE - The conversation is not in confirmed state.
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

DEALLOCATE

Purpose

Issued when the program has finished the conversation. Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct deallocate
    {
    unsigned char   res1[12];          /* 0  Reserved                */
    unsigned char   opcode;            /* 12 Operation code          */
    unsigned char   res2[7];           /* 13 Reserved                */
    unsigned short  primary_rc;        /* 20 Primary return code     */
    unsigned long   secondary_rc;      /* 22 Secondary return code   */
    unsigned char   res3[8];           /* 26 Reserved                */
    unsigned long   conv_id;           /* 34 Conversation identifier */
    unsigned char   res4;              /* Reserved                   */
    unsigned char   dealloc_type;      /* Type                       */
                                       /*    AP_FLUSH                */
                                       /*    AP_ABEND_PROG           */
    };
```

PASCAL

```
TYPE
  deallocate = RECORD
    res1         [00]: STRING(12);  { 0  Reserved                }
    opcode       [12]: BYTE;        { 12 Operation code          }
    res2         [13]: STRING(7);   { 13 Reserved                }
    primary_rc   [20]: WORD;        { 20 Primary return code     }
    secondary_rc [22]: INTEGER4;    { 22 Secondary return code   }
    res3         [26]: STRING(8);   { 26 Reserved                }
    conv_id      [34]: INTEGER4;    { 34 Conversation identifier }
    res4         [38]: BYTE;        { Reserved                   }
    dealloc_type [39]: BYTE;        { Type                       }
                                    {    AP_FLUSH                }
                                    {    AP_ABEND_PROG           }
  END;
```

Parameters

opcode	AP_B_DEALOCATE (X'05')
conv_id	Conversation identifier returned by the ALLOCATE request
dealloc_type	One of the following: AP_FLUSH (X'01') AP_ABEND_PROG (X'02')
primary_rc	Primary return code
secondary_rc	Secondary return code

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001' X'00000002'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return code: AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'0002' X'000000F2'	AP_STATE_CHECK - Check the following secondary return code: AP_SEND_DATA_NOT_SEND_STATE - The conversation is not in send state.
X'0003' X'080F6051' X'084B6031' X'084C0000' X'10086021' X'10086031' X'10086032' X'10086034' X'10086041'	AP_ALLOCATION_ERROR - Host allocation error. Check the following secondary return codes: AP_SECURITY_NOT_VALID AP_TRANS_PGM_NOT_AVAIL_RETRY AP_TRANS_PGM_NOT_AVAIL_NO_RETRY AP_TP_NAME_NOT_RECOGNIZED AP_PIP_NOT_ALLOWED AP_PIP_NOT_SPECIFIED_CORRECTLY AP_CONVERSATION_TYPE_MISMATCH AP_SYNC_LEVEL_NOT_SUPPORTED
X'0006'	AP_DEALLOC_ABEND_PROG
X'0008'	AP_DEALLOC_ABEND_TIMER
X'0009'	AP_DEALLOC_NORMAL
X'000E'	AP_PROG_ERROR_PURGING
X'000F'	AP_CONV_FAILURE_RETRY
X'0010'	AP_CONV_FAILURE_NO_RETRY
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

FLUSH

Purpose

Flushes the send buffer of the AS/400 router by sending all buffered data to the remote program.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct flush
   {
   unsigned char   res1[12];          /* 0  Reserved                */
   unsigned char   opcode;            /* 12 Operation code          */
   unsigned char   res2[7];           /* 13 Reserved                */
   unsigned short  primary_rc;        /* 20 Primary return code     */
   unsigned long   secondary_rc;      /* 22 Secondary return code   */
   unsigned char   res3[8];           /* 26 Reserved                */
   unsigned long   conv_id;           /* 34 Conversation identifier */
   };
```

PASCAL

```
TYPE
  flush = RECORD
    res1         [00]: STRING(12);   { 0  Reserved                }
    opcode       [12]: BYTE;         { 12 Operation code          }
    res2         [13]: STRING(7);    { 13 Reserved                }
    primary_rc   [20]: WORD;         { 20 Primary return code     }
    secondary_rc [22]: INTEGER4;     { 22 Secondary return code   }
    res3         [26]: STRING(8);    { 26 Reserved                }
    conv_id      [34]: INTEGER4;     { 34 Conversation identifier }
  END;
```

Parameters

opcode	AP_B_FLUSH (X'06')
conv_id	Conversation identifier returned by the ALLOCATE request
primary_rc	Primary return code
secondary_rc	Secondary return code

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001' X'00000002'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return code: AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'0002' X'000000F2'	AP_STATE_CHECK - Check the following secondary return code: AP_SEND_DATA_NOT_SEND_STATE - The conversation is not in send state.
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

GET_ATTRIBUTES

Purpose

Returns information about the specified conversation.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct get_attributes
   {
   unsigned char   res1[12];          /* 0  Reserved                */
   unsigned char   opcode;            /* 12 Operation code          */
   unsigned char   res2[7];           /* 13 Reserved                */
   unsigned short  primary_rc;        /* 20 Primary return code     */
   unsigned long   secondary_rc;      /* 22 Secondary return code   */
   unsigned char   res3[8];           /* 26 Reserved                */
   unsigned long   conv_id;           /* 34 Conversation identifier */
   unsigned char   res4[9];           /* Reserved                   */
   unsigned char   sync_level;        /* Sync level                 */
                                      /*    AP_NONE                 */
                                      /*    AP_CONFIRM              */
   unsigned char   mode_name[8];      /* Mode name                  */
   unsigned char   res5[8];           /* Reserved                   */
   unsigned char   lu_name[8];        /* Own LU name                */
   unsigned char   plu_name[8];       /* Partner LU name            */
   unsigned char   res6[19];          /* Reserved                   */
   unsigned char   userid_Len;        /* Length of user ID          */
   unsigned char   user_id[10];       /* User ID                    */
   };
```

PASCAL

```
TYPE
  get_attributes = RECORD
    res1         [00]: STRING(12);{ 0  Reserved                }
    opcode       [12]: BYTE;      { 12 Operation code          }
    res2         [13]: STRING(7); { 13 Reserved                }
    primary_rc   [20]: WORD;      { 20 Primary return code     }
    secondary_rc [22]: INTEGER4;  { 22 Secondary return code   }
    res3         [26]: STRING(8); { 26 Reserved                }
    conv_id      [34]: INTEGER4;  { 34 Conversation identifier }
    res4         [38]: STRING(9); { Reserved                   }
    sync_level   [47]: BYTE;      { Sync level                }
                                  {    AP_NONE                }
                                  {    AP_CONFIRM             }
    mode_name    [48]: STRING(8); { Mode name                 }
    res5         [56]: STRING(8); { Reserved                  }
    lu_name      [64]: STRING(8); { Own LU name               }
    plu_name     [72]: STRING(8); { Partner LU name           }
    res6         [80]: STRING(19);{ Reserved                  }
    userid_len   [99]: BYTE;      { Length of user ID         }
    user_id     [100]: STRING(10);{ User ID                   }
  END;
```

Parameters

opcode	AP_B_GET_ATTRIBUTES (X'07')
conv_id	Conversation identifier returned by the ALLOCATE request
primary_rc	Primary return code
secondary_rc	Secondary return code
sync_level	One of the following: AP_NONE (X'00') AP_CONFIRM (X'01')
mode_name	The characteristics of the session for the conversation (in EBCDIC)
lu_name	Own LU name
plu_name	Partner LU name
userid_len	Length of user ID
user_id	User ID

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001' X'00000002'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return code: AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

PREPARE_TO_RECEIVE

Purpose

Changes the conversation state from send to receive. Prepare To Receive can be used by the application to change from send state to receive state without waiting for information.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct prepare_to_receive
   {
   unsigned char   res1[12];          /* 0  Reserved                */
   unsigned char   opcode;            /* 12 Operation code          */
   unsigned char   res2[7];           /* 13 Reserved                */
   unsigned short  primary_rc;        /* 20 Primary return code     */
   unsigned long   secondary_rc;      /* 22 Secondary return code   */
   unsigned char   res3[8];           /* 26 Reserved                */
   unsigned long   conv_id;           /* 34 Conversation identifier */
   };
```

PASCAL

```
TYPE
  prepare_to_receive = RECORD
    res1         [00]: STRING(12);{ 0  Reserved                }
    opcode       [12]: BYTE;      { 12 Operation code          }
    res2         [13]: STRING(7); { 13 Reserved                }
    primary_rc   [20]: WORD;      { 20 Primary return code     }
    secondary_rc [22]: INTEGER4;  { 22 Secondary return code   }
    res3         [26]: STRING(8); { 26 Reserved                }
    conv_id      [34]: INTEGER4;  { 34 Conversation identifier }
  END;
```

Parameters

opcode	AP_B_PREPARE_TO_RECEIVE (X'0A')
conv_id	Conversation identifier returned by the ALLOCATE request. Only AP_FLUSH (X'01') is supported.
primary_rc	Primary return code
secondary_rc	Secondary return code

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001' X'00000002'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return code: AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'0002' X'000000F2'	AP_STATE_CHECK - Check the following secondary return code: AP_SEND_DATA_NOT_SEND_STATE - The conversation is not in send state.

Return Code	Description
X'0003'	AP_ALLOCATION_ERROR - Host allocation error. Check the following secondary return codes:
X'080F6051'	AP_SECURITY_NOT_VALID
X'084B6031'	AP_TRANS_PGM_NOT_AVAIL_RETRY
X'084C0000'	AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
X'10086021'	AP_TP_NAME_NOT_RECOGNIZED
X'10086031'	AP_PIP_NOT_ALLOWED
X'10086032'	AP_PIP_NOT_SPECIFIED_CORRECTLY
X'10086034'	AP_CONVERSATION_TYPE_MISMATCH
X'10086041'	AP_SYNC_LEVEL_NOT_SUPPORTED
X'0006'	AP_DEALLOC_ABEND_PROG
X'0008'	AP_DEALLOC_ABEND_TIMER
X'000E'	AP_PROG_ERROR_PURGING
X'000F'	AP_CONV_FAILURE_RETRY
X'0010'	AP_CONV_FAILURE_NO_RETRY
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

RECEIVE_AND_WAIT

Purpose

Waits for information to be available for the specified conversation and receives the information.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct receive_and_wait
   {
   unsigned char   res1[12];          /* 0  Reserved                 */
   unsigned char   opcode;            /* 12 Operation code           */
   unsigned char   res2[7];           /* 13 Reserved                 */
   unsigned short  primary_rc;        /* 20 Primary return code      */
   unsigned long   secondary_rc;      /* 22 Secondary return code    */
   unsigned char   res3[8];           /* 26 Reserved                 */
   unsigned long   conv_id;           /* 34 Conversation identifier  */
   unsigned char   what_rcvd;         /* What received               */
                                      /*    AP_DATA                  */
                                      /*    AP_DATA_COMPLETE         */
                                      /*    AP_DATA_INCOMPLETE       */
                                      /*    AP_SEND                  */
                                      /*    AP_CONFIRM               */
                                      /*    AP_CONFIRM_SEND          */
                                      /*    AP_CONFIRM_DEALLOCATE    */
   unsigned char  fill;               /* Fill                        */
                                      /*    AP_BUFFER                */
                                      /*    AP_LL                    */
   unsigned char  rts_rcvd;           /* Request to send received    */
                                      /*    AP_NO                    */
                                      /*    AP_YES                   */
   unsigned short  max_len;           /* Maximum length              */
   unsigned short  dlen;              /* Received data length        */
   unsigned char   far *dptr;         /* Data address                */
   };
```

PASCAL

```
TYPE
  receive_and_wait = RECORD
    res1         [00]: STRING(12);{ 0  Reserved                 }
    opcode       [12]: BYTE;      { 12 Operation code           }
    res2         [13]: STRING(7); { 13 Reserved                 }
    primary_rc   [20]: WORD;      { 20 Primary return code      }
    secondary_rc [22]: INTEGER4;  { 22 Secondary return code    }
    res3         [26]: STRING(8); { 26 Reserved                 }
    conv_id      [34]: INTEGER4;  { 34 Conversation identifier  }
    what_rcvd    [38]: BYTE;      { What received               }
                                  {    AP_DATA                  }
                                  {    AP_DATA_COMPLETE         }
                                  {    AP_DATA_INCOMPLETE       }
                                  {    AP_SEND                  }
                                  {    AP_CONFIRM               }
                                  {    AP_CONFIRM_SEND          }
                                  {    AP_CONFIRM_DEALLOCATE    }
    fill         [39]: BYTE;      { Fill                        }
                                  {    AP_BUFFER                }
                                  {    AP_LL                    }
    rts_rcvd     [40]: BYTE;      { Request to send received    }
                                  {    AP_NO                    }
                                  {    AP_YES                   }
    max_len      [41]: WORD;      { Maximum length              }
    dlen         [43]: WORD;      { Received data length        }
    dptr         [45]: ADSMEM;    { Data address                }
  END;
```

Parameters

opcode	AP_B_RECEIVE_AND_WAIT (X'0B')
conv_id	Conversation identifier returned by the ALLOCATE request
fill	One of the following: AP_BUFFER (X'00') AP_LL (X'01')
max_len	Maximum amount of data to receive
dptr	Pointer to buffer to receive the data
primary_rc	Primary return code
secondary_rc	Secondary return code
what_rcvd	One of the following: AP_DATA (X'00') AP_DATA_COMPLETE (X'01') AP_DATA_INCOMPLETE (X'02') AP_CONFIRM_WHAT_RECEIVED (X'03') AP_CONFIRM_SEND (X'04') AP_CONFIRM_DEALLOCATE (X'05') AP_SEND (X'06')
dlen	Length of data received
rts_rcvd	One of the following: AP_NO (X'00') AP_YES (X'01')

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return code:
X'00000002'	AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'00000006'	AP_INVALID_DATA_SEGMENT - Data area crosses segment boundary.
X'0002'	AP_STATE_CHECK - Check the following secondary return code:
X'000000F2'	AP_SEND_DATA_NOT_SEND_STATE - The conversation is not in send state.
X'0003'	AP_ALLOCATION_ERROR - Host allocation error. Check the following secondary return codes:
X'080F6051'	AP_SECURITY_NOT_VALID
X'084B6031'	AP_TRANS_PGM_NOT_AVAIL_RETRY
X'084C0000'	AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
X'10086021'	AP_TP_NAME_NOT_RECOGNIZED
X'10086031'	AP_PIP_NOT_ALLOWED
X'10086032'	AP_PIP_NOT_SPECIFIED_CORRECTLY
X'10086034'	AP_CONVERSATION_TYPE_MISMATCH
X'10086041'	AP_SYNC_LEVEL_NOT_SUPPORTED

Return Code	Description
X'0006'	AP_DEALLOC_ABEND_PROG
X'0008'	AP_DEALLOC_ABEND_TIMER
X'0009'	AP_DEALLOC_NORMAL
X'000C'	AP_PROG_ERROR_NO_TRUNC
X'000D'	AP_PROG_ERROR_TRUNC
X'000E'	AP_PROG_ERROR_PURGING
X'000F'	AP_CONV_FAILURE_RETRY
X'0010'	AP_CONV_FAILURE_NO_RETRY
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

RECEIVE_IMMEDIATE

Purpose

Receives information that is available from the specified conversation, but does not wait for any information to arrive.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct receive_immediate
   {
   unsigned char   res1[12];          /* 0  Reserved                */
   unsigned char   opcode;            /* 12 Operation code          */
   unsigned char   res2[7];           /* 13 Reserved                */
   unsigned short  primary_rc;        /* 20 Primary return code     */
   unsigned long   secondary_rc;      /* 22 Secondary return code   */
   unsigned char   res3[8];           /* 26 Reserved                */
   unsigned long   conv_id;           /* 34 Conversation identifier */
   unsigned char   what_rcvd;         /* What received              */
                                      /*    AP_DATA                 */
                                      /*    AP_DATA_COMPLETE        */
                                      /*    AP_DATA_INCOMPLETE      */
                                      /*    AP_SEND                 */
                                      /*    AP_CONFIRM              */
                                      /*    AP_CONFIRM_SEND         */
                                      /*    AP_CONFIRM_DEALLOCATE   */
   unsigned char  fill;               /* Fill                       */
                                      /*    AP_BUFFER               */
                                      /*    AP_LL                   */
   unsigned char  rts_rcvd;           /* Request to send received   */
                                      /*    AP_NO                   */
                                      /*    AP_YES                  */
   unsigned short  max_len;           /* Maximum length             */
   unsigned short  dlen;              /* Data length                */
   unsigned char   far *dptr;         /* Data address               */
   };
```

PASCAL

```
TYPE
  receive_immediate = RECORD
    res1          [00]: STRING(12);{ 0  Reserved                }
    opcode        [12]: BYTE;      { 12 Operation code          }
    res2          [13]: STRING(7); { 13 Reserved                }
    primary_rc    [20]: WORD;      { 20 Primary return code     }
    secondary_rc  [22]: INTEGER4;  { 22 Secondary return code   }
    res3          [26]: STRING(8); { 26 Reserved                }
    conv_id       [34]: INTEGER4;  { 34 Conversation identifier }
    what_rcvd     [38]: BYTE;      { What received              }
                                   {    AP_DATA                 }
                                   {    AP_DATA_COMPLETE        }
                                   {    AP_DATA_INCOMPLETE      }
                                   {    AP_SEND                 }
                                   {    AP_CONFIRM              }
                                   {    AP_CONFIRM_SEND         }
                                   {    AP_CONFIRM_DEALLOCATE   }
    fill          [39]: BYTE;      { Fill                       }
                                   {    AP_BUFFER               }
                                   {    AP_LL                   }
    rts_rcvd      [40]: BYTE;      { Request to send received   }
                                   {    AP_NO                   }
                                   {    AP_YES                  }
    max_len       [41]: WORD;      { Maximum length             }
    dlen          [43]: WORD;      { Data length                }
    dptr          [45]: ADSMEM;    { Data address               }
  END;
```

Parameters

opcode	AP_B_RECEIVE_IMMEDIATE (X'0C')
conv_id	Conversation identifier returned by the ALLOCATE request
fill	One of the following: AP_BUFFER (X'00') AP_LL (X'01')
max_len	Maximum amount of data to receive
dptr	Pointer to data buffer to receive data
primary_rc	Primary return code
secondary_rc	Secondary return code
dlen	Length of data received
what_rcvd	One of the following: AP_DATA (X'00') AP_DATA_COMPLETE (X'01') AP_DATA_INCOMPLETE (X'02') AP_CONFIRM_WHAT_RECEIVED (X'03') AP_CONFIRM_SEND (X'04') AP_CONFIRM_DEALLOCATE (X'05') AP_SEND (X'06')
rts_rcvd	One of the following: AP_NO (X'00') AP_YES (X'01')

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return codes:
X'00000002'	AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'00000006'	AP_INVALID_DATA_SEGMENT - Data area crosses segment boundary.
X'0002'	AP_STATE_CHECK - Check the following secondary return code:
X'000000C1'	AP_RCV_IMMD_BAD_STATE - The conversation is not in receive state.
X'0003'	AP_ALLOCATION_ERROR - Host allocation error. Check the following secondary return codes:
X'080F6051'	AP_SECURITY_NOT_VALID
X'084B6031'	AP_TRANS_PGM_NOT_AVAIL_RETRY
X'084C0000'	AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
X'10086021'	AP_TP_NAME_NOT_RECOGNIZED
X'10086031'	AP_PIP_NOT_ALLOWED
X'10086032'	AP_PIP_NOT_SPECIFIED_CORRECTLY
X'10086034'	AP_CONVERSATION_TYPE_MISMATCH
X'10086041'	AP_SYNC_LEVEL_NOT_SUPPORTED

Return Code	Description
X'0006'	AP_DEALLOC_ABEND_PROG
X'0008'	AP_DEALLOC_ABEND_TIMER
X'0009'	AP_DEALLOC_NORMAL
X'000C'	AP_PROG_ERROR_NO_TRUNC
X'000D'	AP_PROG_ERROR_TRUNC
X'000E'	AP_PROG_ERROR_PURGING
X'000F'	AP_CONV_FAILURE_RETRY
X'0010'	AP_CONV_FAILURE_NO_RETRY
X'0014'	AP_UNSUCCESSFUL
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

REQUEST_TO_SEND

Purpose

Notifies the remote program that the local program is requesting to enter the send state.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct request_to_send
   {
   unsigned char   res1[12];          /* 0  Reserved                */
   unsigned char   opcode;            /* 12 Operation code          */
   unsigned char   res2[7];           /* 13 Reserved                */
   unsigned short  primary_rc;        /* 20 Primary return code     */
   unsigned long   secondary_rc;      /* 22 Secondary return code   */
   unsigned char   res3[8];           /* 26 Reserved                */
   unsigned long   conv_id;           /* 34 Conversation identifier */
   };
```

PASCAL

```
TYPE
  request_to_send = RECORD
    res1          [00]: STRING(12);{ 0  Reserved                }
    opcode        [12]: BYTE;      { 12 Operation code          }
    res2          [13]: STRING(7); { 13 Reserved                }
    primary_rc    [20]: WORD;      { 20 Primary return code     }
    secondary_rc  [22]: INTEGER4;  { 22 Secondary return code   }
    res3          [26]: STRING(8); { 26 Reserved                }
    conv_id       [34]: INTEGER4;  { 34 Conversation identifier }
  END;
```

Parameters

opcode	AP_B_REQUEST_TO_SEND (X'0E')
conv_id	Conversation identifier returned by the ALLOCATE request
primary_rc	Primary return code
secondary_rc	Secondary return code

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001' X'00000002'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return code: AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'0002' X'000000E1'	AP_STATE_CHECK - Check the following secondary return code: AP_R_T_S_BAD_STATE - The conversation is in the wrong state.
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

SEND_DATA

Purpose

Sends data to the remote program. A send with a length of 0 is valid; the router checks for the returned parameter rts_rcvd, and error conditions without any data.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct send_data
   {
   unsigned char   res1[12];          /* 0  Reserved                */
   unsigned char   opcode;            /* 12 Operation code          */
   unsigned char   res2[7];           /* 13 Reserved                */
   unsigned short  primary_rc;        /* 20 Primary return code     */
   unsigned long   secondary_rc;      /* 22 Secondary return code   */
   unsigned char   res3[8];           /* 26 Reserved                */
   unsigned long   conv_id;           /* 34 Conversation identifier */
   unsigned char   rts_rcvd;          /* Request to send received   */
                                      /*    AP_NO                   */
                                      /*    AP_YES                  */
   unsigned char   res4;              /* Reserved                   */
   unsigned short  dlen;              /* Data length                */
   unsigned char   far *dptr;         /* Data address               */
   };
```

PASCAL

```
TYPE
  send_data = RECORD
    res1          [00]: STRING(12);{ 0  Reserved                }
    opcode        [12]: BYTE;      { 12 Operation code          }
    res2          [13]: STRING(7); { 13 Reserved                }
    primary_rc    [20]: WORD;      { 20 Primary return code     }
    secondary_rc  [22]: INTEGER4;  { 22 Secondary return code   }
    res3          [26]: STRING(8); { 26 Reserved                }
    conv_id       [34]: INTEGER4;  { 34 Conversation identifier }
    rts_rcvd      [38]: BYTE;      { Request to send received   }
                                   {    AP_NO                   }
                                   {    AP_YES                  }
    res4          [39]: BYTE;      { Reserved                   }
    dlen          [40]: WORD;      { Data length                }
    dptr          [42]: ADSMEM;    { Data address               }
  END;
```

Parameters

opcode	AP_B_SEND_DATA (X'0F')
conv_id	Conversation identifier returned by the ALLOCATE request
dlen	Amount of data to send
dptr	Pointer to the data to be sent
primary_rc	Primary return code
secondary_rc	Secondary return code
rts_rcvd	One of the following: AP_NO (X'00') AP_YES (X'01')

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001' X'00000002' X'00000006'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return codes: AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier. AP_INVALID_DATA_SEGMENT - Data area crosses segment boundary.
X'0002' X'000000F2'	AP_STATE_CHECK - Check the following secondary return code: AP_SEND_DATA_NOT_SEND_STATE - The conversation is not in send state.
X'0003' X'080F6051' X'084B6031' X'084C0000' X'10086021' X'10086031' X'10086032' X'10086034' X'10086041'	AP_ALLOCATION_ERROR - Host allocation error. Check the following secondary return codes: AP_SECURITY_NOT_VALID AP_TRANS_PGM_NOT_AVAIL_RETRY AP_TRANS_PGM_NOT_AVAIL_NO_RETRY AP_TP_NAME_NOT_RECOGNIZED AP_PIP_NOT_ALLOWED AP_PIP_NOT_SPECIFIED_CORRECTLY AP_CONVERSATION_TYPE_MISMATCH AP_SYNC_LEVEL_NOT_SUPPORTED
X'0006'	AP_DEALLOC_ABEND_PROG
X'0008'	AP_DEALLOC_ABEND_TIMER
X'0009'	AP_DEALLOC_NORMAL
X'000E'	AP_PROG_ERROR_PURGING
X'000F'	AP_CONV_FAILURE_RETRY
X'0010'	AP_CONV_FAILURE_NO_RETRY
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

SEND_ERROR

Purpose

Detects a remote TP error or flush buffer.

Procedure Declaration

Refer to "DOS Environment" on page 10-1 and "OS/2 Environment" on page 10-4 for the include files and formats to use.

C

```
struct send_error
   {
   unsigned char   res1[12];          /* 0  Reserved                */
   unsigned char   opcode;            /* 12 Operation code          */
   unsigned char   res2[7];           /* 13 Reserved                */
   unsigned short  primary_rc;        /* 20 Primary return code     */
   unsigned long   secondary_rc;      /* 22 Secondary return code   */
   unsigned char   res3[8];           /* 26 Reserved                */
   unsigned long   conv_id;           /* 34 Conversation identifier */
   unsigned char   rts_rcvd;          /* Request to send received   */
                                      /*    AP_NO                   */
                                      /*    AP_YES                  */

   };
```

PASCAL

```
TYPE
  send_error = RECORD
    res1          [00]: STRING(12);{ 0  Reserved                }
    opcode        [12]: BYTE;      { 12 Operation code          }
    res2          [13]: STRING(7); { 13 Reserved                }
    primary_rc    [20]: WORD;      { 20 Primary return code     }
    secondary_rc  [22]: INTEGER4;  { 22 Secondary return code   }
    res3          [26]: STRING(8); { 26 Reserved                }
    conv_id       [34]: INTEGER4;  { 34 Conversation identifier }
    rts_rcvd      [38]: BYTE;      { Request to send received   }
                                   {    AP_NO                   }
                                   {    AP_YES                  }
  END;
```

Parameters

opcode	AP_B_SEND_ERROR (X'10')
conv_id	Conversation identifier returned by the ALLOCATE request. Only AP_PROG (X'00') is supported.
primary_rc	Primary return code
secondary_rc	Secondary return code
rts_rcvd	One of the following: AP_NO (X'00') AP_YES (X'01')

Return Codes

Return Code	Description
X'0000'	AP_OK - Remote program replied CONFIRMED.
X'0001' X'00000002'	AP_PARAMETER_CHECK - Error in the parameter list. Check the following secondary return code: AP_BAD_CONV_ID - The router does not recognize the specified conversation identifier.
X'0002' X'00000101'	AP_STATE_CHECK - Check the following secondary return code: AP_SEND_ERROR_NOT_ALLOWED - The conversation is in reset state.

Return Code	Description
X'0003'	AP_ALLOCATION_ERROR - Host allocation error. Check the following secondary return codes:
X'080F6051'	AP_SECURITY_NOT_VALID
X'084B6031'	AP_TRANS_PGM_NOT_AVAIL_RETRY
X'084C0000'	AP_TRANS_PGM_NOT_AVAIL_NO_RETRY
X'10086021'	AP_TP_NAME_NOT_RECOGNIZED
X'10086031'	AP_PIP_NOT_ALLOWED
X'10086032'	AP_PIP_NOT_SPECIFIED_CORRECTLY
X'10086034'	AP_CONVERSATION_TYPE_MISMATCH
X'10086041'	AP_SYNC_LEVEL_NOT_SUPPORTED
X'0006'	AP_DEALLOC_ABEND_PROG
X'0008'	AP_DEALLOC_ABEND_TIMER
X'0009'	AP_DEALLOC_NORMAL
X'000E'	AP_PROG_ERROR_PURGING
X'000F'	AP_CONV_FAILURE_RETRY
X'0010'	AP_CONV_FAILURE_NO_RETRY
X'F002'	AP_APPC_BUSY - Router was servicing another request. This return code is possible when performing two or more tasks at the same time.
X'FEFF'	AP_ROUTER_NOT_LOADED - The router is not loaded.

For more information about the return codes, see "Verb Return Codes" on page 4-45.

Service Verbs

Query System Names (QRYSYS)

Purpose

Queries the PC Support router to get a list of system names. The router could contain up to 32 different system names.

Procedure Declaration

C

```
#include <RTRAPI.H>
unsigned pascal qrysys(nameptr)
char far *nameptr;      /* Far address pointing to buffer that contains */
                        /* system name                                  */
```

PASCAL

```
{$include: 'RTRAPI_C.INC'} { Constant section include file}
{$include: 'RTRAPI_T.INC'} { Type section include file    }
{$include: 'RTRAPI_E.INC'} { External procedure declarations include file }
FUNCTION QRYSYS(nameptr : ADSMEM) : WORD;
```

Parameters

nameptr — On completion of the call, this variable contains a pointer to a buffer that has at least 258 bytes with the following offsets:

- Bytes 0 to 1: Number of system names.
- Bytes 2 to 257: System names, 8 characters each.

Return Codes

Return Code	Description
X'0000'	AP_OK - Successful.
X'FEFF'	AP_ROUTER_NOT_LOADED - PC Support router is not loaded.

Remote Program Start (EHNAPPC_RemoteProgramStart)

Purpose

Start a program on a remote AS/400 system.

Note: This API is available only in the OS/2 environment.

Procedure Declaration

C

```
#include <RTRAPI.H>
unsigned far pascal EHNAPPC_RemoteProgramStart(HWND hWindow,
                                               char far *lpszHostSystemName,
                                               char far *lpszHostProgramName,
                                               char far *lpszHostLibraryName,
                                               char far *lpchPipData,
                                               unsigned short wPipDataLength);
```

Parameters

hWindow — Identifies the Window of the calling application. If this is not a Presentation Manager* application, the value should be NULL.

lpszHostSystemName — Contains the name of the remote system. The default system will be used if the value in NULL. This must be an ASCIIZ string of up to eight characters in length.

lpszHostProgramName — Contains the name of the host program that is to be started. This must be an ASCIIZ string.

lpszHostLibraryName — Contains the library name for the host program. The default library list is used if this value is set to NULL. This must be an ASCIIZ string.

lpchPipData — Contains the program parameters for the host program. No data is sent if the value is NULL.

Note: Only one parameter may be sent.

wPipDataLength — Contains the length of the PIP data.

Return Codes

The return code for this function is the primary return code from the router APIs. If the functions fails because of an error in the conversion to EBCIDIC, the return code will be EHNRT_UNSUCCESSFUL.

Chapter 11. Transfer Function High-Level Application Program Interface

This chapter describes the high-level language API for the PC Support transfer function. These functions may be used in a DOS or an OS/2 environment.

The API supports the following languages:

- IBM Pascal/2
- IBM C/2
- IBM Macro Assembler/2

The library routines described in this chapter are written to support large memory models.

Sample statements, procedure definitions, or data structure definitions for C and Pascal are provided in this chapter. An assembler language program could call the functions by sequence, so there are no such definitions given for assembler language.

Sample programs, include files, and other information for using APIs are provided in the PC Support tools folder (QIWSTOOL). See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

DOS Environment

This section defines the high-level language API for PC programs running in a DOS environment.

Note: For information describing the high-level API for the transfer function for programs running in the Windows environment, see Chapter 21, "Transfer Function Windows Application Program Interface."

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the transfer function API:

STFAPI.H	C language include file
STFAPI_A.INC	Macro Assembler include file
STFAPI_T.INC	Pascal language type section
STFAPI_C.INC	Pascal language constant section
STFAPI_E.INC	Pascal language external procedure declarations

Functions

The transfer function API is based on the current functions of STF.EXE. These functions are:

- Open a transfer request
- Retrieve the templates
- Retrieve the records
- Close the transfer request
- End all transfer request conversations
- Send records

- End the transfer request conversation

Purpose

Transfers data from a file on a personal computer to an AS/400 file member or from an AS/400 file member to a file on a personal computer.

Procedure Declaration

The declarations for all functions are the same.

C

Statements to use in application programs:

```
#include <stfapi.h>

unsigned char func;      /* A 1-byte value, from 01 to 07, specifying  */
                         /* the code for the function to perform.      */
struct stf far *params;  /* Parameters */
```

Declarations provided in include files:

```
unsigned pascal stfapi(func,params);

struct stf {
          char far *buffaddr;    /* Far address pointing to the transfer */
                                 /* request buffer.                      */
          unsigned bufflen;      /* The length of the transfer request   */
                                 /* buffer (must be 4096).               */
          unsigned reqlen;       /* The length of data being sent.       */
          char far *sysaddr;     /* Far address pointing to name of the  */
                                 /* system in which your request is run. */
          unsigned char conver;  /* Type of translation.                 */
          char far *retaddr;     /* Far address pointing to the address  */
                                 /* within the transfer request buffer.  */
          unsigned retlen;       /* The length of data returned, or      */
                                 /* the number of templates that can be  */
                                 /* retrieved.                           */
          unsigned retcode2;     /* A secondary return code, or          */
                                 /* the number of templates that may be  */
                                 /* retrieved if a warning message       */
                                 /* occurred during the open of a        */
                                 /* REPLACE transfer request.            */
} ;
```

PASCAL

Statements to use in application programs:

```
{$include: 'stfapi_c.inc' } { Constant section include file                }
{$include: 'stfapi_t.inc' } { Type section include file                    }
{$include: 'stfapi_e.inc' } { External procedures declarations include file }
```

Declarations provided in include files:

```
FUNCTION stfapi(func : BYTE ,params : stf_ptr) : WORD;

TYPE
  stf = RECORD
     buffaddr  [00]: ADSMEM;    { Far address pointing to the          }
                                { transfer request buffer.             }
     bufflen   [04]: WORD;      { The length of the transfer request   }
                                { buffer (must be 4096).               }
     reqlen    [06]: WORD;      { The length of data being sent.       }
     sysaddr   [08]: ADSMEM;    { Address of system name.              }
     conver    [12]: BYTE;      { Type of translation.                 }
     retaddr   [13]: ADSMEM;    { Address within the buffer.           }
     retlen    [17]: WORD;      { The length of data returned, or      }
                                { the number of templates that can be  }
                                { retrieved.                           }
     retcode2  [19]: WORD;      { A secondary return code, or          }
                                { the number of templates that may     }
                                { be retrieved if a warning message    }
                                { occurred during the open of a        }
                                { REPLACE transfer request.            }
  END;
 stf_ptr          = ADS of stf;
 stf_buffer       = ARRAY[1..4096] OF BYTE;
 stf_buffer_ptr   = ADS of stf_buffer;
 stf_sysname      = STRING(16);
 stf_sysname_ptr  = ADS of stf_sysname_ptr;
```

Parameters

buffaddr	Far address pointing to the transfer request buffer.
bufflen	The length of the transfer request buffer, which must be 4096.
reqlen	The length of data being sent.
sysaddr	Far address pointing to the name of the system on which your request will run.
conver	Type of translation.
stfapi	Returns 0 if request is successful.
retaddr	Returns the far address pointing to the address within the transfer request buffer.
retlen	Returns the length of data returned, or a count of the number of templates that can be retrieved.
retcode2	A secondary return code.

Additional Information

STF.EXE must be loaded before making a function call. If STF.EXE is not loaded, a return code of X'00FE' is issued.

The value of **func** specifies the function to perform and is as follows:

STF_OPEN_TRANSFER(X'01')
: Open a transfer request function

STF_RETRIEVE_TEMPLATE(X'02')
: Retrieve the templates function

STF_RETRIEVE_RECORD(X'03')
Retrieve the records function

STF_CLOSE_TRANSFER(X'04')
Close the transfer request function

STF_END_ALL_REQUEST(X'05')
End all transfer request conversations function

STF_SEND_RECORD(X'06')
Send records function

STF_END_REQUEST(X'07')
End the transfer request conversation function

When **func** is not in the range X'01' to X'07', the API returns X'2020'.

If STF.EXE is loaded and the function code is valid, the transfer function API receives the parameters on call and then passes them to STF.EXE.

Stepwise, the transfer function API does the following:

1. Receives control.
2. Sets registers to received parameters.
3. Calls STF by interrupt.
4. Sets output parameters to output registers.
5. Returns to caller. The return code in AX from STF is kept.

During step 2, the API puts the following values in the input registers to STF.EXE:

AL	func
AH	conver
DS: BX	buffaddr
ES: DI	sysaddr
CX	reqlen
DX	bufflen

During step 4, the API puts the following values in the output parameters to the caller:

retaddr	DS: BX
retlen	CX
retcode2	DX

During step 5, the primary return code is returned in the AX register:

Return Codes

Return Code	Description
X'0000'	STF_SUCCESS - Successful completion.
X'00FE'	STF_NOT_LOADED - STF not loaded.
X'0103'	STF_REQUEST_ENDED - Transfer request is already ended.
X'0300'	STF_UNTRANS_DATA_FOUND - Untranslatable data found in transferred record.
X'0302'	STF_UNTRANS_DATA_SENT - Untranslatable data in record to be transferred to the AS/400 system.

Return Code	Description
X'0400'	STF_WARN_DETECTED_BY_AS400 - Warning detected by the AS/400 system.
X'1FFF'	STF_END_OF_FILE - End of file.
X'2000'	STF_MAX_TRANS_ACTIVE - Maximum number of transfer requests active.
X'2001'	STF_INVALID_REQUEST_LENGTH - Transfer request length not valid.
X'2002'	STF_CHAR_NOT_TRANSLATED - Transfer request character cannot be translated.
X'2003'	STF_REC_NOT_TRANSFERED - Record not transferred to AS/400 system due to untranslatable numeric data.
X'2004'	STF_NUMERIC_NOT_FIT - Numeric data does not fit into range specified by digits.
X'2005'	STF_INVALID_REC_LENGTH - Record length given by your application program is incorrect.
X'2006'	STF_STACK_TOO_SMALL - Program stack is not large enough.
X'2010'	STF_REQUEST_NOT_OPENED - Transfer request not opened.
X'2011'	STF_TEMPLATE_NOT_RETRIEVE - Template cannot be retrieved now.
X'2012'	STF_GET_REC_FAIL - Cannot get records on a Personal Computer-to-AS/400 request.
X'2013'	STF_SEND_REC_FAIL - Cannot send records on an AS/400-to-Personal Computer request.
X'2020'	STF_INVALID_REQUEST - Incorrect function requested.
X'2021'	STF_INVALID_BUF_LENGTH - Incorrect buffer length.
X'2022'	STF_INVALID_BUF_ADDR - Incorrect buffer address.
X'2023'	STF_REQUEST_BUF_OVERLAPPED - Transfer request buffer overlaps a previous transfer request buffer.
X'2024'	STF_INCORRECT_BUFADDR - Incorrect address specified for the buffer address parameter.
X'2025'	STF_INVALID_RETURN_ADDR - Incorrect address specified for the return address parameter.
X'2026'	STF_INVALID_RETURN_LENGTH_ADDR - Incorrect address specified for the return length parameter.
X'2027'	STF_INVALID_2ND_RETURN_ADDR - Incorrect address specified for the secondary return code parameter.
X'2028'	STF_INVALID_SYSTEM_ADDR - Incorrect address specified for the system address parameter.
X'3000'	STF_ERROR_W_STATE - Error detected by the AS/400 system; statement offset was provided.
X'3100'	STF_ERROR_WO_STATE - Error detected by the AS/400 system; no statement offset was provided.
X'5048'	STF_CONNECT_FAILED - Connection failed to system &&&&&&&&.
X'5050'	STF_INVALID_SYSTEM_NAME - System name is incorrect or inactive.
X'5052'	STF_PROGRAM_NOT_FOUND - System &&&&&&&& &&&&&&&&&&&&&&&&&&&&&&&&&&&& program not found.

Return Code	Description
X'5054'	STF_PROGRAM_ENDED_ABNORMALLY - System &&&&&&&& &&&&&&&&&&&&&&&&&&&&&&&&&&&& program ended unexpectedly.
X'5056'	STF_SECURITY_ERROR - Security error occurred for system &&&&&&&&.
X'5058'	STF_SYSTEM_NOT_SUPPORTED - System &&&&&&&& is not a supported system.
X'5060'	STF_INCORRECT_RELEASE - Cannot continue: &&&&&&&& must be Release &.& Level &&.
X'5062'	STF_CONTACT_ENDED - Contact with system &&&&&&&& ended.
X'5064'	STF_CONTACT_INTERRUPTED - Contact with system &&&&&&&& temporarily interrupted.
X'5066'	STF_RESOURCE_FAILED - Resource failure on system &&&&&&&&.
X'5067'	STF_CONVERSATION_ENDED_ABNOR - Conversation with &&&&&&&& was ended unexpectedly.
X'5400'	STF_OS_ERROR - Operating system returned error code &&&&.
X'9999'	STF_UNDEFINED_ERROR - Undefined error. Call your service representative.

OS/2 Environment

IBM OS/2 EE Communication Manager (CM) provides an APPC interface for PC programs running in an OS/2 environment through dynamic link library (DLL) routines. This section describes the transfer function DLL routines.

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the transfer function API:

STFAPI.H	C language include file
STFAPI_A.INC	Macro assembler include file
STFAPI_T.INC	Pascal language type section
STFAPI_C.INC	Pascal language constant section
STFAPI_E.INC	Pascal language external procedure declarations

Functions

The transfer function API is based on the current functions of EHNTFSTF.DLL. These functions are:

- Open a transfer request
- Retrieve the templates
- Retrieve the records
- Close the transfer request
- End all transfer request conversations
- Send records
- End the transfer request conversation

Procedure Declaration

The declarations for all functions are the same.

C

Statements to use in application programs:

```
#include <stfapi.h>

unsigned char func;      /* A 1-byte value, from 01 to 07, specifying  */
                         /* the code for the function to perform.      */
struct stf far *params;  /* Parameters */
```

Declarations provided in include files:

```
unsigned pascal stfapi(func,params);

struct stf {
          char far *buffaddr;    /* Far address pointing to the transfer */
                                 /* request buffer.                      */
          unsigned bufflen;      /* The length of the transfer request   */
                                 /* buffer (must be 4096).               */
          unsigned reqlen;       /* The length of data being sent.       */
          char far *sysaddr;     /* Far address pointing to name of the  */
                                 /* system in which your request is run. */
          unsigned char conver;  /* Type of translation.                 */
          char far *retaddr;     /* Far address pointing to the address  */
                                 /* within the transfer request buffer.  */
          unsigned retlen;       /* The length of data returned, or      */
                                 /* the number of templates that can be  */
                                 /* retrieved.                           */
          unsigned retcode2;     /* A secondary return code, or          */
                                 /* the number of templates that may be  */
                                 /* retrieved if a warning message       */
                                 /* occurred during the open of a        */
                                 /* REPLACE transfer request.            */
} ;
```

PASCAL

Statements to use in application programs:

```
{$include: 'stfapi_c.inc' } { Constant section include file                }
{$include: 'stfapi_t.inc' } { Type section include file                    }
{$include: 'stfapi_e.inc' } { External procedures declarations include file }
```

Declarations provided in include files:

```
FUNCTION stfapi(func : BYTE ,params : @stf) : WORD;

TYPE
  stf = RECORD
     buffaddr      : ADSMEM;    { Far address pointing to the          }
                                { transfer request buffer.             }
     bufflen       : WORD;      { The length of the transfer request   }
                                { buffer (must be 4096).               }
     reqlen        : WORD;      { The length of data being sent.       }
     sysaddr       : ADSMEM;    { Address of system name.              }
     conver        : BYTE;      { Type of translation.                 }
     retaddr       : ADSMEM;    { Address within the buffer.           }
     retlen        : WORD;      { The length of data returned, or      }
                                { the number of templates that can be  }
                                { retrieved.                           }
     retcode2      : WORD;      { A secondary return code, or          }
                                { the number of templates that may     }
                                { be retrieved if a warning message    }
                                { occurred during the open of a        }
                                { REPLACE transfer request.            }
  END;
```

Parameters

buffaddr	Far address pointing to the transfer request buffer.
bufflen	The length of the transfer request buffer, which must be 4096.
reqlen	The length of data being sent.
sysaddr	Far address pointing to the name of the system on which your request will run.
conver	Type of translation.
stfapi	Returns 0 if request is successful.
retaddr	Returns the far address pointing to the address within the transfer request buffer.
retlen	Returns the length of data returned, or the number of templates that can be retrieved.
retcode2	A secondary return code.

Additional Information

The transfer function API passes input parameters to EHNTFSTF. For example, a call to the transfer function API of

```
stfapi((char)func,(stf far *)&params);
```

results in the following call to EHNTFSTF:

```
enhtfstf(func,
         params.buffaddr,
         params.bufflen,
         params.reqlen,
         params.sysaddr,
         params.conver,
         (char *)&params.retaddr,
         (unsigned *)&params.retlen,
         (unsigned *)&params.retcode2);
```

Stepwise, the transfer function API does the following:

1. Receives control.
2. Sets parameters to received parameter structure.
3. Calls EHNTFSTF.
4. Sets output parameters to output from EHNTFSTF.
5. Returns to caller.

Return Codes

Return Code	Description
X'0000'	STF_SUCCESS - Successful completion.
X'0103'	STF_REQUEST_ENDED - Transfer request is already ended.
X'0300'	STF_UNTRANS_DATA_FOUND - Untranslatable data found in transferred record.
X'0302'	STF_UNTRANS_DATA_SENT - Untranslatable data in record to be transferred to the AS/400 system.
X'0400'	STF_WARN_DETECTED_BY_AS400 - Warning detected by the AS/400 system.
X'1FFF'	STF_END_OF_FILE - End-of-file.
X'2000'	STF_MAX_TRANS_ACTIVE - Maximum number of transfer requests active.
X'2001'	STF_INVALID_REQUEST_LENGTH - Transfer request length not valid.
X'2002'	STF_CHAR_NOT_TRANSLATED - Transfer request character cannot be translated.
X'2003'	STF_REC_NOT_TRANSFERED - Record not transferred to AS/400 system due to untranslatable numeric data.
X'2004'	STF_NUMERIC_NOT_FIT - Numeric data does not fit into range specified by digits.
X'2005'	STF_INVALID_REC_LENGTH - Record length given by your application program is incorrect.
X'2006'	STF_STACK_TOO_SMALL - Program stack is not large enough.
X'2010'	STF_REQUEST_NOT_OPENED - Transfer request not opened.
X'2011'	STF_TEMPLATE_NOT_RETRIEVE - Template cannot be retrieved now.
X'2012'	STF_GET_REC_FAIL - Cannot get records on a Personal Computer-to-AS/400 request.
X'2013'	STF_SEND_REC_FAIL - Cannot send records on an AS/400-to-Personal Computer request.
X'2020'	STF_INVALID_REQUEST - Incorrect function requested.

Return Code	Description
X'2021'	STF_INVALID_BUF_LENGTH - Incorrect buffer length.
X'2022'	STF_INVALID_BUF_ADDR - Incorrect buffer address.
X'2023'	STF_REQUEST_BUF_OVERLAPPED - Transfer request buffer overlaps a previous transfer request buffer.
X'2024'	STF_INCORRECT_BUFADDR - Incorrect address specified for the buffer address parameter.
X'2025'	STF_INVALID_RETURN_ADDR - Incorrect address specified for the return address parameter.
X'2026'	STF_INVALID_RETURN_LENGTH_ADDR - Incorrect address specified for the return length parameter.
X'2027'	STF_INVALID_2ND_RETURN_ADDR - Incorrect address specified for the secondary return code parameter.
X'2028'	STF_INVALID_SYSTEM_ADDR - Incorrect address specified for the system address parameter.
X'3000'	STF_ERROR_W_STATE - Error detected by the AS/400 system; statement offset was provided.
X'3100'	STF_ERROR_WO_STATE - Error detected by the AS/400 system; no statement offset was provided.
X'5042'	STF_ROUTER_NOT_STARTED - AS/400 PC Support router has not been started.
X'5048'	STF_CONNECT_FAILED - Connection failed to system &&&&&&&&.
X'5050'	STF_INVALID_SYSTEM_NAME - System name is incorrect or inactive.
X'5052'	STF_PROGRAM_NOT_FOUND - System &&&&&&&& &&&&&&&&&&&&&&&&&&&&&&&&&&&& program not found.
X'5054'	STF_PROGRAM_ENDED_ABNORMALLY - System &&&&&&&& &&&&&&&&&&&&&&&&&&&&&&&&&&&& program ended unexpectedly.
X'5056'	STF_SECURITY_ERROR - Security error occurred for system &&&&&&&&.
X'5057'	STF_CM_NOT_AVAILABLE - Communications manager not available.
X'5058'	STF_SYSTEM_NOT_SUPPORTED - System &&&&&&&& is not a supported system.
X'5059'	STF_LOCAL_NOT_STARTED - Local name has not been started.
X'5060'	STF_INCORRECT_RELEASE - Cannot continue: &&&&&&&& must be Release &.& Level &&.
X'5062'	STF_CONTACT_ENDED - Contact with system &&&&&&&& ended.
X'5063'	STF_TOO_MANY_PROG - Too many communications manager programs active.
X'5064'	STF_CONTACT_INTERRUPTED - Contact with system &&&&&&&& temporarily interrupted.
X'5066'	STF_RESOURCE_FAILED - Resource failure on system &&&&&&&&.
X'5067'	STF_CONVERSATION_ENDED_ABNOR - Conversation with &&&&&&&& was ended unexpectedly.
X'5071'	STF_MEMORY_NOT_ENOUGH - Insufficient memory to perform function.

Return Code	Description
X'5080'	STF_PC_MEMORY_ERROR - PC memory cannot be used.
X'5400'	STF_OS_ERROR - Operating system returned error code &&&&.
X'9999'	STF_UNDEFINED_ERROR - Undefined error. Call your service representative.

Chapter 12. Work Station Function High-Level Application Program Interface

This chapter describes the high-level language API for the PC Support work station function. There is a set of library routines for the work station function (display), and a set for the work station function (printer). These functions may be used in a DOS or an OS/2 environment.

The API supports the following languages:

- IBM Pascal/2
- IBM C/2
- IBM Macro Assembler/2

The library routines described in this chapter are written to support large memory models.

Sample statements, procedure definitions, or data structure definitions for C and Pascal are provided in this chapter. An assembler language program could call the functions by sequence, so there are no such definitions given for assembler language.

Sample programs, include files, and other information for using APIs are provided in the PC Support tools folder (QIWSTOOL). See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

Work Station Function (Display)

The following sections describe the library routines for the work station function (display) API.

DOS Environment

This section defines the high-level language API for PC programs running in a DOS environment.

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOLS) for the work station function API:

WSFDSP.H	C language include file
WSFDSP_A.INC	Macro Assembler include file
WSFDSP_T.INC	Pascal language type section
WSFDSP_C.INC	Pascal language constant section
WSFDSP_E.INC	Pascal language external procedure declarations

Functions

The work station function API is based on the current functions of WSFAPI.EXE. The functions of the work station function API are grouped as follows:

- Supervisor services
  - Name resolution
- Session information services
  - Query session ID
  - Query session parameters
  - Query session cursor
  - Query system level
  - Query session status
  - Define hot-key characteristics
- Keyboard services
  - Connect to keyboard
  - Disconnect from keyboard
  - Read input
  - Write keystroke
  - Disable input
  - Enable input
  - Enable work station function DOS key processing
  - Disable work station function DOS key processing
- Copy services
  - Copy string
- Operator information area services
  - Read operator information area group

For all functions above, except name resolution, there is a corresponding entry point provided by the work station function API.

Name resolution is used to obtain the gate ID of a particular gate name. Valid gate names which correspond to a group are SESSMGR, KEYBOARD, COPY, or OIAM. Before requesting a certain service, application programs must first get the corresponding gate ID by requesting a name resolution service. To get all gate IDs, four name resolution services must be issued. When gate IDs are available, they are used as an input parameter to the work station function API. Service to the work station function API is identified by gate ID and the value in register AL.

Procedure Declaration

The declarations are the same for all functions except name resolution.

C

Statements to use in application programs:

```
#include <wsfdsp.h>

char far *params;    /* Parameters list */
```

Declarations provided in include files:

```
unsigned pascal entryname(params);
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION entryname(params : ADSMEM) : WORD;
```

Parameters

The **params** variable is a far address pointing to a function-specific structure, called a **parameter list**, which is defined in each section.

Additional Information

WSFAPI.EXE must be loaded before issuing a request. If WSFAPI.EXE is not loaded, or if it returns a value other than X'12' in register CH when called, a return code of X'00FF' is issued.

When any service is started before name resolution, a return code of X'0001' is issued.

The work station function API receives the parameters in the call and then passes the address of the parameter list to the work station function API. Before being passed, all reserved fields in the parameter lists are cleared. Afterwards, the work station function API returns its application return code in register CL. The API then returns to the caller with the application return code in register AX.

The API does not change the parameter list. The contents of all registers except AX are restored by the API.

Stepwise, the API does the following:

1. Receives control
2. Sets registers to received parameters
3. Calls WSFAPI by interrupt
4. Sets AX to the value in CL
5. Returns to the caller

There are two types of return codes returned by PCSAPI or WSFAPI:

Application return code	General to all services and returned in AX
Service return code	Specific to service and returned in the first byte of the parameter list

Generally, both return codes are set by the work station function API and are never changed by the PC Support API.

The section on each function describes the procedure and data structure declarations, supplied parameters and returned parameters, and return codes for each function.

Return Codes

Return Code	Description
X'0000'	WSF_SUCCESS - Request accepted.
X'0001'	WSF_NO_NAME_RESL - Name resolution not issued.
X'007F'	WSF_SESSION_NOT_AVAIL - Session not available.
X'00FF'	WSF_WSFAPI_NOT_LOADED - WSFAPI not loaded.

OS/2 Environment

The work station function API for OS/2 can be called through the emulator high-level language application program interface (EHLLAPI) functions.

IBM OS/2 EE communication manager (CM) provides EHLLAPI functions through dynamic link library (DLL) routines. This section describes the work station function DLL routines.

Include Files

The following include files are provided by OS/2 CM:

HAPI_C.H	C language include file
HAPI_M.INC	Macro Assembler include file
HAPI_P.INC	Pascal include file

These include files are in path \CMLIB after OS/2 CM is installed.

Related Functions

Several related functions are supported in the OS/2 communications manager through the EHLLAPI DLL (ACS3EHAP.DLL). The following list shows the related function codes and their DOS version API function:

Function=3	Write keystroke
Function=7	Query session cursor
Function=10	Query session ID
Function=13	Operator Information Area Services
Function=15	Copy string
Function=20	Query system level
Function=22	Query session parameters
Function=50	Connect to keyboard
Function=51	Read input
Function=53	Disconnect from keyboard

OS/2 EHLLAPI Format

MACRO ASSEMBLER

```
EXTRN HLLAPI: FAR

PUSH@ FUNCTION
PUSH@ DATA_STRING
PUSH@ LENGTH
PUSH@ RETCODE
CALL  HLLAPI
```

C

```
EXTERN void far pascal hllapi( int far *,
                               char far *,
                               int far *,
                               int far * );
```

PASCAL

```
TYPE STR_TYPE = ARRAY[1..1920] of CHAR;
PROCEDURE HLLAPI( VARS FUNC : INTEGER;
                  VARS DATA_STR:STR_TYPE;
                  VARS LEN,RETC : INTEGER ) : EXTERN;
```

Supervisory Service Requests

In the supervisory service requests group, the work station function API provides only one service, name resolution.

Name Resolution (NAMERESL)

Purpose

Issues SESSMGR, KEYBOARD, COPY, or OIAM according to the value of *name*:

Value	Gate Name
0	SESSMGR
1	KEYBOARD
2	COPY
3	OIAM

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>
int name;
```

Declarations provided in include files:

```
unsigned pascal nameresl(name);
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION NAMERESL(name : integer) : WORD;
```

Parameters

name	Integer 0 to 3 (represents SESSMGR, KEYBOARD, COPY, or OIAM).

Return Codes

Return Code	Description
X'0002'	WSF_INVALID_NAME_ID - Name resolution service number is not valid.

Session Information Service Requests

The session information services allow your application program to obtain the display session ID, session characteristics, cursor position, system level, session status, and hot-key characteristics of the work station function.

Query Session ID (QSESNID)

Purpose

Obtains the session ID of the display or printer specified. Users can specify a session by its short name, or ask for the IDs of all sessions.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct qsess_id_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal qsesnid(params);

struct qsess_id_lst {    /* parameter list */
            char ret_code;               /* Return code          */
            char func_id;                /* Must be zero         */
            char op_code;                /* Option code          */
            char data_code;              /* Data code            */
            struct name_array far *names; /* Address of name array */
            char reserved[8];            /* Reserved 8 bytes     */
};

struct name_block { /* declare name array element structure */
            char short_name;      /* Short name                   */
            char sess_type;       /* Session type                 */
            char session_id;      /* Session ID                   */
            char reserved[9];     /* Reserved 9 bytes             */
};

struct name_array {  /* name array header */
            char array_len;       /* Name array length  <= 62       */
            char matched_sess;    /* Number of match sessions <= 5  */
            struct name_block name[5]; /* Array of name block       */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION QSESNID(params : qsess_id_lst_ptr) : WORD;

TYPE
  qsess_id_lst = RECORD { Parameter list }
     ret_code  [00]: BYTE;        { Return code           }
     func_id   [01]: BYTE;        { Must be zero          }
     op_code   [02]: BYTE;        { Option code           }
     data_code [03]: BYTE;        { Data code             }
     names     [04]: ADSMEM;      { Address of name array }
     reserved  [08]: STRING(8);   { Reserved 8 bytes      }
  END;

  name_block = RECORD   { name array element structure}
     short_name [00] : BYTE;      { Short name            }
     sess_type  [01] : BYTE;      { Session type          }
     session_id [02] : BYTE;      { Session ID            }
     reserved1  [03] : BYTE;      { Reserved 9 bytes      }
     reserved2  [04] : BYTE;
     reserved3  [05] : BYTE;
     reserved4  [06] : BYTE;
     reserved5  [07] : BYTE;
     reserved6  [08] : BYTE;
     reserved7  [09] : BYTE;
     reserved8  [10] : BYTE;
     reserved9  [11] : BYTE;
  END;

  name_array = RECORD                  { Name array header             }
     array_len     [00]: BYTE;         { Name array length <= 62       }
     matched_sess  [01]: BYTE;         { Number of match sessions <= 5 }
     name : ARRAY[1..5] OF name_block; { Array of name block           }
  END;

  qsess_id_lst_ptr = ADS of qsess_id_lst;
```

Parameters

op_code	X'00' or X'01'.
data_code	X'02' or X'06', when op_code is X'00'; 1 character (A through E), when op_code is X'01'.
names	Address of name array.
array_len	Name array length.
func_id	Function ID, which must be X'6B'.
matched_sess	Returns the number of sessions that matched the request.
short_name	Returns the 1-character uppercase ASCII alphabetic name of the session (A through E).
sess_type	Returns X'02' for a host system display session; X'06' for a host system printer session.
session_id	Returns the ID that the work station function uses to identify the session. Use the session ID to specify the host session in any subsequent API service requests.

ret_code See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'06'	WSF_SESSION_ID_NOT_USED - Specified session ID not in use.
X'09'	WSF_INVALID_SESSION_TYPE - Session type is not valid.
X'0B'	WSF_INVALID_SHORT_NAME - Specified short name is not valid (not A-E).
X'0D'	WSF_INVALID_OPTION - Option code is not valid.
X'12'	WSF_INVALID_NAME_LENGTH - The name array length is not valid.
X'13'	WSF_LOWERCASE_NAME - Specified short name is not an uppercase ASCII alphabetic character.

Query Session Parameters (QSESNPM)

Purpose

Obtains the session characteristics of the display or printer specified.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct qsess_pm_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal qsesnpm(params);

struct qsess_pm_lst { /* Parameter list */
            char ret_code;        /* Return code                */
            char func_id;         /* Must be zero               */
            char session_id;      /* Session id                 */
            char reserved1;       /* Reserved                   */
            char session_type;    /* Session type               */
            char reserved2;       /* Reserved                   */
            char session_rows;    /* Screen rows  (24 or 27)    */
            char session_cols;    /* Screen columns  (80 or 132) */
            char reserve3[4];     /* Reserved 4 byte            */
            char model_id;        /* Device or model id         */
            char display_type;    /* Display adapter type       */
            char far *e_to_a_tbl; /* EBCDIC to ASCII translation */
            char far *a_to_e_tbl; /* ASCII to EBCDIC translation */
} ;
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION QSESNPM(params : qsess_pm_lst_ptr) : WORD;

TYPE
   qsess_pm_lst = RECORD            { Parameter list              }
     ret_code     [00] : BYTE;      { Return code                 }
     func_id      [01] : BYTE;      { Must be zero                }
     session_id   [02] : BYTE;      { Session ID                  }
     reserved1    [03] : BYTE;      { Reserved                    }
     session_type [04] : BYTE;      { Session type                }
     reserved2    [05] : BYTE;      { Reserved                    }
     session_rows [06] : BYTE;      { Screen rows  (24 or 27)     }
     session_cols [07] : BYTE;      { Screen columns  (80 or 132) }
     reserved3    [08] : STRING(4); { Reserved 4 byte             }
     model_id     [12] : BYTE;      { Device or model id          }
     display_type [13] : BYTE;      { Display adapter type        }
     e_to_a_tbl   [14] : ADSMEM;    { EBCDIC to ASCII translation }
     a_to_e_tbl   [18] : ADSMEM;    { ASCII to EBCDIC translation }
  END;

  qsess_pm_lst_ptr = ADS of qsess_pm_lst;
```

Parameters

session_id — ID of the display session whose characteristics were requested.

e_to_a_tbl — Address of a 256-byte area, which may be supplied to receive the contents of the EBCDIC-to-ASCII translation table. If the address is X'0000', the EBCDIC-to-ASCII table is not retrieved.

a_to_e_tbl — Address of a 256-byte area, which may be supplied to receive the contents of the ASCII-to-EBCDIC translation table. If the address is X'0000', the ASCII-to-EBCDIC table is not retrieved.

func_id — Function ID, which must be X'6B'.

session_type — Returns X'02' for a display session; X'06' for a printer session.

session_rows — Returns the number of rows in the session presentation space; X'18' (decimal 24) or X'1B' (decimal 27) for a display session and 0 for a printer session.

session_cols — Returns the number of columns in the session presentation space. X'50' (decimal 80) or X'84' (decimal 132) for a display session and 0 for a printer session.

model_id — Returns the model ID for a display session is:

- X'18' for 3196 Model A1 emulation.
- X'58' for 3197 Model C2 emulation.
- X'49' for 5292 Model 2 emulation.
- X'28' for 3180 Model 2 emulation.

The device ID for a printer session is:

- X'2D' for 5219 emulation.

- X'24' for 5224 emulation.
- X'60' for 5256 emulation.
- X'AD' for 3812 emulation.

display_type — Indicates whether a monochrome or color adapter is in use:

- X'00' for monochrome display adapter.
- X'01' for color display adapter.

ret_code — See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'06'	WSF_SESSION_ID_NOT_USED - Specified session ID not in use.

Query Session Cursor (QSESNCR)

Purpose

Obtains the cursor type and the row and column addresses of the specified session cursor.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct qsess_cr_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal qsesncr(params);

struct qsess_cr_lst {             /* Parameter list                */
            char ret_code;        /* Return code                   */
            char func_id;         /* Must be zero                  */
            char session_id;      /* Session ID                    */
            char cursor_type;     /* Cursor type                   */
            char row_addr;        /* Row address <= (24 or 27)     */
            char col_addr;        /* Column address <= (80 or 132) */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file              }
{$include: 'wsfdsp_t.inc'} { Type section include file                  }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION QSESNCR(params : qsess_cr_lst_ptr) : WORD;

TYPE
   qsess_cr_lst = RECORD          { Parameter list               }
      ret_code    [00] : BYTE;    { Return code                  }
      func_id     [01] : BYTE;    { Must be zero                 }
      session_id  [02] : BYTE;    { Session ID                   }
      cursor_type [03] : BYTE;    { Cursor type                  }
      row_addr    [04] : BYTE;    { Row address <= (24 or 27)    }
      col_addr    [05] : BYTE;    { Column address <= (80 or 132) }
   END;

qsess_cr_lst_ptr = ADS of qsess_cr_lst;
```

Parameters

session_id — ID of the display session whose cursor information is requested.

func_id — Function ID, which must be X'6B'.

row_addr — Returns the address in the session presentation space representing the cursor row position. The row address is a value from 1 to 27 for a display session, or X'00' for a printer session.

col_addr — Returns the address in the session presentation space representing the cursor column position. The column address is a value from 1 to 132 for a display session, or X'00' for a printer session.

ret_code — See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID is not valid.
X'06'	WSF_SESSION_ID_NOT_USED - Specified session ID not in use.

Query System Level (QSYSLVL)

Purpose

Obtains the system level information for the current release of the API for PC Support work station function.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct qsys_lvl_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal qsyslvl(params);

struct qsys_lvl_lst {             /* Parameter list               */
            char ret_code;        /* Return code                 */
            char func_id;         /* Must be zero on input       */
            char reserved1;       /* Reserved                    */
            char product_id[4];   /* Product ID                  */
            int  reserved2;       /* Unchanged                   */
            char api_version[2];  /* API version                 */
            char api_release[2];  /* API release                 */
            char ptf_level[2];    /* PTF level                   */
            char reserved4;       /* Reserved                    */
            char control_type;    /* Control program type        */
            char reserved5[3];    /* Reserved                    */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file              }
{$include: 'wsfdsp_t.inc'} { Type section include file                  }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION QSYSLVL(params : qsys_lvl_lst_ptr) : WORD;

TYPE
    qsys_lvl_lst = RECORD            { Parameter list       }
       ret_code     [00] : BYTE;     { Return code          }
       func_id      [01] : BYTE;     { Must be zero         }
       reserved1    [02] : BYTE;     { Reserved             }
       product_id   [03] : STRING(4);{ Product ID           }
       reserved2    [07] : WORD;     { Unchanged            }
       api_version  [11] : STRING(2);{ API version          }
       api_release  [11] : STRING(2);{ API release          }
       ptf_level    [13] : STRING(2);{ PTF level            }
       reserved4    [15] : BYTE;     { Reserved             }
       control_type [16] : BYTE;     { Control program type }
       reserved5    [17] : STRING(3);{ Reserved             }
    END;

qsys_lvl_lst_ptr = ADS of qsys_lvl_lst;
```

Parameters

func_id	Function ID; must be zero on input.
func_id	Function ID; will be X'6B' on return.
product_id	Returns the ID of the given product. It is a 4-byte ASCII string. The value is WSF for work station function.
api_version	Returns the 2-byte ASCII API version number (for example, 20 for API version 2.0, 00 for API version 1.0).

api_release	Returns the 2-byte ASCII API release number (for example, 10 for API release 1.0).
ptf_level	Returns the 2-byte ASCII API program temporary fix (PTF) level number (for example, 01 for API level 01).
control_type	Returns the control program types: • C - 3270 personal computer control program. • F - 5250 work station function. • W - 3270 work station program.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.

Query Session Status (QSESNST)

Purpose

Determines whether or not a display or printer session is active. Also determines whether or not the EBCDIC buffer's contents of the work station function display station have changed since the last update to the presentation space. This also indicates whether the Sys Req line or the message line is active. The message line is 25 on a 25x80 display session or line 28 on a 29x132 display session.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct qsess_st_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal qsesnst(params);

struct qsess_st_lst {            /* Parameter list                */
            char ret_code;       /* Return code                   */
            char func_id;        /* Must be zero on input         */
            char session_id;     /* Session ID                    */
            char reserved1;      /* Reserved                      */
            char status;         /* Session status                */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file              }
{$include: 'wsfdsp_t.inc'} { Type section include file                  }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION QSESNST(params : qsess_st_lst_ptr) : WORD;

TYPE
    qsess_st_lst = RECORD          { Parameter list        }
       ret_code   [00] : BYTE;     { Return code           }
       func_id    [01] : BYTE;     { Must be zero on input }
       session_id [02] : BYTE;     { Session ID            }
       reserved1  [03] : BYTE;     { Reserved              }
       status     [04] : BYTE;     { Session status        }
    END;

qsess_st_lst_ptr = ADS of qsess_st_lst;
```

Parameters

session_id — ID of the display session returned by the Query Session ID service.

func_id — Function ID; must be zero on input.

func_id — Function ID; will be X'6B' on return.

status — Returns the session status byte, defined as follows:

- Intel Bit 7 (IBM Bit 0)

  0 = Inactive session. An inactive session is one that is configured but is not communicating with the host.
  1= Active session. An active session is one that is configured and is communicating with the host.

- Intel Bit 6 (IBM Bit 1) - Buffer change status

  0=EBCDIC buffer has not been changed.
  1=EBCDIC buffer has been changed.

- Intel Bit 5 (IBM Bit 2) - Application program session use

  0=The session is not being used by an application program for keyboard services.
  1=The session is being used by an application program for keyboard services.

- Intel Bit 4 (IBM Bit 3) - Sys Req status:

  0=Sys Req is not active for this session.
  1=Sys Req is active for this session.

- Intel Bit 3 (IBM Bit 4) - Message line status:

  0=There is no message on the message line. The message line is line 25 on a 25x80 display session or line 28 on a 29x132 display session.
  1=There is a message on the message line. The message line is line 25 on a 25x80 display session or line 28 on a 29x132 display session.

- Intel Bit 0-2 (IBM Bit 5-7) - Reserved.

ret_code See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'06'	WSF_SESSION_ID_NOT_USED - Specified session ID not in use.

Define Hot-Key Characteristics (DEFHOTK)

Purpose

Defines the hot-key characteristics of a session.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct def_hk_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal defhotk(params);

struct def_hk_lst {
            char ret_code;        /* Return code                    */
            char func_id;         /* Must be zero on input          */
            char session_id;      /* Session ID                     */
            char reserved1;       /* Reserved                       */
            char hk_control;      /* Hot-key control                */
            char hk_char_row;     /* Hot-key character row          */
            char hk_char_col;     /* Hot-key character column       */
            char hk_char;         /* Hot-key character              */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION DEFHOTK(params : def_hk_lst_ptr) : WORD;

TYPE
  def_hk_lst = RECORD              { Parameter list          }
     ret_code    [00] : BYTE;      { Return code             }
     func_id     [01] : BYTE;      { Must be zero on input   }
     session_id  [02] : BYTE;      { Session ID              }
     reserved1   [03] : BYTE;      { Reserved                }
     hk_control  [04] : BYTE;      { Hot-key control         }
     hk_char_row [05] : BYTE;      { Hot-key character row   }
     hk_char_col [06] : BYTE;      { Hot-key character column}
     hk_char     [07] : BYTE;      { Hot-key character       }
  END;

def_hk_lst_ptr = ADS of def_hk_lst;
```

Parameters

func_id — Function ID; must be zero on input.

session_id — ID of the display session whose characteristics are being defined.

hk_control — Specifies whether you should make the operator's ability to hot-key into a session effective or ineffective.

- X'01' makes the hot-key effective.
- X'00' makes the hot-key ineffective.

hk_char_row — Row number (1-27) in which the specified character hot-key must be to have work station function automatically switch to the DOS session.

hk_char_col — Column number (1-132) in which the specified character hot-key must be to have work station function automatically switch to the DOS session. If you specify zero for the column value, automatic hot-key sequence to DOS on the specified character stops.

hk_char — ASCII character which, when found at the specified row and column on the display, automatically causes the work station function to do a hot-key sequence to DOS.

func_id — Function ID; will be X'6B' on return.

ret_code — See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'06'	WSF_SESSION_ID_NOT_USED - Specified session ID not in use.
X'14'	WSF_INVALID_HK_INPUT - Input parameter not valid.

Keyboard Service Request

The keyboard services allow the application program to write keystroke data to the work station function display session and to start and stop operator input from the keyboard.

Connect to Keyboard (CONTKBD)

Purpose

Connects to a display session for keyboard services.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct con_kbd_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal contkbd(params);

struct con_kbd_lst {              /* Parameter list                */
            char ret_code;        /* Return code                   */
            char func_id;         /* Must be zero on input         */
            char session_id;      /* Session ID                    */
            char reserved[7];     /* Reserved                      */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION CONTKBD(params : con_kbd_lst_ptr) : WORD;

TYPE
    con_kbd_lst = RECORD             { Parameter list        }
       ret_code    [00] : BYTE;      { Return code           }
       func_id     [01] : BYTE;      { Must be zero on input }
       session_id  [02] : BYTE;      { Session ID            }
       reserved    [03] : STRING(7); { Reserved              }
    END;

con_kbd_lst_ptr = ADS of con_kbd_lst;
```

Parameters

func_id	Function ID; must be zero on input.
session_id	ID of the display session to which you want to connect.
func_id	Function ID; will be X'62' on return.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'04'	WSF_ALREADY_CONNECTED - Session is already connected for keyboard services.
X'06'	WSF_SESSION_ID_NOT_USED - Specified session ID not in use.
X'15'	WSF_NOT_DISPLAY - Session must be host display session.

Disconnect from Keyboard (DISCNKBD)

Purpose

Disconnects from a display session when users finish using keyboard services.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct dis_kbd_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal discnkbd(params);

struct dis_kbd_lst {              /* Parameter list                */
            char ret_code;        /* Return code                   */
            char func_id;         /* Must be zero on input         */
            char session_id;      /* Session ID                    */
            char reserved1[3];    /* Reserved                      */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file              }
{$include: 'wsfdsp_t.inc'} { Type section include file                  }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION DISCNKBD(params : dis_kbd_lst_ptr) : WORD;

TYPE
  dis_kbd_lst = RECORD              { Parameter list        }
     ret_code    [00] : BYTE;       { Return code           }
     func_id     [01] : BYTE;       { Must be zero on input }
     session_id  [02] : BYTE;       { Session ID            }
     reserved    [03] : STRING(3); { Reserved              }
  END;

dis_kbd_lst_ptr = ADS of dis_kbd_lst;
```

Parameters

func_id	Function ID; must be zero on input.
session_id	ID of the display session from which you want to disconnect.
func_id	Function ID; will be X'62' on return.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'06'	WSF_SESSION_ID_NOT_USED - Specified session ID not in use.
X'04'	WSF_NOT_CONNECTED - Session not connected for keyboard services.
X'15'	WSF_NOT_DISPLAY - Session must be host display session.

Read Input (READIN)

Purpose

Reads intercepted keystroke data destined for the display session.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct read_in_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal readin(params);

struct read_in_lst {              /* Parameter list                */
            char ret_code;        /* Return code                   */
            char func_id;         /* Must be zero on input         */
            char session_id;      /* Session ID                    */
            char reserved1;       /* Reserved                      */
            int  reserved2;       /* Must be 0                     */
            char option;          /* Option byte, must be 0x80     */
                                  /* 0x80 = read, 0x90 = write     */
            char asc_length;      /* Length of ASCII/ASCII mnemonic */
                                  /* Returned in next field        */
            char mnemonic[6];     /* ASCII/ASCII mnemonic (6 bytes) */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION READIN(params : read_in_lst_ptr) : WORD;

TYPE
    read_in_lst = RECORD            { Parameter list               }
       ret_code   [00] : BYTE;      { Return code                  }
       func_id    [01] : BYTE;      { Must be zero on input        }
       session_id [02] : BYTE;      { Session ID                   }
       reserved1  [03] : BYTE;      { Reserved                     }
       reserved2  [04] : WORD;      { Must be 0                    }
       option     [06] : BYTE;      { Option byte, must be 0x80    }
                                    { 0x80 = read, 0x90 = write    }
       asc_length [07] : BYTE;      { Length of ASCII/ASCII mnemonic}
                                    { Returned in next field       }
       mnemonic   [08] : STRING(6); { ASCII/ASCII mnemonic:6 bytes }
    END;

read_in_lst_ptr = ADS of read_in_lst;
```

Parameters

func_id	Function ID; must be zero on input.
session_id	ID of the display whose keyboard you want to start.
option	Must have the byte value of X'80', indicating that an ASCII, or ASCII short form of the keystroke name is being received.
func_id	Function ID; will be X'62' on return.
mnemonic	Returns the ASCII, or ASCII short form of the operation names. This is from 1 to 6 bytes long. ASCII short form names start with @.
asc_length	Returns the length of the ASCII code or the ASCII short form name. Any bytes left unused remain unchanged.

ret_code See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'04'	WSF_NOT_CONNECTED - Session not connected for keyboard services.
X'06'	WSF_SESSION_ID_NOT_USED - Specified session ID not in use.
X'09'	WSF_NO_KEYSTROKE - There were no keystrokes to process.
X'15'	WSF_NOT_DISPLAY - Session must be host display session.

Write Keystroke (WRTKEY)

Purpose

Sends keystroke data to the display session just as an operator would enter keystrokes on a 5250 keyboard.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct write_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal wrtkey(params);

struct write_lst {              /* Parameter list                */
            char ret_code;      /* Return code                   */
            char func_id;       /* Must be zero on input         */
            char session_id;    /* Session ID                    */
            char reserved1;     /* Reserved                      */
            int  reserved2;     /* Must be 0                     */
            char option;        /* Option byte, must be 0x80     */
                                /* 0x80 = read, 0x90 = write     */
            char asc_length;    /* Length of ASCII/ASCII mnemonic */
                                /* Returned in next field        */
            char mnemonic[6];   /* ASCII/ASCII mnemonic:6 bytes  */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION WRTKEY(params : write_lst_ptr) : WORD;

TYPE
    write_lst = RECORD              { Parameter list                }
       ret_code   [00] : BYTE;      { Return code                   }
       func_id    [01] : BYTE;      { Must be zero on input         }
       session_id [02] : BYTE;      { Session ID                    }
       reserved1  [03] : BYTE;      { Reserved                      }
       reserved2  [04] : WORD;      { Must be 0                     }
       option     [06] : BYTE;      { Option byte, must be 0x80     }
                                    { 0x80 = read, 0x90 = write     }
       asc_length [07] : BYTE;      { Length of ASCII/ASCII mnemonic}
                                    { Returned in next field        }
       mnemonic   [08] : STRING(6); { ASCII/ASCII mnemonic:6 bytes  }
    END;

write_lst_ptr = ADS of write_lst;
```

Parameters

func_id	Function ID; must be zero on input.
session_id	ID of the display session to which keystrokes are written.
option	Must have the byte value of X'90', indicating that a single ASCII, or ASCII short form of operation name of the keystroke is being sent.
asc_length	Number of bytes of ASCII, or ASCII short form of operation names to send.
mnemonic	ASCII, or ASCII short form of operation name.
func_id	Function ID; will be X'62' on return.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'04'	WSF_NOT_CONNECTED - Session not connected for keyboard services.
X'10'	WSF_INVALID_KEYSTROKE - Processing stopped due to a keystroke value being sent that is not valid.
X'15'	WSF_NOT_DISPLAY Session must be host display session.

Disable Input (DISINPUT)

Purpose

Disables operator input to the display session.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct disable_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal disinput(params);

struct disable_lst {              /* Parameter list                */
            char ret_code;        /* Return code                   */
            char func_id;         /* Must be zero on input         */
            char session_id;      /* Session ID                    */
            char reserved[3];     /* Reserved                      */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION DISINPUT(params : disable_lst_ptr) : WORD;

TYPE
   disable_lst = RECORD           { Parameter list        }
      ret_code   [00] : BYTE;     { Return code           }
      func_id    [01] : BYTE;     { Must be zero on input }
      session_id [02] : BYTE;     { Session ID            }
      reserved   [03] : STRING(3);{ Reserved              }
   END;

disable_lst_ptr  = ADS of disable_lst;
```

Parameters

func_id	Function ID; must be zero on input.
session_id	ID of the display session whose keyboard you want to disable.
func_id	Function ID; will be X'62' on return.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'04'	WSF_NOT_CONNECTED - Session not connected for keyboard services.
X'15'	WSF_NOT_DISPLAY - Session must be host display session.

Enable Input (ENINPUT)

Purpose

Enables operator input to the display session again.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct enable_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal eninput(params);

struct enable_lst {              /* Parameter list                */
            char ret_code;       /* Return code                   */
            char func_id;        /* Must be zero on input         */
            char session_id;     /* Session ID                    */
            char reserved[3];    /* Reserved                      */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ENINPUT(params : enable_lst_ptr) : WORD;

TYPE
   enable_lst = RECORD            { Parameter list        }
      ret_code   [00] : BYTE;     { Return code           }
      func_id    [01] : BYTE;     { Must be zero on input }
      session_id [02] : BYTE;     { Session ID            }
      reserved   [03] : STRING(3);{ Reserved              }
   END;

enable_lst_ptr   = ADS of enable_lst;
```

Parameters

func_id	Function ID; must be zero on input.
session_id	ID of the display whose keyboard you want to enable.
func_id	Function ID; will be X'62' on return.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'04'	WSF_NOT_CONNECTED - Session not connected for keyboard services.
X'15'	WSF_NOT_DISPLAY - Session must be host display session.

Enable Work Station Function DOS Key Processing (ENDOSK)

Purpose

Enables work station function DOS key processing.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>
```

Declarations provided in include files:

```
unsigned pascal endosk(params);

struct enable_doskey_lst {        /* Parameter list                */
            char ret_code;        /* Return code                   */
            char func_id;         /* Must be zero on input         */
            char session_id;      /* Session ID                    */
            char reserved[3];     /* Reserved                      */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ENDOSK(params : enable_doskey_lst_ptr) : WORD;

TYPE
   enable_doskey_lst_ptr          { Parameter list        }
      ret_code   [00] : BYTE;     { Return code           }
      func_id    [01] : BYTE;     { Must be zero on input }
      session_id [02] : BYTE;     { Session ID            }
      reserved   [03] : STRING(3);{ Reserved              }
   END;

enable_doskey_lst_ptr   = ADS of enable_doskey_lst
```

Parameters

func_id	Function ID; must be zero on input.
session_id	ID of the display session to process the keys from the DOS session.
func_id	Function ID; will be X'62' on return.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'04'	WSF_NOT_CONNECTED - Session not connected for keyboard services.
X'15'	WSF_NOT_DISPLAY - Session must be host display session.
X'16'	WSF_DOS_KEY_ACTIVE_NOW - Another session is already processing the DOS keys.

Disable Work Station Function DOS Key Processing (DISDOSK)

Purpose

Disables work station function DOS key processing.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>
```

Declarations provided in include files:

```
unsigned pascal disdosk(params);

struct disable_doskey_lst {       /* Parameter list                */
            char ret_code;        /* Return code                   */
            char func_id;         /* Must be zero on input         */
            char session_id;      /* Session ID                    */
            char reserved[3];     /* Reserved                      */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION DISDOSK(params : disable_doskey_lst_ptr) : WORD;

TYPE
   disable_doskey_lst_ptr         { Parameter list        }
      ret_code   [00] : BYTE;     { Return code           }
      func_id    [01] : BYTE;     { Must be zero on input }
      session_id [02] : BYTE;     { Session ID            }
      reserved   [03] : STRING(3);{ Reserved              }
   END;

disable_doskey_lst_ptr   = ADS of disable_doskey_lst
```

Parameters

func_id	Function ID; must be zero on input.
session_id	ID of the display session to stop processing the keys from the DOS session.
func_id	Function ID; will be X'62' on return.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.

Return Code	Description
X'04'	WSF_NOT_CONNECTED - Session not connected for keyboard services.
X'15'	WSF_NOT_DISPLAY - Session must be host display session.

Copy Service Requests

In the copy service requests group, the work station function API provides only one service, copy string.

Copy String (COPYSTR)

Purpose

Copies a string either from an application buffer to the 5250 EBCDIC buffer, or from the 5250 EBCDIC buffer to an application buffer.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct copy_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal copystr(params) : WORD;

struct copy_lst {                 /* Parameter list                */
            char ret_code;        /* Return code                   */
            char func_id;         /* Must be zero on input         */
            char source_id;       /* Source session ID or zero     */
            char reserved1;       /* Reserved                      */
            char far *sourcebuff; /* Address of source buffer      */
            char reserved2;       /* Reserved                      */
            char source_stype;    /* Source session type           */
            int  source_start;    /* Offset of source start        */
            int  source_end;      /* Offset of source end          */
            char target_id;       /* Target session ID or zero     */
            char reserved3;       /* Reserved                      */
            char far *targetbuff; /* Address of target buffer      */
            char reserved4;       /* Reserved                      */
            char target_stype;    /* Target session type           */
            int  target_start;    /* Offset of target start        */
            int  reserved5;       /* Must be 0                     */
            char target_type;     /* Target type                   */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file                }
{$include: 'wsfdsp_t.inc'} { Type section include file                    }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION COPYSTR(params : copy_lst_ptr) : WORD;

TYPE
  copy_lst = RECORD                { Parameter list            }
     ret_code     [00] : BYTE;     { Return code               }
     func_id      [01] : BYTE;     { Must be zero on input     }
     source_id    [02] : BYTE;     { Source session ID or zero }
     reserved1    [03] : BYTE;     { Reserved                  }
     sourcebuff   [04] : ADSMEM;   { Address of source buffer  }
     reserved2    [08] : BYTE;     { Reserved                  }
     source_stype [09] : BYTE;     { Source session type       }
     source_start [10] : WORD;     { Offset of source starting }
     source_end   [12] : WORD;     { Offset of source ending   }
     target_id    [14] : BYTE;     { Target session id or zero }
     reserved3    [15] : BYTE;     { Reserved                  }
     targetbuff   [16] : ADSMEM;   { Address of target buffer  }
     reserved4    [20] : BYTE;     { Reserved                  }
     target_stype [21] : BYTE;     { Target session type       }
     target_start [22] : WORD;     { Offset of target starting }
     reserved5    [24] : WORD;     { Must be 0                 }
     target_type  [26] : BYTE;     { Target type               }
  END;

copy_lst_ptr = ADS of copy_lst;
```

Parameters

func_id	Function ID; must be zero on input.
source_id	Source session ID or zero.
sourcebuff	Address of source buffer.
source_stype	Source session type.
source_start	Offset of source start.
source_end	Offset of source end.
target_id	Target session ID or zero.
targetbuff	Address of target buffer.
target_stype	Target session type.
target_start	Offset of target start.
target_type	Target type.
func_id	Function ID; will be X'64' on return.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'03'	WSF_INPUT_INHIBITED - Target window is input inhibited.
X'05'	WSF_OVERLAYED_COPY - Source and target area overlap (copy performed).
X'06'	WSF_INVALID_SOURCE - Missing or invalid parameter on source definition.
X'07'	WSF_INVALID_TARGET - Missing or invalid parameter on target definition.
X'09'	WSF_TRUNCATED - Truncation occurred (copy performed).
X'0D'	WSF_TARGET_NOT_ALLOWED - Target not allowed.
X'15'	WSF_NOT_DISPLAY - Session must be host display session.

Operator Information Area Service Requests

In the operator information area service requests group, the work station function API provides only one service, read operator information area group.

Read Operator Information Area Group (READOIAM)

Purpose

Obtains a bit string that indicates the current settings of a group of indicators in the operator information area of the specified display session, and returns the indicator history data.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfdsp.h>

struct read_oia_lst far *params;
```

Declarations provided in include files:

```
unsigned pascal readoiam(params);

struct read_oia_lst {              /* Parameter list                 */
           char ret_code;          /* Return code - must be a 0 or 1 */
                                   /*   on call to WSFAPI            */
           char func_id;           /* Must be zero on input          */
           char source_id;         /* Source session ID or zero      */
           char reserved1;         /* Reserved                       */
           struct oia_mask far *oia_buff;
                                   /* Address of buffer              */
                                   /*   3 or 4 bytes                 */
};

struct oia_mask {                 /* Indicators:                    */
           unsigned mw:1;         /* Message waiting                */
           unsigned reserved1:1;  /* Reserved                       */
           unsigned ks:1;         /* Keyboard shift                 */
           unsigned reserved2:1;  /* Reserved                       */
           unsigned im:1;         /* Insert mode                    */
           unsigned dm:1;         /* Diacritic mode                 */
           unsigned ii:1;         /* Input inhibit                  */
           unsigned sa:1;         /* System available               */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfdsp_c.inc'} { Constant section include file               }
{$include: 'wsfdsp_t.inc'} { Type section include file                   }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION READOIAM(params : read_oia_lst_ptr) : WORD;

TYPE
  read_oia_lst = RECORD           { Parameter list              }
     ret_code  [00] : BYTE;       { Return code                 }
     func_id   [01] : BYTE;       { Must be zero on input       }
     source_id [02] : BYTE;       { Source session ID or zero   }
     reserved1 [03] : BYTE;       { Reserved                    }
     oia_buff  [04] : ADSMEM;     { Point to 3 or 4 oia_mask    }
                                  {   records                   }
  END;

oia_mask = ARRAY[1..4] of byte;   { Byte 1 : current status     }
                                  { Byte 2 : set history        }
                                  { Byte 3 : reset history      }
                                  { Byte 4 : special            }

read_oia_lst_ptr = ADS of read_oia_lst;
oia_mask_ptr   = ADS of oia_mask;
```

Parameters

func_id	Function ID; must be zero on input.
source_id	ID of the display session returned by the Query Session ID service from which operator information area (OIA) data is retrieved.
oia_buff	Buffer where the OIA data is returned. The buffer must be 3 bytes long.
oia_mask	The OIA data is returned in the OIA buffer, which is organized as follows: Byte 1 - current indicator status Byte 2 - indicator set history Byte 3 - indicator reset history
func_id	Function ID; will be X'6D' on return.
ret_code	See Return Codes.

Return Codes

Return Code	Description
X'00'	WSF_SUCCESS - Successful completion.
X'02'	WSF_INVALID_SESSION_ID - Session ID not valid.
X'15'	WSF_NOT_DISPLAY - Session must be host display session.

Work Station Function (Printer)

The following sections describe the library routines for the work station function (printer) API.

DOS Environment

This section defines the application program interface (API) for the PC Support work station function printer for PC programs running in a DOS environment.

Include Files

The following files are provided in the PC Support tools folder for the work station function printer API:

WSFPRT.H	C language include file
WSFPRT_A.INC	Macro Assembler include file
WSFPRT_T.INC	Pascal language type section
WSFPRT_C.INC	Pascal language constant section
WSFPRT_E.INC	Pascal language external procedure declarations

Functions

The functions provided by the work station function printer API are:

- Run printer panel option
- Get current printer status

Input and Output

WSF.EXE must be loaded before making a function call. If WSF.EXE is not loaded, a return code of X'00FF' is issued.

The work station function printer API is only valid for a printer session. When these functions are issued to a display session, a return code of X'0002' is issued.

The section on each function describes the procedure and data structure declarations, supplied parameters and returned parameters, and return codes of each function respectively.

Run Printer Panel Option (WFPXPNL)

Purpose

Processes an emulated printer option. The session_id specifies the target session. The valid values for session_id are 1 to 5.

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfprt.h>

struct wfpxopt far *params;
```

Declarations provided in include files:

```
unsigned pascal wfpxpnl(params);

struct wfpxopt {                /* Parameter list               */
            char ret_code;      /* Return code                 */
            char reserved;      /* Must be zero                */
            char session_id;    /* Session ID                  */
            char option;        /* Option                      */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfprt_c.inc'} { Constant section include file               }
{$include: 'wsfprt_t.inc'} { Type section include file                   }
{$include: 'wsfprt_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION wfpxpnl(params : wfpxopt_ptr);

TYPE
wfpxopt     = RECORD            { Parameter list   }
     ret_code    [00]: BYTE;    { Return code      }
     reserved    [01]: BYTE;    { Must be zero     }
     session_id  [02]: BYTE;    { Session ID       }
     option      [03]: BYTE;    { Option           }
END;

wfpxopt_ptr  = ADS of wfpxopt;
```

The following options can be selected:

Start	Readies the printer session. If the Ready indicator is on, you do not have to use this option.
Stop/reset	Stops printing until you make adjustments to the printer, such as changing the paper, and resets error conditions. To continue printing, you must use the Start option.
Suspend	Stops the AS/400 printing session and allows you to use the printer as a PC printer. When using the work station function, suspend your printer before turning off the power. When you turn on the power again, use the Start option. This prepares the printer for use with the work station function. Before starting the printer, align the paper.
Cancel	Notifies the AS/400 system that you want to cancel the print request.
Line feed	Advances the paper to the next print line. If you use the line feed switch on the printer during a printer session, the paper may no longer be aligned. Stop must be selected before using the line feed switch.
Form feed	Advances the paper to the first print line of the next form. Be sure to use this option from the panel. If you use the form feed switch on the printer, the paper may no longer be aligned. Stop must be selected before using the form feed switch.
Select printer values	Allows you to specify the number of lines per inch and the number of characters per inch to be printed. Stop must be selected before selecting printer values.

The input parameter option contains the option to be processed.

The valid options are:

WSFP_PRTSTART(X'0001')
: Start

WSFP_PRTSTOP (X'0002')
: Stop/reset

WSFP_PRTSUSP (X'0003')
: Suspend

WSFP_PRTCANCL(X'0004')
: Cancel

WSFP_PRTLINFD(X'0005')
: Line feed (for 5224/5256 emulation only)

WSFP_PRTFRMFD(X'0006')
Form feed (for 5224/5256 emulation only)
WSFP_PRT6LPI (X'0008') Select 6 LPI (for 5224/5256 emulation only)
WSFP_PRT8LPI (X'0009') Select 8 LPI (for 5224/5256 emulation only)
WSFP_PRT10CPI(X'000A')
Select 10 CPI (for 5224/5256 emulation only)
WSFP_PRT15CPI(X'000B')
Select 15 CPI (for 5224/5256 emulation only)

If the option contains a number that is not valid, a return code of X'0001' is issued.

If the function is issued to a 5219 or 3812 session with option containing number 5, 6, 8, 9, X'A',or X'B', a return code of X'0003' is issued.

Parameters

session_id	Specifies the target session.
option	The valid options are shown above.
ret_code	Dummy field, which is for structure consistency.

Return Codes

Return Code	Description
X'0000'	WSFP_PRT_SUCCESS - Request accepted.
X'0001'	WSFP_INVALID_OPTION - Option not valid.
X'0002'	WSFP_NOT_PRT_SESSION - Not a printer session.
X'0003'	WSFP_NOT_5224_5256 - Not a 5224 or 5256 printer session.
X'0004'	WSFP_INVALID_SESSION - Session number not valid.
X'007F'	WSFP_SESSION_NOT_AVAIL - Session not available.
X'00FF'	WSFP_NOT_LOADED - WSF not loaded.

Get Current Printer Status (WFPGETST)

Purpose

Gets the current printer status. The session_id specifies the target session. The valid values for session_id are 1 to 5.

The status is returned in the parameter passed to the API. The bits within pnlstate represent the state of each of the indicators on the emulated operator panel. Indicators are on if the corresponding bits are on.

The bits are defined as follows for 5256 printer emulation (all other bits are reserved):

- Bit2 = (X'04') system available
- Bit3 = (X'08') data cleared
- Bit4 = (X'10') graphics check
- Bit5 = (X'20') forms

- Bit6 = (X'40') attention
- Bit7 = (X'80') ready

The bits are defined as follows for 5224 printer emulation (all other bits are reserved):

- Bit1 = (X'02') change font
- Bit2 = (X'04') system available
- Bit4 = (X'10') graphics check
- Bit5 = (X'20') forms
- Bit6 = (X'40') attention
- Bit7 = (X'80') ready

The bits are defined as follows for 5219 and 3812 SCS printer emulation (all other bits are reserved):

- Bit2 = (X'04') system available
- Bit3 = (X'08') change font
- Bit4 = (X'10') change setup
- Bit5 = (X'20') forms
- Bit6 = (X'40') printer exception
- Bit7 = (X'80') ready

The bits within msgstate represent the state of each of the user messages on the emulated operator panel. Messages are displayed if the corresponding bits are on.

- Bit0 = (X'01') invalid option
- Bit1 = (X'02') attached printer off-line
- Bit2 = (X'04') suspended
- Bit3 = (X'08') data loss may have occurred

Procedure Declaration

C

Statements to use in application programs:

```
#include <wsfprt.h>

struct wfpstate far *status;      /* Printer status                */
```

Declarations provided in include files:

```
unsigned pascal wfpgetst(status);

struct wfpstate {                 /* Parameter list                */
            char ret_code;        /* Return code                   */
            char reserved;        /* Must be zero                  */
            char session_id;      /* Session ID                    */
            char pnlstate;        /* Panel status                  */
            char msgstate;        /* Message status                */
            char emul_type;       /* Emulation type                */
            char drawer;          /* Drawer                        */
            char prt_font;        /* Printer font                  */
            char out_data;        /* Output data                   */
            char prt_id[10];      /* Printer ID,in EBCDIC          */
            char sys_name[8];     /* System name                   */
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'wsfprt_c.inc'} { Constant section include file                }
{$include: 'wsfprt_t.inc'} { Type section include file                    }
{$include: 'wsfdsp_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION wfpgetst(status : wfpstate_ptr);

TYPE
wfpstate    = RECORD              { Parameter list         }
     ret_code    [00]: BYTE;      { Return code            }
     reserved    [01]: BYTE;      { Must be zero           }
     session_id  [02]: BYTE;      { Session ID             }
     pnlstate    [03]: BYTE;      { Panel status           }
     msgstate    [04]: BYTE;      { Message status         }
     emul_type   [05]: BYTE;      { Emulation type         }
     drawer      [06]: BYTE;      { Drawer                 }
     prt_font    [07]: BYTE;      { Printer font           }
     out_data    [08]: BYTE;      { Output data            }
     prt_id      [09]: STRING(10);{ Printer ID,in EBCDIC   }
     sys_name    [19]: STRING(8); { System name            }
END;

wfpstate_ptr = ADS of wfpstate;
```

Parameters

session_id	Specifies the target session.
pnlstate	Returns the panel status.
msgstate	Returns the message status.
emul_type	Returns the emulation type.
drawer	Returns the drawer.
prt_font	Returns the printer font.
out_data	Returns the output data.
prt_id[10]	Returns the printer ID, in EBCDIC.
sys_name[8]	Returns the system name.
ret_code	Dummy field, which is for structure consistency.

Return Codes

Return Code	Description
X'0000'	WSFP_PRT_SUCCESS - Request accepted.
X'0002'	WSFP_NOT_PRT_SESSION - Not a printer session.
X'0004'	WSFP_INVALID_SESSION - Session number is not valid.
X'007F'	WSFP_SESSION_NOT_AVAIL - Session not available.
X'00FF'	WSFP_NOT_LOADED - WSF not loaded.

Chapter 13. Submit Remote Command High-Level Application Program Interface

This chapter discusses the Submit Remote Command application program interface (API) and its reply messages and return codes.

The API consists of the EHNSRSBM and EHNSRSTC routines. The EHNSRSBM routine sends a command to a remote system after starting a conversation, if necessary, with a remote system. The EHNSRSTC routine stops a conversation with a remote system.

The DOS API (EHNSRAPI.LIB) is a library of object modules. The API has two entry points, EHNSRSBM and EHNSRSTC. Each returns a return code in the AX register. Only one conversation is supported for each application.

The OS/2 API (EHNSRAPI.DLL) is a dynamic link library (DLL) that must be included in your library path. The API has two entry points, EHNSRSBM and EHNSRSTC. Each returns a return code in the AX register. Only one conversation is supported for each application.

For DOS API only:

1. All public labels except EHNSRSBM and EHNSRSTC are prefixed with _EHNSR_.
2. Applications are linked again so the API program temporary fixes (PTFs) can be used.
3. Stack size must be at least 350 bytes.
4. Linker/2 is required to link this API into your application.

For OS/2 API only:

1. The local area network uses your application program's stack when your application program performs requests. The local area network requires approximately 2560 bytes. You need additional stack space for your application program's own variables and the variables used by the Submit Remote Command API. Therefore, the recommended stack size is at least 3000 bytes. Refer to the *OS/2 Extended Edition Version APPC Programmer's Reference* for additional information about stack size requirements.
2. Linker/2 is required to link this API into your application.

Submit Remote Command Function Library Routines

The API supports the following languages:

- IBM Pascal/2
- IBM C/2
- IBM Macro Assembler/2

The library routines described in this chapter are written to support large memory models.

Sample statements, procedure definitions, or data structure definitions for C and Pascal are provided in this chapter. An assembler language program could call the functions by sequence, so there are no such definitions given for assembler language.

Sample programs, include files, and other information for using APIs are provided in the PC Support tools folder (QIWSTOOL). See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

DOS Environment

This section defines the high-level language API for the submit remote command function for PC programs running in a DOS environment.

Note: For information describing the high-level API for the submit remote command function for programs running in the Windows environment, see Part 3, "PC Support Windows Application Program Interfaces" on page 18-23.

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the submit remote command function API:

SBMCMD.H	C language include file.
SBMCMD_A.INC	Macro Assembler include file.
SBMCMD_T.INC	Pascal language type section.
SBMCMD_C.INC	Pascal language constant section.
SBMCMD_E.INC	Pascal language external procedure declarations.

Functions

The submit remote command function APIs for the DOS environment are:

- Remote commands
- Parse message

OS/2 Environment

IBM OS/2 EE Communication Manager (CM) provides an APPC interface for PC programs running in an OS/2 environment through dynamic link library (DLL) routines. This section describes the submit remote command DLL routines.

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the submit remote command function API:

SBMCMD.H	C language include file.
SBMCMD_A.INC	Macro Assembler include file.
SBMCMD_T.INC	Pascal language type section.
SBMCMD_C.INC	Pascal language constant section.
SBMCMD_E.INC	Pascal language external procedure declarations.

Functions

The submit remote command function APIs for the OS/2 environment are:

- Remote commands
- Parse message

EHNSRSBM

This entry point submits a command to the remote system. The process includes the following:

1. A conversation is allocated when the API is first called by your application.
2. The conversation continues until EHNSRSTC is called by your application, or until a communication problem occurs. If a communication problem occurs, the next call to EHNSRSBM causes another conversation allocation attempt.

EHNSRSBM uses the PASCAL calling convention and requires the following far pointers and word value to be pushed onto a stack in the order listed:

1. A far pointer to an ASCIIZ string containing the command to be submitted. The maximum length of a command string is 2000 bytes (not including the ending null).
2. A far pointer to an ASCIIZ string containing the name of the remote system. The maximum length of the remote system name is 8 bytes (not including the ending null). The current default system is used when you do not provide a name.
3. A far pointer to a buffer where reply messages will be placed. The maximum length of the reply message buffer is 64K minus 1.

   Note: The API binary zeros out the reply message buffer before placing reply messages in the buffer.
4. A 2-byte word value that shows the unsigned length in bytes of the reply message buffer.

For OS/2 applications only:

If a multi-threaded application is calling the RMTCMD API, the API call should not be made by thread one. Only threads created by the DosCreateThread command should call the RMTCMD API.

Reply Message Buffer

After a call to EHNSRSBM, the reply message buffer contains one to 10 reply messages. If there is more than one message, the highest severity return code of all the reply messages is returned.

The `reply message truncated` return code is returned if:

- The reply message buffer is too small for all of the messages.
- Only one reply message is being returned, but the reply message buffer is too small for it.

The reply message buffer sequentially stores the reply messages. Each reply message is a variable length string. The layout of each reply message is as follows:

```
LL1  CP1  LL  CP  Data  LL  CP  Data ...  LL  CP  Data
```

LL1 Two bytes long. Provides the length of the entire reply message.

CP1 Two byte code point. A unique number that indicates that this message is a reply message.

The data is divided into scalar objects. Each scalar object has a length (LL, two bytes long), a code point (CP, two bytes long) and data (for example, the message text returned by the DDM on the remote system).

Refer to the following list for the reply messages and scalar objects that can be returned. The code points are hexadecimal values.

X'D201' AS/400 system specific message. An AS/400 system specific request completed normally or abnormally. Possible scalar objects:

- severity code (0 to 128; required)
- AS/400 message identifier (optional)
- AS/400 message text (optional)
- AS/400 message type (optional)
- AS/400 message replacement data (optional)
- AS/400 message file name (optional)
- server diagnostic information (optional)

This reply message is returned if the DDM running on the remote system does not support the Submit Remote Command:

X'1250' Command not supported. The command specified is not recognized or not supported for the specified target object. Possible scalar objects:

- severity code (8; required)
- code point attribute (required)
- server diagnostic information (optional)

The following reply messages should rarely, if ever, be returned:

X'1232' Permanent system error. The command requested could not be completed because of a permanent error condition detected at the remote system. Possible scalar objects:

- severity code (16 to 64; required)
- record count (minimum value = 0; optional)
- server diagnostic information (optional)

X'1254' Command check. The requested command encountered a condition which is not architected and is implementation specific, for which there is no architected message. Possible scalar objects:

- severity code (0 to 128; required)
- file name (optional)
- declared name (optional)
- cursor position status (optional)
- record lock status (required)

record count (optional)

server diagnostic information (optional)

X'1233' Resource limits reached. The requested command could not be completed because of insufficient target server resources. Possible scalar objects:

severity code (8 to 128; required)

cursor position status (optional)

record lock status (optional)

declared name (optional)

file name (optional)

record count (minimum value = 0; optional)

server diagnostic information (optional)

These reply messages are unexpected. If one of them is returned, it indicates an internal error, either in the DLL or in the target DDM.

X'1245' Conversational protocol error. A conversational protocol error occurred. Possible scalar objects:

severity code (8, 16, 128; required)

conversational protocol error code (required)

record count (minimum value = 0; optional)

server diagnostic information (optional)

X'1251' Parameter not supported. The parameter specified is not recognized or not supported for the specified command. Possible scalar objects:

severity code (8; required)

code point attribute (required)

server diagnostic information (optional)

X'124C' Data stream syntax error. The data sent to the remote system does not structurally conform to the requirements of DDM architecture. Possible scalar objects:

severity code (8; required)

syntax error code (required)

record count (minimum value = 0; optional)

server diagnostic information (optional)

X'1252' Parameter value not supported. The parameter value specified is not recognized or not supported for the specified parameter. Possible scalar objects:

severity code (8; required)

code point attribute (required)

record count (minimum value = 0; optional)

server diagnostic information (optional)

The various scalar objects are listed below. The code points are hexadecimal values.

X'1149' Severity code. The severity code is an indicator of the severity of a condition detected when a command runs. It is a 16-bit, binary number. The possible range of values is:

Value	Description
0	Informational
4	Warning
8	Error
16	Severe
32	Access damage
64	Permanent damage
128	Session damage

X'1153' Server diagnostic information. Specifies diagnostic information on reply messages that is defined by the responding server. It is a byte string with a maximum length of 255.

X'000C' Code point attribute. Specifies a hexadecimal alias of a class in a dictionary. Code points reduce the number of bytes required to identify the class of an object in data streams. It is a 2-byte word value.

X'D112' AS/400 system message identifier. It is a character string with a length of 7.

X'D116' AS/400 system message text. It is a character string with a minimum length of 1 and a maximum length of 512.

X'D117' AS/400 system message type. It is a 2-byte, character string. The possible range of values is:

Value	Description
'01'	Completion message
'02'	Diagnostic message
'04'	Informational message
'15'	Escape message

X'D10E' AS/400 system message replacement data. It is a byte string.

X'D111' AS/400 system message file name. It is a 10-character string.

These scalar objects appear only in the unexpected reply messages:

X'111A' Record count. Specifies the number of records to be retrieved or inserted. It is a 32-bit, binary number.

X'115B' Cursor position status. Specifies in a reply message, the status of the cursor. It is a 1-byte, Boolean value of X'F1' or X'F0'.

X'115C' Record lock status. Specifies in a reply message, the status of the record lock. It is a 1-byte, Boolean value of X'F1' or X'F0'.

X'110E' File name. A DDM file name is a character string that is not architected. The minimum length is 1 byte and the maximum length is 255 bytes.

X'1136' Declared name. A declared name for a file is a short alias defined by the source system to the target system through the DCLFIL command. It is a byte string with a maximum length of 7.

X'113F' Conversational protocol error code. It is a 1-byte, hexadecimal digit.

X'114A' Syntax error code. It is a 1-byte, hexadecimal digit.

EHNSRSTC

This entry point stops your conversation with the remote system. If no conversation exists when EHNSRSTC is called, no action is taken and a return code is returned indicating that the call was successful.

This entry point does not require any input.

For OS/2 applications only: If the application ends without calling EHNSRSTC, the local area network ends any conversation that is active.

Return Codes

The return codes returned by the API are shown in Table 13-1.

Table 13-1. Return Codes Returned by the RMTCMD API

Code Number	Description
0	No errors or warnings.
10	Warning(s) returned from the remote system. Refer to the reply messages to determine the problem.
20	Error(s) returned from the remote system. Refer to the reply messages to determine the problem.
21	Severe error(s) returned from the remote system. Refer to the reply messages for more information.
22	DDM on the remote system has had its ability to access a file damaged. Refer to the reply messages for more information.
23	Permanent damage has occurred to permanent objects on the remote system. Refer to the reply messages for more information.
24	Target DDM has been damaged, no further communication will be possible on this session. Refer to the reply messages for more information.
31	Command string syntax problem. This error occurs if the command string is longer than 2000 bytes.
41	Remote system name syntax problem. This error occurs if the system name is longer than 8 bytes.
43	The reply message(s) did not fit in the supplied buffer. This error occurs when the length of the reply message buffer is smaller than the combined length of the reply messages that were returned. This error also occurs when the far pointer to the reply message buffer is equal to zero.
48	A different remote system name than the current remote system name was used. Stop the current conversation before changing the remote systems.
49	The remote system does not support the DDM command to submit remote commands.
50	Contact with remote system ended.
51	Conversation with remote system unexpectedly ended.
52	Target DDM unexpectedly ended.

Table 13-1. Return Codes Returned by the RMTCMD API

Code Number	Description
53	Conversation allocation unsuccessful because a session was not available.
54	Remote system named not found.
55	Insufficient remote system resource to start DDM on the remote system.
56	Contact with remote system temporarily interrupted. Try the operation again.
57	DDM not on the remote system.
58	User ID or password not correct.
68	Too many programs using the OS/2 Communications Manager.
69	OS/2 Communications Manager is not available.
70	The router is not loaded.
71	The LCLN entry in the configuration file does not match names with the LU configured using the local area network.
72	The API could not dynamically allocate sufficient memory (OS/2 operating system only).
73	Reply message buffer not valid. For example, the buffer pointer or the buffer length was incorrect.
74	Command string buffer not valid. For example, the buffer pointer was incorrect or the buffer string was not ended with null bytes.
75	Remote system buffer not valid. For example, the buffer pointer was incorrect or the buffer string was not ended with null bytes.
80	Insufficient stack was allocated.
81	A reserved MODE name was used (OS/2 operating system only).
82	The remote system unexpectedly interrupted the request. This is a RMTCMD API internal error that should not normally occur.
83	The conversation is not in a valid state. This is a RMTCMD API internal error that should not normally occur.
84	An unexpected return code was received. This is a RMTCMD API internal error that should not normally occur.
85	An unexpected return code was received. This is a RMTCMD API internal error that should not normally occur.
86	An unexpected return code was received. This is a RMTCMD API internal error that should not normally occur.
87	An unexpected return code was received. This is a RMTCMD API internal error that should not normally occur.
88	An unexpected reply message was received. This is a RMTCMD API internal error that should not normally occur.
89	An unexpected OS/2 return code was received. This is a RMTCMD API internal error that should not normally occur.
90	An unexpected OS/2 router return code was received. This is a RMTCMD API internal error that should not normally occur.
91	The wrong amount of data was received. This is a RMTCMD API internal error that should not normally occur.
99	This is a RMTCMD API internal error that should not normally occur.

Chapter 14. Shared Folders Function High-Level Application Program Interface

This chapter describes the high-level language API for the PC Support shared folders function. These functions may be used in a DOS or an OS/2 environment.

The API supports the following languages:

- IBM Pascal/2
- IBM C/2
- IBM Macro Assembler/2

The library routines described in this chapter are written to support large memory models.

Sample statements, procedure definitions, or data structure definitions for C and Pascal are provided in this chapter. An assembler language program could call the functions by sequence, so there are no such definitions given for assembler language.

Sample programs, include files, and other information for using APIs are provided in the PC Support tools folder (QIWSTOOL). See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

DOS Environment

This section defines the high-level language API for the shared folders function for PC programs running in a DOS environment.

Note: For information describing the high-level API for the shared folders function for programs running in the Windows environment, see Chapter 23, "Shared Folders Windows Application Program Interface."

Include Files

The following files are provided in the PC Support tools folder for the shared folders function API:

SFLR.H	C language include file.
SFLR_A.INC	Macro Assembler include file.
SFLR_T.INC	Pascal language type section.
SFLR_C.INC	Pascal language constant section.
SFLR_E.INC	Pascal language external procedure declarations.

Functions

The shared folders function APIs for the DOS environment are:

- Assign folder drive
- Check directory of folder
- Check in file in folder
- Check out file in folder
- Check user of file in folder
- Get drive status
- Query assigned folder

- Query folder names
- Release folder drive

OS/2 Environment

This section defines the high-level language API for the shared folders function for PC programs running with the OS/2 program.

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the shared folders function API:

SFLR.H	C language include file.
SFLR_A.INC	Macro Assembler include file.
SFLR_T.INC	Pascal language type section.
SFLR_C.INC	Pascal language constant section.
SFLR_E.INC	Pascal language external procedure declarations.

Functions

The shared folders function APIs for the OS/2 environment are:

- Assign folder drive
- Check directory of folder
- Check in file in folder
- Check out file in folder
- Check user of file in folder
- Query assigned folder
- Query folder names
- Release folder drive

Shared Folders Function

The following sections for the shared folders function describe in detail:

- Purpose
- Procedure and data structure declarations
- User-supplied values
- Returned values
- Return codes

Assign Folder Drive (SFASGN)

This procedure can be used with the shared folders function for extended DOS support or for the OS/2 operating system.

Purpose

Assigns the drive identified by the variable drv to a folder name pointed to by the pointer flrnaptr on the system name pointed to by the pointer sysnaptr.

Procedure Declaration

C

Statements to use in application programs:

```
#include <SFLR.H>

char   drv,                /* drive letter -- A (a) to Z (z)       */
  far  *sysnaptr,          /* pointer to the AS/400 system name    */
  far  *flrnaptr;          /* pointer to the folder name           */
```

Declarations provided in include files:

```
unsigned pascal sfasgn( drv, sysnaptr, flrnaptr );
```

PASCAL

Statements to use in application programs:

```
{$include: 'SFLR_C.INC'} { Constant section include file               }
{$include: 'SFLR_T.INC'} { Type section include file                   }
{$include: 'SFLR_E.INC'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION  SFASGN(drv : CHAR; sysnaptr : sysnameptr;
                 flrnaptr :  folderptr) : WORD;
                           {drv     --drive letter                  }
                           {sysnaptr--pointer to the AS/400 system  }
                           {          name                          }
                           {flrnaptr -- pointer to the folder name  }

CONST
  SYSLEN = 9;
  FLRLEN = 64;

TYPE
  sysname = STRING(SYSLEN);       {AS/400 system name                }
  folder  = STRING(FLRLEN);       {AS/400 folder name                }
  sysnameptr = ADS of sysname;    {Pointer to AS/400 system name     }
  folderptr  = ADS of folder;     {Pointer to AS/400 folder name     }
```

Parameters

drv — Drive letter. Valid drive letters are A (a) to Z (z).

sysnaptr — A pointer to the AS/400 system name, an ASCIIZ string. If this is a null pointer, the default system is used.

flrnaptr — A pointer to a folder name, an ASCIIZ string. If this is a null pointer, the drive is assigned to all folders on the system.

Return Codes

Return Code	Description
X'0000'	SF_SUCCESS - Successful.
X'0001'	SF_DRV_NOT_DEF - Drive is not a shared folders drive.

Return Code	Description
X'0003'	SF_FLR_NAME_NOT_FOUND - Folder name not found on host system.
X'0004'	SF_FLR_NAME_NOT_VALID - Folder name not valid.
X'0005'	SF_NOT_AUTHORIZED_TO_FOLDER - Not authorized to the specified folder name.
X'0007'	SF_USER_NOT_FOUND_OR_VALID - The user ID or password used to sign on to the router was not valid.
X'0008'	SF_NOT_ENOUGH_PC_RESOURCE - Not enough PC resources to assign a drive.
X'000F'	SF_DRV_NOT_VALID - The drive specified is not valid. If using DOS, the shared folders function may not have been started.
X'0015'	SF_DRV_NOT_READY - OS/2 shared folders function has not been started.
X'001F'	SF_COMMUNICATION_ERR - Communications error occurred.
X'003B'	SF_NETWORK_ERROR - An error occurred in the communications layer.
X'0054'	SF_NO_DRV_AVAILABLE - No drives are available.
X'0055'	SF_DRV_ALREADY_ASSIGNED - Drive is already assigned.
X'005B'	SF_VERSION_NOT_MATCH - Version mismatch between host and PC program levels.
X'005C'	SF_SYSTEM_NOT_VALID_OR_ACTIVE - System name is not valid or system is inactive.
X'005D'	SF_CM_NOT_ACTIVE - Communications manager is not active.
X'005E'	SF_ROUTER_NOT_LOADED - PC Support router is not loaded.
X'005F'	SF_LOCAL_LU_NOT_ACTIVE -Local LU specified in CONFIG.PCS is not correct.
X'0060'	SF_FUNCTION_NOT_SUPPORTED - The shared folders program running does not support this function. The shared folders function for extended DOS must be used.
X'0061'	SF_SYS_NAME_NOT_VALID - System name too long.
X'0062'	SF_SYS_CONTACT_ENDED - The data link was lost or an error occurred on the host system.
X'0063'	SF_SYS_RESOURCE_FAILURE - A resource error occurred on the host system.
X'00FC'	SF_NOT_LOADED - EHNSFL0.DLL was not loaded.

Release Folder Drive (SFRELS)

This procedure can be used with the shared folders function for extended DOS support or for the OS/2 operating system.

Purpose

Releases the drive or drives identified by the variable drv assigned to an AS/400 system. When releasing all drives, if some drives are not released successfully, the function returns the return code of the last drive that is not released.

Procedure Declaration

C

Statements to use in application programs:

```
#include <SFLR.H>

char   drv;                /* drive letter-- A (a) to Z (z)    */
                           /* or *                             */
```

Declarations provided in include files:

```
unsigned pascal sfrels( drv );
```

PASCAL

Statements to use in application programs:

```
{$include: 'SFLR_C.INC'} { Constant section include file                }
{$include: 'SFLR_T.INC'} { Type section include file                     }
{$include: 'SFLR_E.INC'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION  SFRELS(drv : CHAR) : WORD;
                           {drv  --  drive letter                  }
```

Parameters

drv Drive letter. Valid drive letters are A (a) to Z (z). Use * or the constant ALL_DRV to release all assigned drives. The include files define ALL_DRV = '*'.

Return Codes

Return Code	Description
X'0000'	SF_SUCCESS - Successful.
X'0001'	SF_DRV_NOT_DEF - Drive is not a shared folders drive.
X'0003'	SF_FILE_OPEN - Drive has open files.
X'000F'	SF_DRV_NOT_VALID - The drive specified is not valid. If using DOS, the shared folders function may not have been started.
X'0015'	SF_DRV_NOT_READY - OS/2 shared folders function has not been started.
X'001F'	SF_COMMUNICATION_ERR - Communications error occurred.
X'003B'	SF_NETWORK_ERROR - An error occurred in the communications layer.
X'0055'	SF_DRV_NOT_ASSIGNED - Drive not assigned.

Return Code	Description
X'005B'	SF_VERSION_NOT_MATCH - Version mismatch between host and PC program levels.
X'005C'	SF_SYSTEM_NOT_VALID_OR_ACTIVE - System name is not valid or system is inactive.
X'005E'	SF_ROUTER_NOT_LOADED - PC Support router is not loaded.
X'0060'	SF_FUNCTION_NOT_SUPPORTED - The shared folders program running does not support this function. The shared folders function for extended DOS must be used.
X'0062'	SF_SYS_CONTACT_ENDED - The data link was lost or an error occurred on the host system.
X'008E'	SF_DRV_BUSY - Drive is being used by another program.
X'00FC'	SF_NOT_LOADED - EHNSFL0.DLL was not loaded.

Get Drive Status (SFGDS)

This procedure can only be used with the shared folders function for extended DOS support.

Purpose

Determines if the shared folders function is loaded and if each drive in the range of A to Z is one of the following:

- Not available to the shared folders function
- Available to the shared folders function and not assigned
- Available to the shared folders function and assigned

Procedure Declaration

C

Statements to use in application programs:

```
#include <SFLR.H>

char far *drvptr;             /*pointer to drive status array     */
```

Declarations provided in include files:

```
unsigned pascal sfgds( drvptr );
```

PASCAL

Statements to use in application programs:

```
{$include: 'SFLR_C.INC'} { Constant section include file               }
{$include: 'SFLR_T.INC'} { Type section include file                   }
{$include: 'SFLR_E.INC'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION  SFGDS(drvptr : drivarrptr) : WORD;

drive_stat = STRING(26);
drivarrptr = ADS of drive_stat;
```

Parameters

drvptr Pointer to drive status array. The drive status array used for the get drive status routine is a 26 element array of characters. Each element corresponds to a drive letter from A to Z. On output from the routine, if the request completed successfully, the status of each drive is defined by the value of its corresponding array element as follows:

X'30' - The drive is not available to the shared folders function.
X'31' - The drive is available to the shared folders function, but is not assigned.
X'32' - The drive is available to the shared folders function and is assigned.

Return Codes

Return Code	Description
X'0000'	SF_SUCCESS - Successful.
X'0006'	SF_NO_DEVICE_DRIVER - Shared folders function has not been started.
X'0060'	SF_FUNCTION_NOT_SUPPORTED - The shared folders program running does not support this function. The shared folders function for extended DOS must be used.

Query Assigned Folder (QRYASSF)

This procedure can be used with the shared folders function for extended DOS support or for the OS/2 operating system.

Purpose

Queries a drive letter to determine if it is assigned as a shared folders function drive. If the drive is assigned, the system name, the folder name length, and the folder name are returned in the data area pointed to by the qassf pointer. The system name will contain a maximum of 8 characters plus one additional NULL character. The folder name will contain a maximum of 63 characters plus one additional NULL character.

Procedure Declaration

C

Statements to use in application programs:

```
#include <SFLR.H>

char drv;                          /* Drive letter -- A (a) to Z (z)     */
struct flr_info far *qassf;        /* Far address pointing to a buffer in*/
                                   /* which to store the returned data   */
```

Declarations provided in include files:

```
unsigned pascal qryassf(drv, qassf)

#define   FLRLEN         128
#define   SYSLEN           9

struct flr_info { /* parameter list */
     char sysname[SYSLEN];
     int  folder_len;          /* Actual length of folder name */
     char folder_name[FLRLEN];
};
```

PASCAL

Statements to use in application programs:

```
{$include: 'SFLR_C.INC'} { Constant section include file               }
{$include: 'SFLR_T.INC'} { Type section include file                   }
{$include: 'SFLR_E.INC'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION QRYASSF(drv : CHAR ; qassf : flr_info_ptr) : WORD;

CONST
  SYSLEN =   9;                                { AS/400 system name length   }
  FLRLEN = 128;                                { Folder name length          }

TYPE
  flr_info = RECORD
     sysname[0]              : STRING(SYSLEN);
     folder_len[SYSLEN]      : INTEGER;
     folder_name[SYSLEN + 2] : STRING(FLRLEN);
  END;

flr_info_ptr = ADS of flr_info;
```

Parameters

drv — Drive letter. Valid drive letters are A (a) to Z (z).

qassf — A pointer to the flr_info structure where the return data is to be stored.

Return Codes

Return Code	Description
X'0000'	SF_SUCCESS - Successful.
X'0001'	SF_DRV_NOT_DEF - Drive is not defined.
X'0009'	SF_BUFFER_TOO_SMALL - The specified buffer size is not large enough.
X'000F'	SF_DRV_NOT_VALID - The drive specified is not valid.
X'001F'	SF_COMMUNICATION_ERR - Communications error occurred.
X'005E'	SF_ROUTER_NOT_LOADED - PC Support router is not loaded.

Query Folder Names (QRYFLR)

This procedure can be used with the shared folders function for extended DOS support or for the OS/2 operating system.

Purpose

Queries an AS/400 folder to get a list of the names of all folders within that folder.

Procedure Declaration

C

Statements to use in application programs:

```
#include <SFLR.H>

char far *sysnaptr,   /* Pointer to the AS/400 system name           */
     far *nameptr;    /* Far address pointing to buffer that contains */
                      /* an ASCIIZ folder name on input and a list of */
                      /* the folder names within that folder on output*/

unsigned int bufsize; /* The size of the specified buffer            */
```

Declarations provided in include files:

```
unsigned pascal qryflr(sysnaptr, nameptr, bufsize)
```

PASCAL

Statements to use in application programs:

```
{$include: 'SFLR_C.INC'} { Constant section include file                }
{$include: 'SFLR_T.INC'} { Type section include file                     }
{$include: 'SFLR_E.INC'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION QRYFLR(sysnaptr : ADSMEM; nameptr : ADSMEM ; bufsize : WORD ) : WORD;
```

Parameters

sysnaptr
: A pointer to the AS/400 system name; an ASCIIZ string. If this is a null pointer, the default system is used.

nameptr
: A pointer to the buffer area. On input, the buffer should contain the ASCIIZ folder name of the folder to be queried. Do not include a drive, colon, or backslash before the folder name. If no folder name is specified, all folder names at the root level are returned.

bufsize
: The size of the specified buffer.

nameptr
: A pointer to the buffer area. On output, the buffer contains a list of folder names as follows:

  - Bytes 0 to 1: Number of folder names.
  - Bytes 2 to bufsize-1: Folder names, each ending with a NULL character (X'00').

Return Codes

Return Code	Description
X'0000'	SF_SUCCESS - Successful.
X'0003'	SF_FLR_NAME_NOT_FOUND - Folder name not found on host system.
X'0004'	SF_FLR_NAME_NOT_VALID - Folder name not valid.
X'0005'	SF_NOT_AUTHORIZED_TO_FOLDER - Not authorized to the specified folder name.
X'0008'	SF_NOT_ENOUGH_PC_RESOURCE - Not enough PC resources to assign a drive.
X'0009'	SF_BUFFER_TOO_SMALL - The specified buffer size is not large enough.
X'0015'	SF_DRV_NOT_READY - OS/2 shared folders function has not been started.
X'001F'	SF_COMMUNICATION_ERR - Communications error occurred.
X'003B'	SF_NETWORK_ERROR - An error occurred in the communications layer.
X'0054'	SF_NO_DRV_AVAILABLE - No drives are available.
X'005B'	SF_VERSION_NOT_MATCH - Version mismatch between host and PC program levels.
X'005C'	SF_SYSTEM_NOT_VALID_OR_ACTIVE - System name is not valid or system is inactive.
X'005D'	SF_CM_NOT_ACTIVE - Communications manager is not active.
X'005E'	SF_ROUTER_NOT_LOADED - PC Support router is not loaded.
X'005F'	SF_LOCAL_LU_NOT_ACTIVE -Local LU specified in CONFIG.PCS is not correct.
X'0060'	SF_FUNCTION_NOT_SUPPORTED - The shared folders program running does not support this function. The shared folders function for extended DOS must be used.
X'0061'	SF_SYS_NAME_NOT_VALID - System name too long.
X'0062'	SF_SYS_CONTACT_ENDED - The data link was lost or an error occurred on the host system.
X'0063'	SF_SYS_RESOURCE_FAILURE - A resource error occurred on the host system.
X'00FC'	SF_NOT_LOADED - EHNSFL0.DLL was not loaded.

Check In File In Folder (SFCHKIN)

This procedure can be used with the shared folders function for extended DOS support or for the OS/2 operating system.

Purpose

Marks a file that is pointed to by TARGET on a folder as checked in, so other users are allowed to update or delete it. It also copies the file pointed to by SOURCE back to the folder, if specified.

This procedure is used after the file has been checked out using the SFCHKOUT routine (see page 14-12).

Procedure Declaration

C

Statements to use in application programs:

```
#include <SFLR.H>

struct file_str far *SOURCE, far *TARGET;
```

Declarations provided in include files:

```
unsigned pascal sfchkin( TARGET, SOURCE);

#define   PATHLEN       128
#define   NAMELEN        13

struct file_str {
   char   drive;                /* drive letter                    */
   char   path[PATHLEN];        /* path name 128 chars             */
   char   fname[NAMELEN];       /* file name 13 chars              */
} ;
```

PASCAL

Statements to use in application programs:

```
{$include: 'SFLR_C.INC'} { Constant section include file               }
{$include: 'SFLR_T.INC'} { Type section include file                   }
{$include: 'SFLR_E.INC'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION SFCHKIN(TARGET, SOURCE: file_str_ptr): WORD;

CONST
  NAMELEN = 13;
  PATHLEN = 128;

TYPE
  file_str=
    RECORD
      drive[0]           : CHAR;            { drive letter}
      path[1]            : STRING(PATHLEN); { path name   }
      fname[1 + PATHLEN] : STRING(NAMELEN); { file name   }
    END;

  file_str_ptr = ADS of file_str;
```

Parameters

TARGET — A pointer to the name of the file to be checked in. If the drive letter or path name (or both) is not specified, the system uses the current drive and path.

SOURCE — A pointer to the file to be copied to the target folder. If no file name is specified but either a drive or a path (or both) is specified, the name of target file is used.

Return Codes

Return Code	Description
X'0000'	SF_SUCCESS - Successful.
X'0003'	SF_PATH_NOT_FOUND - Path not found.
X'0004'	SF_CANNOT_COPY_ITSELF - Cannot copy a file onto itself.
X'0005'	SF_NO_SPACE_TO_COPY - Insufficient space to perform copy.
X'0006'	SF_FILE_NOT_FOUND - File not found.
X'0008'	SF_FILE_IN_USE - File is in use.
X'000A'	SF_NOT_CHECKED_OUT_TO_YOU - File is not checked out to you.
X'000B'	SF_FILE_NOT_CHECKED_OUT - File is not checked out.
X'000C'	SF_FILE_NOT_SPECIFIED - Host file name is not specified.
X'000F'	SF_DRV_NOT_VALID - The drive specified is not valid.
X'0055'	SF_DRV_NOT_ASSIGNED - Drive has not been assigned.
X'005B'	SF_VERSION_NOT_MATCH - Version mismatch between host and PC code levels.

Check Out File in Folder (SFCHKOUT)

This procedure can be used with the shared folders function for extended DOS support or for the OS/2 operating system.

Purpose

Marks a host file pointed to by SOURCE as checked out, so other users are not allowed to update or delete it. It also makes a copy of the host file to the file name pointed to by TARGET, if specified.

The file can be checked in again using the SFCHKIN routine (see page 14-10).

Procedure Declaration

C

Statements to use in application programs:

```
#include <SFLR.H>

struct file_str far *SOURCE, far *TARGET;
```

Declarations provided in include files:

```
unsigned pascal sfchkout(SOURCE, TARGET);

#define   PATHLEN       128
#define   NAMELEN        13

struct file_str {
   char   drive;                /* drive letter                    */
   char   path[PATHLEN];        /* path name 128 chars             */
   char   fname[NAMELEN];       /* file name 13 chars              */
} ;
```

PASCAL

Statements to use in application programs:

```
{$include: 'SFLR_C.INC'} { Constant section include file               }
{$include: 'SFLR_T.INC'} { Type section include file                   }
{$include: 'SFLR_E.INC'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION SFCHKOUT(SOURCE, TARGET: file_str_ptr): WORD;

CONST
  NAMELEN = 13;
  PATHLEN = 128;

TYPE
  file_str=
    RECORD
      drive[0]           : CHAR;            { drive letter}
      path[1]            : STRING(PATHLEN); { path name   }
      fname[1 + PATHLEN] : STRING(NAMELEN); { file name   }
    END;

  file_str_ptr = ADS of file_str;
```

Parameters

SOURCE — A pointer to the host file name to be checked out. If the drive letter and path name are not specified, the system uses the current drive and path.

TARGET — A pointer to the name of the file to copy the source file into. If no file name is specified, but either a drive letter or a path (or both) is specified, the name of the source file is used.

Return Codes

Return Code	Description
X'0000'	SF_SUCCESS - Successful.
X'0003'	SF_PATH_NOT_FOUND - Path was not found.
X'0004'	SF_CANNOT_COPY_ITSELF - Cannot copy a file onto itself.
X'0005'	SF_NO_SPACE_TO_COPY - Insufficient space to perform copy.
X'0006'	SF_FILE_NOT_FOUND - File was not found.

Return Code	Description
X'0007'	SF_ALREADY_CHECKED_OUT - File is already checked out.
X'0008'	SF_FILE_IN_USE - File is in use.
X'000C'	SF_FILE_NOT_SPECIFIED - Host file name is not specified.
X'000F'	SF_DRV_NOT_VALID - The drive specified is not valid.
X'0055'	SF_DRV_NOT_ASSIGNED - Drive has not been assigned.
X'005B'	SF_VERSION_NOT_MATCH - Version mismatch between host and PC code levels.

Check Directory of Folder (SFCHKDIR)

This procedure can be used with the shared folders function for extended DOS support or for the OS/2 operating system.

Purpose

Lists all files and subdirectories that match any specified wildcard criteria pointed to by TARGET in a directory of a folder. This procedure lists the same information as the DOS or OS/2 DIR command. In addition, for all files that are checked out, it lists the user ID that has the file checked out and lists the date and time the file was checked out. All of the information is stored in a table pointed to by FLTBL_PTR. The variable N_FLS indicates the total number of entries in the table.

Procedure Declaration

C

Statements to use in application programs:

```
#include <SFLR.H>

struct file_str far *TARGET;
struct fl_dir far *FLTBL_PTR;
int far *N_FLS;
```

Declarations provided in include files:

```
unsigned pascal sfchkdir( TARGET, FLTBL_PTR, N_FLS);

#define   PATHLEN       128
#define   NAMELEN        13

struct file_str {
   char   drive;               /* drive letter                    */
   char   path[PATHLEN];       /* path name 128 characters        */
   char   fname[NAMELEN];      /* file name 13 characters         */
} ;

struct fl_dir {
   char      name[256],        /* file name                       */
             date[8],          /* file created date MM-DD-YY      */
             time[6],          /* file created time HH-MMp        */
             typ,              /* D means subdirectory name       */
             usr[10],          /* user id who check out the file  */
             chkdate[8],       /* check out date MM-DD-YY         */
             chktime[6];       /* check out time HH-MMp           */
             dummy;            /* for word boundary               */
   long      size;             /* file size                       */
} ;
```

PASCAL

Statements to use in application programs:

```
{$include: 'SFLR_C.INC'} { Constant section include file               }
{$include: 'SFLR_T.INC'} { Type section include file                   }
{$include: 'SFLR_E.INC'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION SFCHKDIR( TARGET : file_str_ptr; FLTBL_PTR  : FILESDIR_PTR;
                   VARS N_FLS : INTEGER) : WORD;

CONST
  NAMELEN = 13;
  PATHLEN = 128;
  MAX_FLS = 100;                {maximum number of files meet wildcard}
                                {criteria                              }

TYPE
  file_str=
    RECORD
      drive[0]           : CHAR;           { drive letter }
      path[1]            : STRING(PATHLEN); { path name    }
      fname[1 + PATHLEN] : STRING(NAMELEN); { file name    }
    END;

  file_str_ptr = ADS of file_str;

  fl_dir=                   { used for check in, out, dir, and user }
    RECORD
      name[0]      : STRING(256);  { file name                      }
      date[256]    : STRING(8);    { date file created MM-DD-YY     }
      time[264]    : STRING(6);    { time file created HH-MMp       }
      typ[270]     : BYTE;         { D means subdirectory name      }
      usr[271]     : STRING(10);   { user id who check out the file }
      chkdate[281] : STRING(8);    { check out date MM-DD-YY        }
      chktime[289] : STRING(6);    { check out time HH-MMp          }
      size[296]    : INTEGER4;     { file size                      }
    END;

  FILESDIR =ARRAY[1..MAX_FLS] of fl_dir;
  FILESDIR_PTR = ADS of FILESDIR;
```

Parameters

TARGET — A pointer to a specific file name or a subset of files (and subdirectories) for which the DIR information is requested. If no file name is specified, but either a drive letter or a path name (or both) is specified, all files and subdirectories on the path are listed.

N_FLS — A pointer to the number of entries in the table pointed to by FLTBL_PTR, with each of the files (or subdirectories) corresponding to one entry.

FLTBL_PTR — Returns the starting address of the file list table, which is prepared by the calling program.

N_FLS — Returns a pointer to the number of entries in the table pointed to by FLTBL_PTR, with each of the files (or subdirectories) corresponding to one entry.

Additional Information

Before calling this routine, prepare a table to receive a list of file information.

Return Codes

Return Code	Description
X'0000'	SF_SUCCESS - Successful.
X'0006'	SF_FILE_NOT_FOUND - File not found.
X'0009'	SF_BUFFER_TOO_SMALL - Table of file entries is too small.
X'000F'	SF_DRV_NOT_VALID - The drive specified is not valid.
X'0055'	SF_DRV_NOT_ASSIGNED - Drive has not been assigned.
X'005B'	SF_VERSION_NOT_MATCH - Version mismatch between host and PC code levels.

Check User of File in Folder (SFCHKUSR)

This procedure can be used with the shared folders function for extended DOS support or for the OS/2 operating system.

Purpose

Lists all files matching any specified wildcard criteria on a directory of a folder that are checked out by the specified user. If the value of KIND is NOTRAVERSE, this routine lists the same information as the DOS or OS/2 DIR command. In addition, for all files that are checked out, it lists the date and time the file was checked out. If the value of KIND is TRAVERSE, only the path and the file name are listed.

Procedure Declaration

C

Statements to use in application programs:

```
#include <SFLR.H>

struct file_str far *TARGET;
char far *USER;
enum trvs KIND;                    /* Specify traverse or not    */
struct fl_dir far *FLTBL_PTR;
int far *N_FLS;
```

Declarations provided in include files:

```
unsigned pascal sfchkusr( TARGET, USER, KIND, FLTBL_PTR, N_FLS);

#define   PATHLEN       128
#define   NAMELEN        13

enum trvs {
     TRAVERSE = 0,
     NOTRAVERSE = 1
} ;

struct file_str {
   char   drive;               /* drive letter                    */
   char   path[PATHLEN];       /* path name 128 chars             */
   char   fname[NAMELEN];      /* file name 13 chars.             */
} ;

struct fl_dir {
   char      name[256],        /* file name                       */
             date[8],          /* date file created MM-DD-YY      */
             time[6],          /* time file created HH-MMp        */
             typ,              /* D means subdirectory name       */
             usr[10],          /* user id who check out the file  */
             chkdate[8],       /* check out date MM-DD-YY         */
             chktime[6];       /* check out time HH-MMp           */
             dummy;            /* for word boundary               */
   long      size;             /* file size                       */
} ;
```

PASCAL

Statements to use in application programs:

```
{$include: 'SFLR_C.INC'} { Constant section include file              }
{$include: 'SFLR_T.INC'} { Type section include file                  }
{$include: 'SFLR_E.INC'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION SFCHKUSR( TARGET : file_str_ptr; USR_PTR  : USERID_PTR; KIND :
                   trvs; FLTBL_PTR : FILESDIR_PTR; VARS N_FLS :
                   INTEGER) : WORD;

CONST
  NAMELEN = 13;
  PATHLEN = 128;
  USERLEN = 9;
  MAX_FLS = 100;                {maximum number of files meet wildcard}
                                {criteria                              }

TYPE
  file_str=
    RECORD
      drive[0]           : CHAR;            { drive letter }
      path[1]            : STRING(PATHLEN); { path name    }
      fname[1 + PATHLEN] : STRING(NAMELEN); { file name    }
    END;

  file_str_ptr = ADS of file_str;

  trvs = (TRAVERSE, NOTRAVERSE); { used in check user                }

  fl_dir=                         { used for check in/out dir and user }
    RECORD
      name[0]      : STRING(256);{ file name                         }
      date[256]    : STRING(8);  { file created date MM-DD-YY        }
      time[264]    : STRING(6);  { file created time HH-MMp          }
      typ[270]     : BYTE;       { D means subdirectory name         }
      usr[271]     : STRING(10); { user id who check out the file    }
      chkdate[281] : STRING(8);  { check out date MM-DD-YY           }
      chktime[289] : STRING(6);  { check out time HH-MMp             }
      size[296]    : INTEGER4;   { file size                         }
    END;

  FILESDIR = ARRAY[1..MAX_FLS] of fl_dir;
  FILESDIR_PTR = ADS of FILESDIR;

  USERID   = STRING(USERLEN);
  USR_PTR = ADS of USERID;
```

Parameters

TARGET
: A pointer to a specific file name or a subset of files on the folder for which the user wants to know if any are checked out to the specified user. If a subset of files is needed, wildcard characters may be used. If the file name is not specified, all files in the directory are searched.

USER
: A pointer to the user ID of the person who checked out the files to be listed. If no user is specified, the user ID used to sign on to the router is used.

KIND
: Traverse option, described as follows:

  NOTRAVERSE
  : The function only searches for files within the specified directory and lists the detail information (name, size, date, time, and checked-out date/time) about them.

TRAVERSE
: The function lists files in the specified directory as well as all files in subdirectories below the specified directory that are checked out to the user. When this option is used, only the file names and path information are listed.

N_FLS
: A pointer to the total number of file list entries.

FLTBL_PTR
: Returns the starting address of the file list table which is prepared by the calling program.

N_FLS
: Returns a pointer to the total number of file list entries.

Additional Information

Before calling this routine, prepare a table to receive list file information.

Return Codes

Return Code	Description
X'0000'	SF_SUCCESS - Successful.
X'0002'	SF_INVALID_KIND - The value of KIND is not correct. The value must be either TRAVERSE or NOTRAVERSE.
X'0007'	SF_USER_NOT_FOUND_OR_VALID - User is not found or is not valid.
X'0008'	SF_NO_FILE_CHECKED_OUT - No files checked out to the user.
X'0009'	SF_BUFFER_TOO_SMALL - Table of file entries is too small.
X'000F'	SF_DRV_NOT_VALID - The drive specified is not valid.
X'0055'	SF_DRV_NOT_ASSIGNED - Drive has not been assigned.
X'005B'	SF_VERSION_NOT_MATCH - Version mismatch between host and PC code levels.

Chapter 15. Remote SQL Function High-Level Application Program Interface

This chapter describes the high-level language API for the remote SQL function of the PC Support licensed program. The API allows applications running in an extended DOS or an OS/2 environment to run SQL statements on a remote AS/400 system. The databases accessed may be either SQL database files or standard AS/400 files.

Additionally, support is provided to enable the following:

- A PC application program to start a program on a remote AS/400 system.
- An AS/400 program to start a program on a personal computer that is using the OS/2 operating system.
- Basic two-way communication between programs on the personal computer and the AS/400 system.

Note: Extended DOS for PC Support can be used only with personal computers with an Intel 80286 or greater microprocessor. All references to DOS in this chapter are for the extended DOS version of PC Support.

The API supports the following languages:

- IBM Pascal/2
- IBM C/2
- IBM Macro Assembler/2

Sample statements, procedure definitions, and data structure definitions for C and Pascal are provided in this chapter. An assembler language program could call the functions by sequence, so there are no such definitions given for assembler language.

Sample programs, include files, and other information for using APIs are provided in the PC Support tools folder (QIWSTOOL). See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

Remote SQL (RMTSQL) Function Overview

Remote SQL provides an API that allows a program running in an extended DOS or an OS/2 environment, and written in any language that can accommodate the PASCAL linkage conventions, to access the SQL database files or conventional database files on a remote AS/400 system. Additionally, functions are provided to enable basic program-to-program communications between a personal computer and an AS/400 system.

In both the extended DOS and OS/2 environments, the API is provided as a small library (LIB) that must be linked with the application program. The API for the extended DOS environment communicates with an OVL module residing in extended memory. This approach provides the following benefits:

- Eliminates the need for the application developer to provide the proper C runtime library.
- Limits the amount of real memory required.

- Helps with the installation of future program fixes or enhancements.

In the OS/2 environment, the API communicates with a dynamic link library (DLL).

The files EHNRQAPI.LIB and EHNRQAPI.OVL are found in the QIWSFL2 folder for single-byte DOS and in the QIWSFL2D folder for double-byte DOS. The files EHNRQAPI.LIB and EHNRQAPI.DLL are found in the QIWSOS2 folder for single-byte OS/2 and in the QIWSOS2D folder for double-byte OS/2.

Take Note

If you are performing operations on DBCS data, you must be using a DBCS personal computer, such as the PS/55. This is because the personal computer performs the conversion between ASCII and EBCDIC, and single-byte operating systems do not have the required conversion routines.

The results of performing operations on DBCS data from a SBCS personal computer are unpredictable. An error may be detected, or incorrect data may be generated.

Both the extended DOS and OS/2 versions communicate with a common AS/400 server program.

An example of the use of the API follows:

1. The PC application program establishes a conversation with a server program on a remote AS/400 system through a call to remote SQL (EHNRQSTART).
2. The application program then creates a valid SQL statement in the form of an ASCIIZ string and sends it to the remote system using a call to remote SQL.
3. The server program on the remote system prepares and runs the statement and returns the results to the application program.

   Note: If an SQL SELECT statement is sent, selected rows are logically returned one at a time with automatic data type conversions or translations.

An option is available that allows a PC application to do SQL table updates based on the SELECT cursor position (with some degradation in performance). The sample code on page 15-50 could be included in a PC program for a simple application.

A conversation may also be established in the opposite direction when OS/2 is being used. For example, when an AS/400 application program calls a cooperating application program on a personal computer:

1. The AS/400 application program calls the server program using a QPXXCALL call to the server and requests the server to call the remote PC application program.
2. The AS/400 program sends some initial data to the PC application program (using a QRQSTRRP call to the server) and then waits for a response from the PC application program.
3. The PC application program calls the remote SQL (EHNRQACCEPT) to do the following:

a. Complete the communication connection.
b. Issue a remote SQL call (EHNRQRECV) to accept the initial data.
c. Possibly issue calls to access databases on the AS/400 system.

4. Finally, the PC application program calls the remote SQL (EHNRQSEND) to return the results to the waiting AS/400 application program.

External User Interface

The remote SQL API has no direct user interface. However, it does interface with the following PC Support programs:

- PCSXI.EXE - PC Support Extended Memory Manager
- PCSWIN.EXE
- DOS PC Support Router
- OS/2 PC Support Router
- DOS PCSXLT.EXE (for DOS DBCS only)

The PCSXI.EXE program provides the extended memory management functions for PC Support. It must be started before an application may use remote SQL in the extended DOS environment.

The PCSWIN.EXE program provides memory management for PC Support with the Windows program. It is recommended that you use PCSWIN /B=48.

The PC Support router is used to establish the initial APPC connection with the host system. This eliminates the need for the application to ask the user for an ID and password if the router's user ID and password are to be used.

PCSXLT.EXE provides ASCII-to-EBCDIC translation support for the double-byte character set (DBCS). It must be loaded and used in conjunction with the PC Support router for the DBCS DOS environment. If it is not available, a DBCS translation error is returned by remote SQL.

Application Development Using the Remote SQL API

Following are items to consider when creating a PC application using the remote SQL API.

- Up to 10 cursors may be active at a given time. These may be any combination of EHNRQSELECT, EHNRQSELECTPM, EHNRQEXEC, EHNRQEXECPM, and EHNRQEXECST requests. If an EHNRQSELECT, EHNRQSELECTVAL, or EHNRQEXECST request is made in a mode other than update mode, calls for operations against cursors that are already active or for new operations may not be issued until: '
  - All of the rows resulting from the nonupdate mode SELECT have been retrieved.
  - The number of rows specified by an EHNRQSETROWS statement have been retrieved.
  - The cursor is closed or freed.
- When a nonzero return code is returned for an SQL operation, the SQLCA should be retrieved. If the SQLCODE is positive, it indicates the operation completed with a warning. If the SQLCODE is negative, the operation did not

complete. For a complete list of SQL return codes, refer to the *SQL/400* Reference*.

- Certain SQL statements may close all active cursors as a result of the function they perform. These statements include COMMIT, ROLLBACK, and CONNECT.
- An alternate sort sequence table can be specified in the user profile specified on the EHNRQSTART or EHNRQSTARTSEC API. This value will be determined by the job attributes at the start time.
- In order to use commit levels other than *NONE, requests must be executed against SQL collections or AS/400 database files with journaling active.
- To use DRDA support, the AS/400 must have the necessary communication configurations for the remote database. There must also be an entry in the Relational Database Directory for the local and remote databases. This can be added using the Add Relational Database Directory Entry (ADDRDBDIRE) command or the Work with Relational Database Directory Entry (WRKRDBDIRE) command. For more information on DRDA support, see the *Distributed Relational Database Guide*.
- While connected to a remote database using DRDA, it is important to check the SQLCA for a communication failure for any remote SQL operations that result in a nonzero return code. If a communications error occurs between the AS/400 and a remote database, the first remote SQL operation performed after the communications failure occurs is notified of the error. Any subsequent operations result in an error indicating that the application process is not in a connected state. To recover from the error, either issue a CONNECT TO to reconnect to the remote database or a CONNECT RESET to connect to the local database.
- For GRAPHIC and variable-length graphic fields, the length returned by EHNRQDESC is the number of double-byte characters and does not include the variable-length LL bytes or the null indicator, if present. For EHNRQATTR, the length represents the field size (in number of bytes) and does not include the variable-length LL bytes or the null indicator. For EHNRQFETCH, the variable-length LL bytes for variable-length graphic fields represent the number of double-byte characters.
- Link existing applications with the EHNRQAPI.LIB file again to take full advantage of the remote SQL support provided for this release. If no new support is desired, linking the application again is not required.

Creating Applications Using Program-to-Program Communications

If the program-to-program communications entry points are being used, there are some restrictions upon the order in which they are issued.

1. Only one remote program (excluding the AS/400 remote SQL server) can be communicated with at a time. Therefore, an AS/400 program that was called as a result of an EHNRQINVOKE by a PC application may not itself issue a QRQSTRRP call. On the personal computer, an EHNRQINVOKE may not be issued after an EHNRQACCEPT.
2. When a program has been called by a remote application (for example, after issuing an EHNRQACCEPT on the personal computer, or after the initial call into an AS/400 program), the first call must be an EHNRQRECV (or

QRQRCVDT) to receive any initial data sent by the program on the other system.

3. Due to the half-duplex nature of the underlying communications link, whenever a QRQRCVDT call (or EHNRQRECV) is done, it must be repeated until an End of Data return code is received, indicating that the partner program has stopped sending.
4. An AS/400 program that calls a remote PC application should issue a QRQENDRP call to end communications before it completes. An AS/400 program that was called by a remote PC application should end by returning to its caller.
5. A PC program may issue remote database requests anytime except:
   - After an EHNRQACCEPT until an End of Data response to a EHNRQRECV.
   - After an EHNRQSEND until an End of Data response to a EHNRQRECV.

SQL Statements Supported

The following SQL statements are supported:

COMMENT ON
COMMIT
CONNECT[4]
CREATE COLLECTION[3]
CREATE INDEX
CREATE TABLE
CREATE VIEW
DELETE
DROP COLLECTION[3]
DROP INDEX
DROP PACKAGE
DROP TABLE
DROP VIEW
GRANT PACKAGE
GRANT TABLE
INSERT
LABEL ON
LOCK TABLE
REVOKE PACKAGE
REVOKE TABLE
ROLLBACK
SELECT[1]
UPDATE[2]

The following SQL statements cannot be explicitly issued, but are rather implicit in the support provided by remote SQL.

CLOSE
DECLARE CURSOR
EXECUTE
EXECUTE IMMEDIATE
FETCH
OPEN
PREPARE

Extended DOS Environment

This section defines the API for the remote SQL function in PC programs running in the extended DOS environment.

Note: For information describing the high-level API for the remote SQL function for programs running in the Windows environment, see Chapter 25, "Remote SQL Windows Application Program Interface."

1 Run via a call to EHNRQSELECT.

2 Positioned updates are run via a call to EHNRQUPCUR.

3 AS/400 restriction: SQL collections only, not AS/400 libraries.

4 Run via a call to EHNRQCONNECT.

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the remote SQL function API:

EHNRQAPI.H C language include file
EHNRQAPI.INC Pascal language include file

Functions

The following functions are provided for PC applications:

- Close cursor (EHNRQCLOSE)
- Connect (EHNRQCONNECT)
- Delete current row (EHNRQDELETE)
- Describe (EHNRQDESC)
- End (EHNRQEND)
- Execute immediate (EHNRQEXEC)
- Execute stored (EHNRQEXECST)
- Execute with parameter markers (EHNRQEXECPM)
- Execute with parameter markers values (EHNRQEXECVAL)
- Fetch (EHNRQFETCH)
- Free statement with parameter markers (EHNRQFREEPM)
- Get formatted row data (EHNRQGETF)
- Prepare and store (EHNRQPREPST)
- Receive (EHNRQRECV)
- Retrieve attributes (EHNRQATTR)
- Retrieve SQLCA (EHNRQSQLCA)
- Retrieve message (EHNRQRTVMSG)
- Return error data (EHNRQERROR)
- Select (EHNRQSELECT)
- Select with parameter markers (EHNRQSELECTPM)
- Select with parameter markers values (EHNRQSELECTVAL)
- Send (EHNRQSEND)
- Set number of rows (EHNRQSETROWS)
- Set options (EHNRQOPTIONS)
- Start (EHNRQSTART)
- Start AS/400 program (EHNRQINVOKE)
- Start session with security (EHNRQSTARTSEC)
- Update current row (EHNRQUPCUR)

The following functions are provided for AS/400 applications:

- Retrieve error data (QRQRTVER)
- Receive data (QRQRCVDT)
- Send data (QRQSNDDT)

Communication Buffer Size

The optimal communication buffer size can be calculated using the following formula:

```
Optimal_size = comm_buffer_size * (max_buffer_size / comm_buffer_size)
```

- The comm_buffer_size is obtained by calling the Query_Router_Capabilities API.
- The max_buffer_size is 2K.
- The minimum comm_buffer_size is 276 bytes.

- Optimal_size will be an INTEGER value with a maximum value of 2048.

OS/2 Environment

This section defines the API for the remote SQL function in PC programs running in an OS/2 environment.

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the remote SQL function API:

EHNRQAPI.H C language include file
EHNRQAPI.INC Pascal language include file

Functions

The following functions are provided for PC applications:

- Accept host connection (EHNRQACCEPT)
- Close cursor (EHNRQCLOSE)
- Connect (EHNRQCONNECT)
- Delete current row (EHNRQDELETE)
- Describe (EHNRQDESC)
- End (EHNRQEND)
- Execute immediate (EHNRQEXEC)
- Execute stored (EHNRQEXECST)
- Execute with parameter markers (EHNRQEXECPM)
- Execute with parameter markers values (EHNRQEXECVAL)
- Fetch (EHNRQFETCH)
- Free statement with parameter markers (EHNRQFREEPM)
- Get formatted row data (EHNRQGETF)
- Prepare and store (EHNRQPREPST)
- Receive (EHNRQRECV)
- Retrieve attributes (EHNRQATTR)
- Retrieve SQLCA (EHNRQSQLCA)
- Retrieve message (EHNRQRTVMSG)
- Return error data (EHNRQERROR)
- Select (EHNRQSELECT)
- Select with parameter markers (EHNRQSELECTPM)
- Select with parameter markers values (EHNRQSELECTVAL)
- Send (EHNRQSEND)
- Set number of rows (EHNRQSETROWS)
- Set options (EHNRQOPTIONS)
- Start (EHNRQSTART)
- Start AS/400 program (EHNRQINVOKE)
- Start session with security (EHNRQSTARTSEC)
- Update current row (EHNRQUPCUR)

The following functions are provided for AS/400 applications:

- End remote program (QRQENDRP)
- Retrieve error data (QRQRTVER)
- Start remote program (QRQSTRRP)
- Receive data (QRQRCVDT)
- Send data (QRQSNDDT)

PC Support Remote SQL Function API

The following sections for the remote SQL API describe:

- Purpose
- Procedure and data structure declarations
- Parameters
- Returned values
- Return codes

Accept Host Connection (EHNRQACCEPT)

Purpose

Accepts an APPC connection initiated by an AS/400 program through the remote SQL server. A call to this entry point must be followed by one or more EHNRQRECV calls to receive the parameters provided by the AS/400 application.

Notes:

1. Only applications running under the OS/2 environment use this function.
2. See "Creating Applications Using Program-to-Program Communications" on page 15-4 for additional information about communicating with an AS/400 application beyond the base database server.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0
#define extentry _saveregs pascal far

extern int  extentry EHNRQACCEPT(char far *,       /* Transaction Program Name */
                                 unsigned short);  /* Buffer Size              */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQACCEPT(tpname : adsmem;
                     buf : integer) : integer; extern;
```

Parameters

tpname The transaction program identifier by which this program is to be known. This is an ASCIIZ string that must match the identifier provided by the host application on its QPXXCALL call.

buf The size of the buffer to be used for data transfers to and from the host. This value represents the maximum length (after translation) of a SQL statement provided on an EHNRQEXEC, EHNRQEXECPM, EHNRQSELECT, or EHNRQSELECTPM call, or constructed as a result of an EHNRQUPCUR, EHNRQEXECVAL, or EHNRQSELECTVAL call.

Note: On double-byte character set (DBCS) systems, the amount of storage required may increase somewhat during ASCII-to-EBCDIC translation due to the insertion of shift-in (SI) and shift-out (SO) characters.

It also represents the maximum amount of data that can be sent (after translation) by a single EHNRQSEND call or received by an EHNRQRECV call. The value provided is adjusted, if necessary, to force it to fall into the range from 1024 to 31 744 bytes. (For example, specifying 0 is valid and results in the use of a 1024-byte buffer). This parameter does not limit the amount of data that can be received by the application on a single EHNRQFETCH or EHNRQGETF call.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Retrieve Attributes (EHNRQATTR)

Purpose

Returns an array of entries that describe the attributes of the columns which are returned by the currently active SELECT statement. There is one entry in the column attributes table for each table column being returned. The information provided is essentially the same as that contained in the SQLDA control block, but in a simpler form. For example, the SQLLEN field of the SQLDA control block is sometimes the length of the corresponding column data, or is sometimes a combination of the precision and scale attributes. In the column attributes table returned by this entry point, there are separate fields for precision, scale, and data width. Additionally, there are offsets to the beginning of the fields within the result area returned by EHNRQFETCH.

Table row structure:

```
Col_Type:     2-byte binary
Col_Prec:     1-byte binary precision
Col_Scale:    1-byte binary scale factor
CCSID:        2-byte binary code page ID
reserved:     character(4)
Col_Width:    2-byte binary field width
Col_Data_Off: 2-byte Offset to data in result area
Col_Ind_Off:  2-byte Offset to indicator (zero if none)
Col_Name:     character(32) name, ASCIIZ string
```

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

typedef unsigned char hcursor;          /* Handle for select cursor         */

struct Col_Attributes
  {
   short Col_Type;                      /* SQL column type                   */
   unsigned char Col_Prec;              /* Column precision (arith types only)*/
   unsigned char Col_Scale;             /* Column scale (arith types only)   */
   char  CCSID[2];                      /* Reserved                          */
   char rsvd2[4];                       /* Reserved                          */
   short Col_Width;                     /* Source data width of column       */
   short Col_Data_Off;                  /* Offset to data in result area     */
   short Col_Ind_Off;                   /* Offset to indicator (zero if none) */
   char Col_Name[32];                   /* Column name (ASCIIZ string)       */
  };

extern int  extentry EHNRQATTR(hcursor,     /* Cursor handle              */
                               struct Col_Attributes far *,
                                            /* Attribute table pointer    */
                               Short far *);
                                            /* Attribute table size       */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;

  Col_Attr_Row =
    RECORD
      Col_Type     : integer;          {SQL column type                    }
      Col_Prec     : byte;             {Column precision (arith types only)}
      Col_Scale    : byte;             {Column scale (arith types only)    }
      CCSID        : string(2);        {Reserved                           }
      rsvd2        : string(4);        {Reserved                           }
      Col_Width    : integer;          {Source data width of column        }
      Col_Data_Off : integer;          {Offset to data in result area      }
      Col_Ind_Off  : integer;          {Offset to indicator (zero if none) }
      Col_Name     : string(32);       {Column name (ASCIIZ string)        }
    end;

  Col_Attr_Table = array[1..999] of Col_Attr_Row;

  Col_Attr_PtrTyp = ADS OF Col_Attr_Table;

function EHNRQATTR(          cur : hcursor;
                        Col_Attr : Col_Attr_PtrTyp;
                    vars attr_len : integer) : integer; extern;
```

Parameters

cur	The cursor handle corresponding to an active SELECT cursor (for example, returned by EHNRQSELECT or EHNRQSELECTPM, and the cursor is still open).
Col_Attr	A far pointer to an application-allocated area where the attribute table will be placed.
attr_len	The size in bytes of the area pointed to by Col_Attr.
attr_len	Returns the size in bytes of the area pointed to by Col_attr.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Close Cursor (EHNRQCLOSE)

Purpose

Used to end a SELECT operation and close the associated cursor. EHNRQCLOSE frees resources (and locks on table rows) in the host system and should be called when all of the selected rows have been retrieved through EHNRQFETCH or EHNRQGETF.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

typedef unsigned char hcursor;          /* Handle for select cursor         */

extern int  extentry EHNRQCLOSE(hcursor);
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;

function EHNRQCLOSE(cur : hcursor)  : integer; extern;
```

Parameters

cur	The cursor handle corresponding to an active SELECT cursor (for example, returned by EHNRQSELECT or EHNRQSELECTPM and the cursor is still open).

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Connect (EHNRQCONNECT)

Purpose

Explicitly connects to a remote SQL database and allows remote SQL to use Distributed Relational Data Access (DRDA) support available on the AS/400 system. More setup and usage information on DRDA is available in the *Distributed Database Guide*.

Procedure Declaration

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQCONNECT(   stmnt:   adsmem;

                         opt_struct: adsmem;
                    vars opt_len:  integer)  : integer; extern;
```

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0
#define extentry _saveregs pascal far

extern int  extentry EHNRQCONNECT(char far *,/* SQL statement     */
                                  struct Options_Struct far *,
                                  short far*); /* Opt struc length*/
```

Parameters

stmnt — A null-terminated character string that defines the SQL CONNECT statement to be run.

opt_struc — Points to the options structure allocated by and filled in by the calling program.

Notes:

1. To have this structure filled in with the current option values, run an EHNRQOPTIONS_QUERY before running the EHNRQCONNECT function.
2. If options are not required, specify a null pointer for the options structure.

See "Options (EHNRQOPTIONS)" on page 15-31 for more information on the options structure.

opt_len — The size in bytes of the area pointed to by the opt_struct parameter.

The remote SQL output from the execution of the EHNRQCONNECT statement is returned via the SQLCA. Therefore, if the application program is interested in this information, it should issue an EHNRQSQLCA API call immediately after the EHNRQCONNECT API. In addition, verify the SQL Parser Options values upon return to see whether any of the values have been changed.

Returned values are:

opt_struct — The option structure (if not null) contains the current settings of the options. Refer to "Options (EHNRQOPTIONS)" on page 15-31 for more information on the options structure.

opt_len — If the size of the opt_struc structure is too small, this value is set to the minimum size needed in bytes. If the size of the opt_struc is above the minimum size, this value is set to the size used.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Delete Current Row (EHNRQDELETE)

Purpose

Deletes the row at the current cursor position. EHNRQDELETE must be preceded by a successful EHNRQFETCH or EHNRQGETF, and the corresponding EHNRQSELECT or EHNRQSELECTVAL call must have specified a nonzero update flag.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

typedef unsigned char hcursor;          /* Handle for select cursor          */

extern int  extentry EHNRQDELETE(hcursor);
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;

function EHNRQDELETE(cur : hcursor) : integer; extern;
```

Parameters

cur — The cursor handle corresponding to an active SELECT cursor (that is, returned by EHNRQSELECT or EHNRQSELECTPM, and the cursor is still open).

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Describe (EHNRQDESC)

Purpose

Obtains the SQLDA structure describing the columns returned by a SELECT statement. This function might be called after a successful call to EHNRQSELECT or EHNRQSELECTVAL in order to determine the attributes of the columns, which are returned by calls to EHNRQFETCH.

Note: The EHNRQATTR function provides an alternative method of obtaining this same information.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

typedef unsigned char hcursor;             /* Handle for select cursor       *,


extern int  extentry EHNRQDESC(hcursor,    /* Cursor handle                  *,
                               struct sqlda far *,
                                           /* SQLDA control block pointer    *,
                               short far *);
                                           /* SQLDA control block size       *,
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;

function EHNRQDESC(          cur : hcursor;
                           sqlda : SQLDA_PTRTYP;
                    vars sqldalen : integer) : integer; extern;
```

Parameters

cur	The cursor handle corresponding to an active or prepared SELECT cursor (that is, returned by EHNRQSELECT or EHNRQSELECTPM, and the cursor is still open).
sqlda	A far pointer to an application-allocated area where the SQLDA control block is placed.
sqldalen	The size in bytes of the area pointed to by sqlda.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

End (EHNRQEND)

Purpose

Ends the RMTSQL environment. EHNRQEND should be called prior to ending an application using RMTSQL or restarting the remote SQL environment with another EHNRQSTART or EHNRQSTARTSEC call.

The EHNRQEND function ends communications with the AS/400 server and any applications started with the EHNRQINVOKE function. In the OS/2 environment, this function frees up any resources used by EHNRQAPP.DLL. In DOS, this function frees all extended memory used. If an EHNRQEND is not issued before exiting DOS, the extended memory used is not freed until an IPL is performed on the personal computer.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQEND(void);
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQEND : integer; extern;
```

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Return Error Data (EHNRQERROR)

Purpose

Returns the APPC major and minor error return codes associated with the last communication attempt with the host system. Normally, EHNRQERROR is issued to obtain the return codes for logging, or for use in an error message to indicate that an error had occurred during communications with the host system (for example, a return code of X'0020' from a previous EHNRQxxxx call).

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern void extentry EHNRQERROR(char far p[4],   /* Primary return code   */
                                char far s[8]);  /* Secondary return code */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
procedure EHNRQERROR(primary, secondary : adsmem); extern;
```

Parameters

primary	Returns a 4-character ASCII string (not null-ended) containing the APPC major return code from the last communications operation.
secondary	Returns an 8-character ASCII string (not null-ended) containing the APPC minor return code from the last communications operation.

Execute Immediate (EHNRQEXEC)

Purpose

Sends an SQL statement (other than a SELECT or CONNECT) to the host system to be prepared and run.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQEXEC(char far *);    /* SQL Statement          */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQEXEC(stmnt : adsmem) : integer; extern;
```

Parameters

stmnt — An ASCIIZ string that contains the SQL statement to be run. The maximum length of the string is limited to the buffer size specified on the EHNRQSTART or EHNRQSTARTSEC call.

Note: On DBCS systems, this limit applies to the length of the statement **after** it has been translated to EBCDIC.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Execute with Parameter Markers (EHNRQEXECPM)

Purpose

Similar to the Execute Immediate process, this process uses a parameter marker. A parameter marker is a question mark (?) that appears in a statement string of dynamic SQL statements. The question mark can appear where a host variable could appear if the statement string was a static SQL statement. Using 0 parameter markers is supported.

Using statements with parameter markers enhances performance by allowing you to prepare the statement once and then execute it many times using different sets of values for the parameter markers.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far
typedef unsigned char hcursor;

extern int  extentry EHNRQEXECPM (char far *,     /* SQL Statement          */
                                  hcursor far *); /* Statement handle       */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;

function EHNRQEXECPM(   stmnt : adsmem;
                      vars cur : hcursor)  :  integer; extern;
```

Parameters

stmnt
: An ASCIIZ string that contains the SQL statement (containing parameter markers) to be run. The maximum length of the string is limited to the buffer size specified on the EHNRQSTART or EHNRQSTARTSEC call.

  Note: On DBCS systems, this limit applies to the length of the statement **after** it has been translated to EBCDIC.

cur
: Returns a handle that must be supplied on succeeding EHNRQEXECVAL calls which reference the parameter markers associated with this statement.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Execute Stored (EHNRQEXECST)

Purpose

This API sends multiple requests for the execution of SQL statements stored in an SQL package on the AS/400 system.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

struct execst_parms
 {
  char *lib_name;
  char *pkg_name;
  short number_of_stmts;
  unsigned  short quit_on_warning;
  struct execst_data
   {
   short stmt_number;
   hcursor *c_hndl;
   unsigned short update;
   char *format;
   void *v_array;
   } execst_data[1];
 };

extern int extentry EHNRQEXECST(struct execst_parms far * );
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  EXECST_PTRTYP = ADS OF RECORD
         lib_name : adsmem;
         pkg_name : adsmem;
         number_of_stmts : integer;
         quit_on_warning : word;
         execst_data : ARRAY [1..256] OF
                       RECORD
              stmt_number : integer;
              c_hndl : ADS OF hcursor;
              update : word;
              format : adsmem;
              v_array : vaptr;
         END;
  END;

function EHNRQEXECST( execst_parms : EXECST_PTRTYP) : integer; extern;
```

Parameters

lib_name — Points to a null-terminated string containing the library name. This name must be between one and ten characters long and have no blanks or wildcard characters. Library QTEMP can be specified if the SQL package already exists in that library.

pkg_name — Points to a null-terminated string containing the name of the SQL package. This name must be between one and ten characters long and have no blanks or wildcard characters.

number_of_stmts — The number of statements to be processed (must be greater than zero). This field is also used to return the number of statements that are successfully executed.

quit_on_warning — Indicates how SQL warning message will be handled.

- TRUE — Quit processing when an SQL warning is encountered.
- FALSE — Ignore SQL warnings.

execst_data — Points to execst_data structure.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

After a return from EHNRQEXECST with a nonzero return code, verify that any cursor handles opened by the call contain nonzero values. If the cursor handle is zero, the cursor is closed.

Additional Information

Stored SQL statements can be executed in any order specified by the parameters passed to this API. All the statements executed in a single call must exist in the same SQL package.

If the quit_on_warning option is set to TRUE, the host system treats warnings the same as errors. If a warning occurs, the host system will stop processing the EHNRQEXECST request. To process errors promptly, you should try to limit the

number of stored statements executed by an EHNRQEXECST request in your application. If the quit_on_warning option is set to FALSE, the host system ignores warnings and continues processing until an error occurs or until it has finished processing the EHNRQEXECST request.

Also, EHNRQEXECST does not support remote databases. In this situation, a remote database is one that is accessed by a call to EHNRQCONNECT.

Execute with Values for Parameter Markers (EHNRQEXECVAL)

Purpose

Sends parameter values for a prepared EHNRQEXECPM statement and then runs the statement.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0
#define extentry _saveregs pascal far

typedef unsigned char hcursor;

extern int extentry EHNRQEXECVAL(hcursor,      /* SQL Statement            */
                                 char far *,   /* Format string            */
                                 void far *);  /* Variables array pointer */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;
  varray = [1..100] of adsmem;
  vaptr = ads of varray;

function EHNRQEXECVAL(      cur : hcursor;
                            fmt : adsmem;
                            vap : vaptr)  : integer; extern;
```

Parameters

cur — The handle corresponding to an active execute with parameter marker statement (that is returned by EHNRQEXECPM).

fmt — An ASCIIZ string of format items that defines the data types to be used for each of the parameter markers specified in the EHNRQEXECPM statement. The number of blank-delimited format items defines the number of elements required in the array of pointers passed as the third parameter.

One format item must be provided for each parameter marker that is specified in the EHNRQEXECPM call. The allowable format items are:

- A — The corresponding variable is a far, ASCIIZ string. The corresponding SQL column must be a CHAR, VAR CHAR, LONG VAR CHAR, DATE, TIME, or TIMESTAMP data type.
- F — The corresponding variable is a FLOAT type. The corresponding SQL column may be any of the SQL numeric data types (SMALLINT, INTEGER, NUMERIC, DECIMAL, single-precision or double-precision FLOAT).
- D — The corresponding variable is a DOUBLE type. The corresponding SQL column may be any of the SQL numeric data types.
- G — The corresponding variable is a far, null-terminated string. The corresponding SQL data type must be GRAPHIC or VAR GRAPHIC. This format is valid for DBCS only.
- L — The corresponding variable is a LONG integer type. The corresponding SQL column may be any of the SQL numeric data types.
- S — The corresponding variable is a SHORT integer type. The corresponding SQL column may be any of the SQL numeric data types.

vap — An array of far pointers to variables of the type indicated by the format string items. The values of these variables is inserted into the SQL statement specified in the EHNRQEXECPM in place of the parameter markers. If a value of NULL is to be used, a NULL pointer must appear in the array of pointers.

Return Codes

If the return code is negative, then a format item that is not valid was found. The absolute value of the return code indicates which format item was being processed when the error was discovered.

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Fetch (EHNRQFETCH)

Purpose

Retrieves the first or next row of data satisfying a previously prepared and run SELECT statement. It differs from the EHNRQGETF operation described on page 15-28 in that it retrieves the raw data as returned by SQL on the remote system. (Raw data does not have any conversion of zoned or decimal data, and the block of data returned is not parsed into individual fields or variables.) The benefits of using EHNRQFETCH are as follows:

- The number and type of columns being returned need not be known in advance.
- The access to the raw data may enable its use without the loss of precision that could occur during data conversions performed automatically by EHNRQGETF (for example, when converting FLOAT or scaled DECIMAL data to INT).

A disadvantage of using EHNRQFETCH is that more detailed knowledge of SQL data formats and control blocks is required, as well as the need to provide explicit data conversions.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

typedef unsigned char hcursor;              /* Handle for select cursor        */

extern int  extentry EHNRQFETCH(hcursor,    /* Cursor handle                   */
                                char far *, /* Receive column value buffer ptr */
                                short far *);
                                            /* Receive column value size       */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;

function EHNRQFETCH(          cur : hcursor;
                             rcvr : adsmem;
                     vars rcvr_len : integer) : integer; extern;
```

Parameters

cur

The cursor handle corresponding to an active SELECT cursor (that is, returned by EHNRQSELECT or EHNRQSELECTPM and the cursor is still open).

rcvr

If a row was successfully fetched, the column values (and possibly indicators) retrieved are stored in the area indicated by this pointer. The area pointed to must be of sufficient size, as indicated by the rcvr_size variable, to contain the returned data.

The size and layout of the result area are as follows:

1. Result columns are returned in the order in which they were described in the SQLDA returned by EHNRQDESC or the attribute table returned by EHNRQATTR.
2. Column values that are an odd number of bytes in length are followed by a single pad byte (each column starts on an even byte boundary).
3. If a column has an associated indicator, the indicator is stored immediately after the column value data (and its trailing pad byte, if present). If a column has no associated indicator, no space is reserved. See Figure 15-1 on page 15-27 for an example of a SELECT statement and how the corresponding result area is laid out.

Note: EHNRQATTR can be used to get a table that contains offsets to the start of the column data in the result area. Also, the *Col_Ind_Off* record of the structure returned by EHNRQATTR indicates whether or not an indicator is in the column.

rcvr_len

The size in bytes of the buffer pointed to by the rcvr parameter.

rcvr_len

Returns the size in bytes of the buffer pointed to by the rcvr parameter. If the size specified is too small, then no data is retrieved and the rcvr_len parameter is modified to contain the minimum size required.

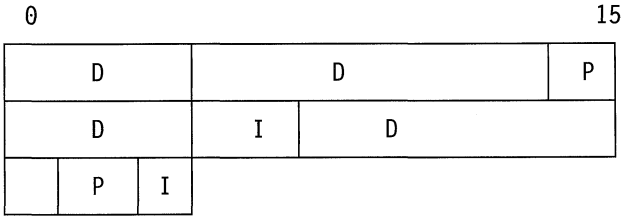
Suppose a SELECT statement similar to the following is run:

```
     SELECT A,B,C,D FROM SAMPLE_TABLE
```

Where:

- A is an integer column with no associated indicator
- B is a fixed character column of width 11 with no associated indicator
- C is an integer column with an associated indicator
- D is a fixed character column of width 11 with an associated indicator

A result area of 36 bytes would be returned, laid out as follows:



where:

- D = Column data
- I = Indicator
- P = Pad byte

Figure 15-1. Sample Result Area for a FETCH

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Free Statement with Parameter Markers (EHNRQFREEPM)

Purpose

Closes or frees either a prepared select or nonselect statement with parameter markers. This is mainly for remote SQL internal tracking and there is nothing to close or free in SQL for the existing prepared statement. This allows remote SQL to reuse handles.

If the handle is for a prepared select statement and the cursor is currently open for that statement, then the cursor will be closed and the handle will be freed.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0
#define extentry _saveregs pascal far

typedef unsigned char hcursor;

extern int ententry EHNRQFREEPM(hcursor);  /* Handle for prepared */
                                           /*  cursor             */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;

function EHNRQFREEPM(handle : hcursor) : integer; extern;
```

Parameters

handle — The handle corresponding to the previously prepared statement (returned by EHNRQSELECTPM or EHNRQEXECPM).

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Get Formatted (EHNRQGETF)

Purpose

An alternative to EHNRQFETCH, described on "Fetch (EHNRQFETCH)" on page 15-25. Rather than yielding a selected SQL table row in raw data form, the selected column values are converted to the appropriate data types, if necessary, and are stored directly into specified variables. If EHNRQGETF is used you are not required to do EHNRQDESC or EHNRQATTR calls, or to be aware of the precise format of the data as provided by SQL. Knowledge of the number of columns and their general type (numeric or character) is still required.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

typedef unsigned char hcursor;              /* Handle for select cursor      */

extern int  extentry EHNRQGETF(hcursor,     /* Cursor handle                 */
                               char far *,  /* Format string                 */
                               void far *); /* Variables array pointer       */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;
  varray = array[1..100] of adsmem;
  vaptr = ads of varray;

function EHNRQGETF(cur : hcursor;
                   fmt : adsmem;
                   vap : vaptr)  : integer; extern;
```

Parameters

cur The cursor handle corresponding to an active SELECT cursor (that is, returned by EHNRQSELECT or EHNRQSELECTPM and the cursor is still open).

fmt An ASCIIZ string of format items that define the number and type of parameters to receive column values. The number of blanked-delimited format items defines the number of additional parameters that must be passed. The operation ends when any one of the following conditions occurs:

- All of the format items are processed.
- All of the selected columns have been processed
- A format item is encountered that is incompatible with the type of the corresponding SQL column.
- Loss of significance occurs while converting or storing a column value.

The allowable format items are:

Annn The corresponding variable is a far pointer to an area to receive an ASCIIZ string with a maximum length of nnn (including the ending null). The corresponding SQL data type must be a CHAR, VAR CHAR, LONG VAR CHAR, DATE, TIME, TIMESTAMP data type. If the column data is longer than the output string, it is truncated without warning.

Gnnn The corresponding variable is a far, null-terminated string. The corresponding SQL data type must be GRAPHIC or VAR GRAPHIC.

F — The corresponding variable is a FLOAT type. The corresponding SQL data type may be any of the SQL numeric data types (SMALLINT, INTEGER, NUMERIC, DECIMAL, single-precision or double-precision FLOAT). Loss of precision may result if the precision or scale of the column value is greater than can be accommodated.

Note: Floating point numbers are stored in the IEEE format.

D — The corresponding variable is a DOUBLE (8-byte float) type. The corresponding SQL data type may be any of the SQL numeric data types. Loss of precision may result if the precision or scale of the column value is greater than can be accommodated.

Note: Floating point numbers are stored in the IEEE format.

L — The corresponding variable is a LONG (4-byte) integer type. The corresponding SQL data type may be any of the SQL numeric data types. Loss of precision results if the column value has a nonzero scale or a fractional part.

S — The corresponding variable is a SHORT (2-byte) integer type. The corresponding SQL data type may be any of the SQL numeric data types. Loss of precision results if the column value has a nonzero scale or a fractional part.

vap — Returns an array of far pointers to the variables into which the converted column values are to be stored. There must be a pointer entry in the array for each item in the format list.

Return Codes

If the return code is negative, then a format item that was not valid was found, or the indicated data type is incompatible with the data type of the corresponding SQL column. The absolute value of the return code indicates which format item was being processed when the error was discovered.

Note: If the return code indicates a format item is in error, the current record is not returned. The next call returns the next record.

If the return code is zero or positive, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Start AS/400 Program (EHNRQINVOKE)

Purpose

Causes the AS/400 server to invoke the specified AS/400 program. This second program may then use the server's QRQRCVDT and QRQSNDT entry points to receive parameters from the PC application and to return results to it. See "Creating Applications Using Program-to-Program Communications" on page 15-4 for additional information about communicating with an AS/400 application beyond the base database server.

Purpose

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQINVOKE(char far *);   /* Library name/program name   */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQINVOKE(pgm : adsmem) : integer; extern;
```

Parameters

pgm A string of the form library_name/program_name identifying the AS/400 program to be started. This string should be either an ASCIIZ string or padded with blanks to the full 21-byte length (22 bytes are allowed if the last byte is a null).

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Options (EHNRQOPTIONS)

Purpose

Allows applications to query and set SQL parser options. The parser options currently supported are:

- Commitment control level
- Date format
- Date separator
- Decimal separator
- Naming convention
- Time format
- Time separator

Note: Options can be queried any time there is no active block mode select or a pending send or receive. Options can be set only before any SQL statements are processed.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQOPTIONS(int,            /* Operation              */
                                  char far *,     /* Data buffer address    */
                                  int far *);     /* Data length            */
```

The options structure (for C language) follows. The options are 2 bytes each.

```
struct    Options_Struct
 {
  char name_conven_indicator;
  char name_conven_value;

  char comm_contrl_indicator;
  char comm_contrl_value;

  char date_format_indicator;
  char date_format_value;

  char date_separt_indicator;
  char date_separt_value;

  char time_format_indicator;
  char time_format_value;

  char time_separt_indicator;
  char time_separt_value;

  char decimal_point_indicator;
  char decimal_point_value;
 }
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQOPTIONS(     option_ind:  short;
                           option_struct:  adsmem;
                     vars opt_len:     integer; extern;
```

The options structure (for PASCAL language) follows. The options are 2 bytes each.

```
Options_Struct = Record;
 {
  comm_contrl_indicator       : byte;
  comm_contrl_value           : byte;

  name_conven_indicator       : byte;
  name_conven_value           : byte;

  date_format_indicator       : byte;
  date_separt_value           : byte;

  time_format_indicator       : byte;
  time_format_value           : byte;

  time_separt_indicator       : byte;
  time_separt_value           : byte;

  decimal_point_indicator     : byte;
  decimal_point_value         : byte;
end;
```

Parameters

option_ind

Indicates whether a SET or QUERY option is performed.

0 RQ_OPTION_QUERY
1 RQ_OPTION_SET

opt_struct Identifies the size in bytes of the area pointed to by the opt_struct parameter. This is the size of Options_Struct.

For a SET operation, the indicator field for each indicator value pair can have one of the following input values:

indicator = 0 The current value of this option is not changed.

indicator = 1 The current value of this option is changed.

indicator = n (where n is any other value) Is interpreted as if it were a zero (0) and, therefore, the current value is not changed.

For the possible values for each parser option, see "Remote SQL Parser Options" on page 15-52.

For a QUERY operation, the indicator field for each pair is ignored.

opt_len Identifies the size of the opt_struct structure. If the size is too small, this value is set to the minimum size required in bytes. If the size is more than the minimum size required, this value is not changed.

The returned values are:

opt_struct For a SET operation, the indicator field for each pair can have one of the following output values:

indicator = 0 The value specified for this option is valid.

indicator = E The value specified for this option is not valid.

For a QUERY operation, the indicator field for each pair has the following output value:

indicator = 0 The value specified for this option has been returned successfully.

For a list of the option values, see "Remote SQL Parser Options" on page 15-52.

opt_len Specifies the size of the returned structure.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Prepare and Store (EHNRQPREPST)

Purpose

Prepare several SQL statements and store them in an SQL package on the host AS/400 system.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

 struct prepst_parms
 {
  char *lib_name;
  char *pkg_name;
  short number_of_stmts;
  char create_package;
  unsigned  short quit_on_warning;
  struct prepst_data
   {
   short stmt_number;
   char *sql_stmt;
   }prepst_data[1];
 };

extern int extentry EHNRQPREPST(struct prepst_parms far *);
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE

  PREPST_PTRTYP = ADS OF RECORD
         lib_name : adsmem;
         pkg_name : adsmem;
         number_of_stmts : integer;
         create_package : byte;
         quit_on_warning : word;
         prepst_data : ARRAY [1..256] OF
                       RECORD
              stmt_number : integer;
              sql_stmt : adsmem;
         END;
  END;

function EHNRQPREPST( execst_parms : PREPST_PTRTYP) : integer; extern;
```

Parameters

lib_name — Points to a null-terminated string containing the library name. This name must be between one and ten characters long and have no blanks or wildcard characters. The user can specify QTEMP as the library for the SQL package. In this situation, the package will be deleted when the job has been completed.

pkg_name — Points to a null-terminated string containing the name of the SQL package. This name must be between one and ten characters long and have no blanks or wildcard characters.

number_of_stmts — The number of statements to be processed (must be greater than zero). This field is also used to return the number of successfully stored statements.

create package — Indicates whether or not a package will be created.

- Y — Create a new package.
- N — Store the statements in an existing package.
- A — If the package already exists, store the statements in the existing package.

quit_on_warning — Indicates how SQL warning message will be handled.

- TRUE — Quit processing when an SQL warning is encountered.
- FALSE — Ignore SQL warnings.

prepst_data — Points to prepst_data structure.

Additional Information

A new package will be created, or an existing package accessed, on the host AS/400 system specified on the call to the EHNRQSTART API.

If the quit_on_warning option is set to TRUE, the host system treats warnings the same as errors. If a warning occurs, the host system will stop processing the EHNRQPREPST request. To process errors promptly, you should try to limit the number of stored statements executed by an EHNRQPREPST request in your application. If the quit_on_warning option is set to FALSE, the host system ignores

warnings and continues processing until an error occurs or until it has finished processing the EHNRQPREPST request.

Also, EHNRQPREPST does not support remote databases. In this situation, a remote database is one that is accessed by a call to EHNRQCONNECT.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Receive (EHNRQRECV)

Purpose

Receives a block of data from the host partner program. It is only valid in an environment where an EHNRQACCEPT or EHNRQINVOKE has been issued. Once issued, this call must be repeated until an end of data return code is received indicating that the host is finished sending. At that time, EHNRQSEND or any of the SQL database access calls may be issued.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQRECV(int,            /* Data buffer size          */
                               char far *,     /* Data buffer address       */
                               int far *);     /* Data length               */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQRECV(     buflen : integer;
                        buffer : adsmem;
                  vars rcvlen : integer) : integer; extern;
```

Parameters

buflen Length of the buffer into which the received data is placed. The maximum amount of data that may be received at one time is limited by the value specified on the EHNRQSTART or EHNRQSTARTSEC call. See "Start (EHNRQSTART)" on page 15-45.

buffer Address of the receive buffer.

rcvlen Returns the actual length of the data received from the AS/400 program.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Retrieve Message (EHNRQRTVMSG)

Purpose

Retrieves the message text associated with the status of the last SQL operation performed. The message text is placed in the user's buffer and is null-terminated.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQRTVMSG(char far *,      /* Buffer address            */
                                 short far *);    /* Buffer length             */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQRTVMSG(     buffer : adsmen;
                      vars buf_len: integer) : integer; extern;
```

Parameters

User-supplied values:

buffer Points to the buffer where the message text is placed.

buf_len The size in bytes of the buffer that holds the message text.

Returned values:

buf_len If the size of the buffer is too small, this value is set to the minimum size needed in bytes to hold the complete message text.

If the size of the buffer is larger than the minimum size, this value is the size in bytes of the message.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Select (EHNRQSELECT)

Purpose

Sends an SQL SELECT statement to the host system to be prepared and run. Up to 10 select statements may be prepared or active at one time.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

typedef unsigned char hcursor;                  /* Handle for select cursor  *
/
extern int  extentry EHNRQSELECT(char far *,    /* SQL statement             */
                                 int,           /* Update flag               */
                                 hcursor far *); /* Cursor handle            */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;

function EHNRQSELECT(     stmnt : adsmem;
                        update : integer;
                      vars cur : hcursor)       : integer; extern;
```

Parameters

stmnt An ASCIIZ string defining the SQL SELECT statement to be run. The maximum length of the string is limited to the buffer size specified on the EHNRQSTART or EHNRQSTARTSEC call.

Note: On DBCS systems, this limit applies to the length of the statement **after** it has been translated to EBCDIC.

update If a nonzero value is specified, then selected rows are returned one at a time from the host system, and EHNRQUPCUR or EHNRQDELETE calls can optionally be used to update or delete values in selected rows. If a zero value is provided, the physical row data is returned from the host in blocks while selected rows are logically returned to the application one at a time. This provides improved performance, but precludes updates or deletions based on cursor position.

You can use the EHNRQSETROWS verb to specify the number of rows you want this verb to return. If you do not use EHNRQSETROWS, all rows are returned.

cur Returns a handle that must be supplied on succeeding calls which reference the SQL cursor associated with this statement. (For example, EHNRQGETF and EHNRQUPCUR.)

Note: A cursor is a logical pointer associated with the processing of a SELECT statement that is used to keep track of which selected rows have been returned. EHNRQFETCH and EHNRQGETF cause the cursor to be moved to the next available selected row, if any. EHNRQUPCUR and EHNRQDELETE act upon the most recently retrieved row as indicated by the current cursor position. See *SQL/400* Reference* for more information.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Prepare Select with Parameter Markers (EHNRQSELECTPM)

Purpose

Similar to the SELECT process, this processes statements that contain parameter markers. A parameter marker is a question mark (?) that appears in a statement string of dynamic SQL statements. The question mark can appear where a host variable could appear if the statement string was a static SQL statement.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0
#define extentry _saveregs pascal far

typedef unsigned char hcursor;                    /* Handle for select cursor

extern int  extentry EHNRQSELECTPM(char far *,    /* SQL statement
                                   hcursor far *); /* Cursor handle
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;

function EHNRQSELECTPM(stmnt     : adsmem;
                       vars cur  : hcursor) : integer; extern;
```

Parameters

stmt — An ASCIIZ string that contains the SQL statement (containing parameter markers) to be run. The maximum length of the string is limited to the buffer size specified on the EHNRQSTART or EHNRQSTARTSEC call.

Note: On DBCS systems, this limit applies to the length of the statement after it has been translated to EBCDIC.

cur — Returns a handle that must be supplied on succeeding EHNRQSELECTVAL calls which reference the parameter markers associated with this statement.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Execute Select with Parameter Markers (EHNRQSELECTVAL)

Purpose

Executes a prepared select statement with parameter markers.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0
#define extentry _saveregs pascal far

typedef unsigned char hcursor;                    /* Handle for select cursor */

extern int ententry EHNRQSELECTVAL(hcursor far *,  /* Cursor handle  */
                                   char far *,     /* Format string  */
                                   void far *,     /* Variable array */
                                   int      );     /* Update flag    */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;
  varray = [1..100] of adsmem;
  varptr = ads of varray;

function EHNRQSELECTVAL(cur       : hcursor;
                        fmt       : adsmem;
                        vap       : varptr;
                        update    : integer) : integer; extern;
```

Parameters

cur — The handle corresponding to a prepared select with parameter marker statement (that is returned by EHNRQSELECTPM).

fmt — An ASCIIZ string of format items that defines the data types to be used for each of the parameter markers specified in the EHNRQSELECTPM statement. The number of blank-delimited format items defines the number of elements required in the array of pointers passed as the third parameter.

One format item must be provided for each parameter marker that is specified in the EHNRQSELECTPM call. The allowable format items are:

- A — The corresponding variable is a far, ASCIIZ string. The corresponding SQL column must be a CHAR, VAR CHAR, or LONG VAR CHAR type.
- F — The corresponding variable is a FLOAT type. The corresponding SQL column may be any of the SQL numeric data types (SMALLINT, INTEGER, NUMERIC, DECIMAL, single-precision or double-precision FLOAT).
- D — The corresponding variable is a DOUBLE type. The corresponding SQL column may be any of the SQL numeric data types.
- G — The corresponding variable is a far, null-terminated string. The corresponding SQL data type must be GRAPHIC or VAR GRAPHIC.
- L — The corresponding variable is a LONG integer type. The corresponding SQL column may be any of the SQL numeric data types.

S — The corresponding variable is a SHORT integer type. The corresponding SQL column may be any of the SQL numeric data types.

Note: Floating point numbers are assumed to be in the IEEE format.

vap — An array of far pointers to variables of the type indicated by the format string items. The values of these variables is inserted into the SQL statement specified in the EHNRQSELECTPM in place of the parameter markers. If a value of NULL is to be used, a NULL pointer must appear in the array of pointers.

update — If a nonzero value is specified, then selected rows are returned one at a time from the host system, and EHNRQUPCUR or EHNRQDELETE calls can optionally be used to update or delete values in selected rows. If a zero value is provided, the physical row data is returned from the host in blocks while selected rows are logically returned to the application one at a time. This provides improved performance, but precludes updates or deletions based on cursor position.

You can use the EHNRQSETROWS verb to specify the number of rows you want this verb to return. If you do not use EHNRQSETROWS, all rows are returned.

Return Codes

If the return code is negative, then a format item that is not valid was found. The absolute value of the return code indicates which format item was being processed when the error was discovered.

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Send (EHNRQSEND)

Purpose

Sends a block of data to the partner program on the remote system. This call is only valid after an EHNRQACCEPT or an EHNRQINVOKE call has been issued (for example, when an AS/400 application is talking to the remote SQL server on the host side).

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQSEND(int,              /* Data length              */
                               char far *);      /* Data buffer address      */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQSEND(send_len : integer;
                   buffer   : adsmem) : integer; extern;
```

Parameters

send_len	Count of the number of bytes to send. The maximum value allowed is limited to the buffer size specified on the EHNRQSTART or EHNRQSTARTSEC call. See "Start (EHNRQSTART)" on page 15-45.
buffer	Address of the buffer containing the data to be sent.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Set Rows (EHNRQSETROWS)

Purpose

This verb allows you to specify the maximum number of rows sent in each block from the host system to the personal computer for each select (ENHRQSELECT or EHNRQSELECTVAL) and subsequent retrieve (EHNRQFETCH and EHNRQGETF) request only in modes other than update. Until this verb is run, EHNRQSELECT and EHNRQSELECTVAL return all rows in modes other than update.

This verb allows PC applications to interrupt with another API call after some rows have been sent to the host system.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQSETROWS(unsigned short);     /* Number of rows */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQSETROWS(  maxrows : word) : integer; extern;
```

Parameters

max_rows — Maximum number of rows to be blocked together when a SELECT is done in a mode other than update.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Retrieve SQLCA (EHNRQSQLCA)

Purpose

Obtains the SQLCA structure describing the ending conditions associated with the last SQL operation performed.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQSQLCA(struct sqlca far *,  /* SQLCA structure pointer
                                short far *);        /* SQLCA structure size
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQSQLCA(        sqlca : adsmem;
                    vars sqlcalen : integer) : integer; extern;
```

Parameters

sqlca — A far pointer to an area of sufficient size to receive the SQLCA structure.

sqlcalen — The size in bytes of the area pointed to by sqlca. The number of bytes should be greater than or equal to 136. If the size specified is too small, then no data is retrieved and the sqclalen parameter is modified to contain the minimum size required.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Start (EHNRQSTART)

Purpose

Performs the initialization necessary for communication with the RMTSQL server on the host system. EHNRQSTART, EHNRQACCEPT, or EHNRQSTARTSEC must be the first RMTSQL function called by an application.

This verb uses the PC Support router's user ID and password. This is the only difference between this verb and EHNRQSTARTSEC.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

extern int  extentry EHNRQSTART(char far *,          /* System name        */
                                short,               /* Commit flag        */
                                unsigned short);     /* Buffer size        */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQSTART(         sys : adsmem;
                    commit, buf : integer) : integer; extern;
```

Parameters

sys — An ASCIIZ string specifying the APPC Partner LU name (system name) to be used for the host system.

commit — A flag indicating whether COMMIT-type operations are supported. For a list of the START options, see "Remote SQL Start Options" on page 15-51.

buf — The size of the buffer to be used for data transfers to and from the host. This value represents the maximum length (after translation) of a SQL statement provided on a EHNRQEXEC, EHNRQEXECPM, EHNRQSELECT, or EHNRQSELECTPM call, or constructed as a result of an EHNRQUPCUR, EHNRQEXECVAL, or EHNRQSELECTVAL call.

Note: On DBCS systems, the amount of storage required may increase somewhat during ASCII-to-EBCDIC translation due to the insertion of shift-in (SI) and shift-out (SO) characters.

This value also represents the maximum amount of data that can be sent (after translation) by a single EHNRQSEND call or received by an EHNRQRECV call. The value provided is adjusted if necessary to force it to fall into the range from 1024 to 31 744 bytes. (For example, specifying 0 is valid, and results in the use of a 1024-byte buffer). This parameter does not limit the amount of data that can be received by the application on a single EHNRQFETCH or EHNRQGETF call.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Start Session (EHNRQSTARTSEC)

Purpose

Performs the initialization necessary for communication with the remote SQL server on the host system using the specified user ID and password. EHNRQSTARTSEC, EHNRQACCEPT, or EHNRQSTART must be the first RMTSQL function called by an application. The override of the user ID and password is for this conversation only and will revert back to the router sign-on values after this conversation ends.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0
#define extentry _saveregs pascal far

extern int ententry EHNRQSTARTSEC(char far *,     /* System name     */
                                  unsigned short, /* Commit flag     */
                                  short,          /* Buffer size     */
                                  char far *,     /* User ID         */
                                  char far *,     /* Password        */
                                  char far *);    /* Reserved        */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
function EHNRQSTARTSEC(sys       : adsmem;
                       commit, buff : integer;
                       userid    : adsmem;
                       password  : adsmem;
                       reserved  : adsmem) : integer; extern;
```

Parameters

sys — An ASCIIZ string specifying the APPC Partner LU name (system name) to be used for the host system.

commit — A flag indicating whether COMMIT-type operations are supported. For a list of the START options, see "Remote SQL Start Options" on page 15-51.

buffer — The size of the buffer to be used for data transfers to and from the host. This value represents the maximum length (after translation) of a SQL statement provided on an EHNRQEXEC, EHNRQEXECPM, EHNRQSELECT or EHNRQSELCTPM call, or constructed as a result of an EHNRQUPCUR, EHNRQEXECVAL, or EHNRQSELECTVAL call.

userid — The user ID on the Partner LU to be used for this session. This user ID will determine the authority levels for working with the databases.

password — The password associated with the user ID above.

reserved — This field is reserved.

Notes:

1. Sys, userid, password, and reserved must all be null- terminated.
2. On DBCS systems, the amount of storage required may increase somewhat during the ASCII-to-EBCDIC translation due to the insertion of SI/SO characters. It also represents the maximum amount of data that can be sent (after translation) by a single EHNRQSEND call or received by an EHNRQRECV call. The value provided is adjusted if necessary to force it to

fall into the range from 1024 to 31 744. (For example, specifying 0 is valid, and results in the use of a 1024-byte buffer). This parameter does not limit the amount of data that can be received by the application on a single EHNRQFETCH or EHNRQGETF call.

3. On DBCS systems, user ID and password must be specified as single-byte characters.

Return Codes

For return codes, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Update Current Row (EHNRQUPCUR)

Purpose

Performs an update of one or more columns in the row at the current cursor position. EHNRQUPCUR must be preceded by a successful EHNRQFETCH or EHNRQGETF call, and the most recent EHNRQSELECT or EHNRQSELECTVAL call that returned the cursor handle being used must have specified a nonzero update flag. Only columns that are being retrieved by the active SELECT statement can be updated by this call.

Note: Some SELECT statement options preclude updates based on cursor position. For example, ORDER BY, FOR UPDATE OF, and GROUP BY can all affect whether updates may be done. See the topic Read Only Tables in the *SQL/400* Reference*.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNRQAPI.H>
```

Declarations provided in include files:

```
#define RQ_OK 0

#define extentry _saveregs pascal far

typedef unsigned char hcursor;              /* Handle for select cursor      */

extern int extentry EHNRQUPCUR(hcursor,     /* Cursor handle                 */
                               char far *,  /* Format string                 */
                               void far *); /* Variables array pointer       */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNRQAPI.INC'}
```

Declarations provided in include files:

```
TYPE
  hcursor = byte;
  varray = array[1..100] of adsmem;
  vaptr = ads of varray;

function EHNRQUPCUR(     cur : hcursor;
                         fmt : adsmem;
                         vap : vaptr)  : integer; extern;
```

Parameters

cur — The cursor handle corresponding to an active SELECT cursor (that is, returned by EHNRQSELECT or EHNRQSELECTPM and the cursor is still open).

fmt — An ASCIIZ string of format items that define the columns to be updated and the data types of the corresponding arguments. The number of blank-delimited format items defines the number of elements required in the array of pointers passed as the third parameter. The operation ends when either all of the format items have been processed, or all of the selected columns have been processed.

One format item must be provided for each column being returned by the active SELECT statement - at least up to and including the last column being updated. For example, if 6 columns are being returned, but only the first and third are being updated, the format list might look like "A * D." The allowable format items are:

- **A** — The corresponding variable is a far, ASCIIZ string. The corresponding SQL column must be a CHAR, VAR CHAR, or LONG VAR CHAR type. If the update string is longer than the SQL column, the update operation is suppressed.
- **F** — The corresponding variable is a FLOAT type. The corresponding SQL column may be any of the SQL numeric data types (SMALLINT, INTEGER, NUMERIC, DECIMAL, single-precision or double-precision FLOAT).
- **D** — The corresponding variable is a DOUBLE type. The corresponding SQL data type may be any of the SQL numeric data types.
- **G** — The corresponding variable is a far, null-terminated string. The corresponding SQL data type must be GRAPHIC or VAR GRAPHIC.
- **L** — The corresponding variable is a LONG integer type. The corresponding SQL data type may be any of the SQL numeric data types.
- **S** — The corresponding variable is a SHORT integer type. The corresponding SQL data type may be any of the SQL numeric data types.
- ***** — No corresponding variable is required, and one must not be provided. This format item serves as a placeholder to skip over a column that is not being updated.

Notes:

1. Floating point numbers are assumed to be in the IEEE format.
2. Loss of precision may result if the precision or scale of the update value is greater than that of the SQL column.

vap An array of far pointers to variables of the type indicated by the corresponding format string items. The values of these variables are used to update the corresponding selected columns of the current table row. There must be an entry in the array for each item in the format list, excluding asterisks (*). If a value of NULL is to be used, a NULL pointer must appear in the array of pointers.

Additional Information

Performance may be improved by providing values for all the columns specified in the SELECT statement. This applies even when all the columns are not to be updated.

Return Codes

If the return code is negative, then a format item that is not valid was found. The absolute value of the return code indicates which format item was being processed when the error was discovered.

If the return code is zero or positive, see "Return Codes for Remote SQL Function High-Level API" on page 15-53.

Sample PC Program

The following code is a simple example of a portion of a PC application program. The sample is incomplete, but illustrates one way to use the PC Support API entry points.

```
include <string.h>
include <EHNRQAPI.H>
void QRYSELLS(person, product, sales, quota)
char *person;
char *product;
long *sales;       /* Address of callers sales variable           */
long *quota;       /* Address of callers quota variable           */
{
 int rc;
 hcursor cursor;                        /* Cursor handle          */
 int Commit = 0;                        /* No commit/rollback     */
 int Bufsize = 0;                       /* Default buffer size    */
 char sqlstmnt[100]           /* SQL statement          */
 char *Rmtsystem = "RMTSYS";            /* Remote system name     */
 struct sqlca *cap;                     /* -> SQLCA control block */
 void far *Dest_Vars[2];

 rc=EHNRQSTART(Rmtsystem,Commit,Bufsize);   /* Open communications */

 /* Build the SQL select statement to retrieve                     */
 /* old sales and quota data                                       */
 strcpy(sqlstmnt,"Select sales,quota from sampledb/sells where person='");
 strcat(sqlstmnt,person);
 strcat(sqlstmnt,"' and product='");
 strcat(sqlstmnt,product);
 strcat(sqlstmnt,"'");
 Dest_Vars[0] = sales;     /* Initialize destination  */
```

```
Dest_Vars[1] = quota;     /*   variable pointers     */

rc=EHNRQSELECT(sqlstmnt,0,&cursor);    /* Prepare SELECT          */
if (rc==RQ_OK)                         /* Select processed OK?    */
 {                                     /* Yes                     */
  rc=EHNRQGETF(cursor,"L L",Dest_Vars);  /* Get a row of data     */
  if (rc==RQ_OK)                       /*Row retrieved w/o errors?*/
   {                                   /* Yes... Process the data */
    ...
   }
  else
    if (rc == RQ_FETCH_ERR)            /* Was error on SQL fetch? */
     {                                 /* Yes, report the error   */
      cap = NULL;
      cap = EHNRQSQLCA(cap);           /* Retrieved assoc SQLCA   */
      printf("SQLCODE: %ld\nSQLERRM: %.*s\n",cap->sqlcode,
                                            cap->sqlerrml,
                                            cap->sqlerrmc);
      free(cap);
     }
    else                               /* Some other kind of error*/
      printf("Error occurred on GET, rc=%d\n",rc);
  EHNRQCLOSE(cursor);                  /* Close the cursor        */
 }
else                                   /* Error occurred on select*/
  if (rc == RQ_COMM_ERROR)             /* Was this a comm. error? */
   {                                   /* Yes, report it          */
    EHNRQERROR(prime,second);          /* Retrieve maj/min codes  */
    printf("APPC Communications error:\n");
    printf("   primary rc   = %.4s\n   secondary rc = %.8s\n",
                                          prime,second);
   }
} /* QRYSELLS */
```

Remote SQL Start Options

Table 15-1. START Options

Value	Naming Convention	Commit Level
0	AS/400 (library_name/file_name)	*NONE
1	AS/400 (library_name/file_name)	*ALL
2	AS/400 (library_name/file_name)	*CHG
3	AS/400 (library_name/file_name)	*CS
256	SQL (collection_name.table_name)	*NONE
257	SQL (collection_name.table_name)	*ALL
258	SQL (collection_name.table_name)	*CHG
259	SQL (collection_name.table_name)	*CS

The commit_mode flag of the EHNRQSTART and EHNRQSTARTSEC functions control which mode the server operates in, and also the naming conventions to follow. The following describes the commit levels:

*ALL — Specifies that all rows selected, updated, deleted, and inserted are locked until the commit cycle is committed or rolled back.

*CS — This is similar to *ALL, except that rows that are accessed but not updated, deleted, or inserted are unlocked as soon as a different row is accessed. This is known as *cursor stability* locking.

*CHG — Specifies that only updated, deleted, and inserted rows are locked until the commit cycle is committed or rolled back.

*NONE Specifies that commitment control is not used. If DLL statements are included in the program, *NONE must be used.

Remote SQL Parser Options

Table 15-2. Naming Convention Remote SQL Parser Options

Constant Name	Decimal Value	SQL Parser Value
RQ_NAME_SYS	(0)	*SYS
RQ_NAME_SQL	(1)	*SQL

Table 15-3. Commitment Control Remote SQL Parser Options

Constant Name	Decimal Value	SQL Parser Value
RQ_COMMIT_NONE	(0)	*NONE
RQ_COMMIT_ALL	(1)	*ALL
RQ_COMMIT_CHG	(2)	*CHG
RQ_COMMIT_CS	(3)	*CS

Table 15-4. Date Format Remote SQL Parser Options

Constant Name	Decimal Value	SQL Parser Value
RQ_DATEFMT_USA	(0)	*USA
RQ_DATEFMT_ISO	(1)	*ISO
RQ_DATEFMT_EUR	(2)	*EUR
RQ_DATEFMT_JIS	(3)	*JIS
RQ_DATEFMT_MDY	(4)	*MDY
RQ_DATEFMT_DMY	(5)	*DMY
RQ_DATEFMT_YMD	(6)	*YMD
RQ_DATEFMT_JUL	(7)	*JUL

Table 15-5. Date Separator Remote SQL Parser Options

Constant Name	Decimal Value	SQL Parser Value
RQ_DATESEP_PERIOD	(0)	. (period)
RQ_DATESEP_COMMA	(1)	, (comma)
RQ_DATESEP_BLANK	(2)	(blank)
RQ_DATESEP_SLASH	(3)	/ (forward slash)
RQ_DATESEP_DASH	(4)	– (dash)

Table 15-6. Time Format Remote SQL Parser Options

Constant Name	Decimal Value	SQL Parser Value
RQ_TIMEFMT_USA	(0)	*USA
RQ_TIMEFMT_ISO	(1)	*ISO
RQ_TIMEFMT_EUR	(2)	*EUR
RQ_DATESEP_JIS	(3)	*JIS
RQ_DATESEP_HMS	(4)	*HMS

Table 15-7. Time Separator Remote SQL Parser Options

Constant Name	Decimal Value	SQL Parser Value
RQ_TIMESEP_PERIOD	(0)	. (period)
RQ_TIMEFMT_COMMA	(1)	, (comma)
RQ_TIMEFMT_BLANK	(2)	(blank)
RQ_DATESEP_COLON	(3)	: (colon)

Table 15-8. Decimal Point Remote SQL Parser Options

Constant Name	Decimal Value	SQL Parser Value
RQ_DECIMAL_PERIOD	(0)	. (period)
RQ_DECIMAL_COMMA	(1)	, (comma)

Return Codes for Remote SQL Function High-Level API

Functions in the Remote SQL DLL use the following return codes. These are constants defined in EHNRQAPI.H.

Table 15-9 (Page 1 of 3). Return Codes for the Remote SQL Function

Return Code	Hex Value	Description
RQ_OK	X'0000'	Operating completed successfully.
RQ_MSG_NOTF	X'0003'	No message text available to retrieve.
RQ_PREP_ERROR	X'0004'	An error occurred while preparing the statement. Use EHNRQSQLCA to obtain more information.
RQ_NON_SELECT	X'0006'	An attempt was made to run a SELECT statement with EHNRQEXEC, or the statement specified on the EHNRQSELECT or EHNRQSELECTPM entry point is not a valid select statement.
RQ_NON_CONNECT	X'0006'	Invalid connect statement.
RQ_LONG_STMNT	X'0007'	The SQL statement provided is longer than the buffer size specified or defaulted on the EHNRQACCEPT, EHNRQSTART, or EHNRQSTARTSEC call. The statement was not sent to the host for processing.
RQ_EXEC_ERROR	X'0008'	An error occurred while executing the statement. Use EHNRQSQLCA to obtain more information.
RQ_NVLD_OPTION	X'0009'	At least one of the options that you tried to change had an invalid value.

Table 15-9 (Page 2 of 3). Return Codes for the Remote SQL Function

Return Code	Hex Value	Description
RQ_NO_FETCH	X'000A'	A successful EHNRQFETCH or EHNRQGETF was not the last operation run for the specified cursor.
RQ_NOT_UPDATE	X'000B'	The cursor specified was not in update mode. Update mode is required for EHNRQDELETE or EHNRQUFCUR to be used.
RQ_FETCH_ERR	X'000C'	An error occurred on the FETCH. No result data was returned. Use EHNRQSQLCA to obtain more information about the error.
RQ_NOT_ACT	X'000D'	Cursor handle does not correspond to an active cursor.
RQ_OPEN_ERROR	X'000E'	An error occurred opening the cursor.
RQ_LASTROW	X'000F'	No more rows are available that satisfy the selection criteria. No result data was returned.
RQ_MAX_SEL	X'0010'	The maximum number of handles (10) are already active. Each active SELECT statement and each active statement that uses a parameter marker has an associated handle.
RQ_NVLD_STATE	X'0011'	Operation not valid now. An unrelated SELECT statement is active and is not in update mode, or the function is not valid because of a previous SQL statement that has been performed.
RQ_XlateBufSz	X'0013'	Translation buffer size too small.
RQ_ALRDY_INIT	X'0014'	Initialization has already been done through EHNRQSTART, EHNRQSTARTSEC, or EHNRQACCEPT.
RQ_NOT_INIT	X'0014'	Remote SQL is not active.
RQ_NVLD_COMMIT	X'0015'	The commit parameter is not 0, 1, 2, 3, 256, 257, 258, or 259.
RQ_RMT_ACTIVE	X'0016'	Communication with a host application is already active through either EHNRQACCEPT or EHNRQINVOKE.
RQ_NO_RMT_PGM	X'0017'	Send is not valid because there is no active partner application program on the host system.
RQ_RMT_TERM	X'0018'	The partner program which this program initiated through an EHNRQINVOKE has ended. Database operations are still valid, but SEND and RECV are not.
RQ_FORCE_TERM	X'0019'	The partner program that initiated this program has ended communications. An EHNRQEND should be issued.
RQ_NVLD_RW	X'001A'	SEND/RECV exchange pending.
RQ_NVLD_LEN	X'001B'	The buffer provided was too small. No data was returned. Allocate a buffer at least the size of the data returned.
RQ_DATA_TRUNC	X'001C'	Received data truncated. The buffer provided was not large enough to contain all of the data sent.
RQ_EOD	X'001D'	End of data. The partner program has stopped sending. Database operations and EHNRQSEND are now valid.
RQ_INVOC_FAIL	X'001E'	The host program requested on the previous EHNRQINVOKE could not be called. The most likely cause is that the library or program name provided is not correct.
RQ_COMM_ERROR	X'0020'	Either there was no pending connection request from a host system, or an APPC error was detected while establishing the connection. EHNRQERROR may be used to obtain the APPC error return codes. An EHNRQEND should be done.

Table 15-9 (Page 3 of 3). Return Codes for the Remote SQL Function

Return Code	Hex Value	Description
RQ_LEVEL	X'0024'	The host server program and the local RMTSQL API version are at incompatible release levels.
RQ_XI_UNAVAIL	X'0026'	PCSXI program has not been started (DOS only).
RQ_LOAD_FAIL	X'0027'	Unable to load EHNRQAPI.OVL into protected memory (DOS only).
RQ_CIM_UNAVAIL	X'0028'	The PC Support/400 router is not available. Processing cannot continue.
RQ_DBCSXlateErr	X'002A'	DBCS translation error. 8580 Sys1IBM.table not found (DBCS only).
RQ_XLATEOVERFL	X'002C'	Translation output buffer overflow (DBCS only). This is most likely due to an input string expanding during translation from ASCII to EBCDIC to a length greater than the buffer size specified on the EHNRQACCEPT, EHNRQSTART, or EHNRQSTARTSEC call.
RQ_XLT_SETUP	X'002E'	Error initializing for DBCS translation. Unable to obtain ASCII DBCS range vector. PCSXLT may need to be loaded.
RQ_NO_MEMORY	X'0032'	Insufficient memory to continue.
RQ_MEMORY	X'0032'	Remote SQL has been ended. An error occurred when attempting to free memory (DOS only).
RQ_NO_USERID	X'0040'	No user ID specified, but a password was specified.
RQ_USERID_INVLGTH	X'0041'	User ID exceeds maximum length of 10 characters.
RQ_PWORD_INVLENGTH	X'0042'	Password exceeds maximum length of 10 characters.
RQ_INV_SECURITY	X'0043'	User ID and password combination rejected by the host system.
RQ_INV_CHAR	X'0044'	User ID or password contains invalid characters.
RQ_PACKAGE_ERROR	X'0050'	Package error. The package does not exist, is locked, or is damaged.
RQ_NVLD_PACKAGE	X'0051'	This is not a valid parameter for the Create Package API.
RQ_NO_STATEMENT	X'0052'	No statement is specified. The number of the statement is not greater than zero.
RQ_NVLD_NAME	X'0053'	The library or package name was not specified correctly.
RQ_RMT_UPDT_NVLD	X'0054'	Cannot open a cursor for the stored SELECT statement while accessing a remote database.
RQ_NVLD_STMT	X'0055'	Not a valid SQL statement.
RQ_NVLD_STMTNUM	X'0056'	Not a valid SQL statement number.
RQ_CRT_RMTPKG_ERR	X'0057'	Create package is set to 'Y' or 'A' while connected to a remote relational database.

AS/400 Remote SQL Function API

The functions described in this section are available to an AS/400 application communicating with a PC application. The remote SQL interface allows PC applications to invoke host applications and allows host programs to invoke PC applications running in an Operating Systems/2 (OS/2) environment. There is no requirement for a program written by a user to exist on the host system in order to use remote SQL. However, the API does provide a means by which PC and AS/400 host programs can communicate and exchange information.

PC-Initiated Commands

In both DOS and OS/2, a personal computer can invoke an AS/400 host program. The following entry points can be used.

QRQRCVDT
QRQSNDDT
QRQRTVER

Host-Initiated Communication

An AS/400 host application may establish communication with a PC running under the OS/2 operating systems. Before an AS/400 host program can invoke a PC program, some setup is required on both the host system and the personal computer.

PC Setup

The following setup is required on your personal computer.

- Communications manager must be active.
- PC Support/400 router must be started to the host system.
- Attachable transaction program (TP) profiles need to be configured with the following characteristics:
  - Specify *NO* for Service TP.
  - Specify *NONE* for Sync Level.
  - Specify *MAPPED* for Conversation Type.
  - Specify *YES* or *NO* for Conversation Security. If *NO* is specified, the ID and password from the host system are not checked.
  - Specify *NONQUEUED* for TP Operation.
- Attach manager must be started.
- In your PC program, the transaction program name must be specified in the EHNRQ_ACCEPT call.

For additional information, see the *Communications: Management Guide*.

Host Setup

The following setup is required on the AS/400 host system:

- QIWS must be in the current library list.
- An ICF device entry must exist for the target personal computer. This entry can be in QRQICF in library QIWS or it can be in a user ICF file. To create a user ICF file, issue the following command:

```
CRTDUPOBJ  OBJ(QRQICF)  FROMLIB(QIWS)  OBJTYPE(*FILE)
  TOLIB(your library)  NEWOBJ(your icf)  DATA(*NO)
```

  To add the device entry to QIWS/QRQICF or to a user ICF, issue the Add ICF Device Entry (ADDICFDEVE) command. Following is an example of adding a device entry to QRQICF in QIWS:

```
ADDICFDEVE  FILE(QIWS/QRQICF)  PGMDEV(device name)
  RMTLOCNAME(remote name)  CMNTYPE(*APPC)  MODE(QPCSUPP)
```

- When using ICF files other than QIWS/QRQICF, the following command must be issued before QRQSTRRP is attempted.

```
OVRICFF  FILE(QRQICF)  TOFILE(your library/your icf)
```

- The host application must use the device entry name specified in the ICF file. It must also use the transaction program name specified on the personal computer.

  Note: More information is available in the *Communications: Intersystem Communications Function Programmer's Guide*.

  The following entry points can be used:
  - QRQSTRRP
  - QRQSNDDT
  - QRQRCVDT
  - QRQRTVER
  - QRQENDRP

Environment Names

The environment name that must be specified on the QPXXCALL varies according to how the AS/400 programs were called. If they were called through a request from a personal computer, then the environment name must be EHNRSQLX. If the AS/400 application is the initiator, then the environment name can be anything, although the same name must be used throughout a session. That is, the name used on the initial call to the server API (usually QRQSTRRP) should be used on all succeeding calls.

QPXXCALL Environment Cleanup

Normally, if an AS/400 application is establishing an environment through a call to QRQSTRRP (start a remote program), when all communications with the remote PC have been completed (the QRQENDRP call has been done), the environment created by the initial call should be deleted through a call to QPXXDLTE. A PL/I call to QPXXDLTE would look like:

```
call QPXXDLTE(-1,envid);
```

Where:

- -1 specifies that the environment is being deleted
- envid specifies the name of the environment to be deleted (the name that you used earlier on the QPXXCALL --> QRQSTRRP call).

End Remote Program (QRQENDRP)

Purpose

Ends communication with a remote PC application. Issued to end an AS/400 to PC communication session initiated by the AS/400 application.

Procedure Declaration

PL/I

```
dcl qpxxcall entry (char(100), char(10), char(20), *,*,*,*,*)
             options(asm);
dcl entryname char(100) static,
   envid     char(10)  static init('EHNRSQLX'), /* Environment name    */
   progname  char(20)  static init('QRQSRVRX'), /* Program name -      */
                                                /*  must be QRQSRVRX   */
    rc        fixed bin(31) static;             /* Server return code  */

entryname = 'QRQENDRP';                         /* Set entrypoint name */
call qpxxcall(entryname,envid,progname,rc);
```

RPG

```
     **      Data structure for return code
I            DS
I                                     B   1   40RC
 **      Declaration and initialization of variables
C                     MOVE *BLANKS    ENTRY 100
C                     MOVE *BLANKS    ID     10
C                     MOVE *BLANKS    PGMNAM 20
C                     MOVEL'QRQSRVRX'PGMNAM
C                     MOVEL'CENVIRON'ID
C                     MOVEL'QRQENDRP'ENTRY
C                     CALL 'QPXXCALL'
C                     PARM            ENTRY
C                     PARM            ID
C                     PARM            PGMNAM
C                     PARM            RC
```

Parameters

None

Return Codes

For return codes, see "AS/400 System Return Codes for the Remote SQL Function" on page 15-64.

Retrieve Error Data (QRQRTVER)

Purpose

Returns a 4-character EBCDIC string representing the APPC major and minor error return codes associated with the last communication attempt with the personal computer. Normally, QRQRTVER would be issued either to obtain the return codes for logging or for use in an error message created by an application when a communications error occurs (a return code of X'20' from a previous QRQxxxx call).

Procedure Declaration

PL/I

```
dcl qpxxcall entry (char(100), char(10), char(20), *,*,*,*,*)
             options(asm);
dcl entryname char(100) static,
    envid     char(10)  static init('EHNRSQLX'), /* Environment name  */
    progname  char(20)  static init('QRQSRVRX'), /* Program name -    */
                                                 /*  must be QRQSRVRX */
    majminor  char(4)   static;                  /* Return codes      */
entryname = 'QRQRTVER';
call qpxxcall(entryname,envid,progname,majminor);
```

RPG

```
C                   MOVE *BLANKS   ENTRY 100
C                   MOVE *BLANKS   ID     10
C                   MOVE *BLANKS   PGMNAM 20
C                   MOVE *BLANKS   MAJMIN  4

C                   MOVEL'QRQSRVRX'PGMNAM
C                   MOVEL'CENVIRON'ID
C                   MOVEL'QRQRTVER'ENTRY
C                   CALL 'QPXXCALL'
C                   PARM           ENTRY
C                   PARM           ID
C                   PARM           PGMNAM
C                   PARM           MAJMIN
```

Parameters

majminor — Returns a 4-character EBCDIC string representing the APPC major and minor error return codes.

Start Remote Program (QRQSTRRP)

Purpose

Establishes a connection with the specified personal computer and initiates a transaction program. This call is only valid for a personal computer using the OS/2 operating system.

Procedure Declaration

PL/I

```
dcl qpxxcall entry (char(100), char(10), char(20), *,*,*,*,*)
            options(asm);
dcl qpxxdlte entry (fixed bin(31,0), char(10)) options(asm);
dcl entryname char(100) static,
    envid     char(10)  static,                 /* Environment name  */
    progname  char(20)  static init('QRQSRVRX'), /* Program name -    */
                                                 /*  must be QRQSRVRX */

    devent    char(10)  static,     /* ICF file device entry for PC   */
    tpname    char(64)  static,     /* ID of program to run on PC     */
    userid    char(10)  static,
    password  char(10)  static,
    rc        fixed bin(31) static; /* Server return code             */
 entryname = 'QRQSTRRP';
 envid = 'CENVIRON';     /* Environment name specified by user        */
 tpname = 'EXPERT';      /* TP identifier                             */
 devent = 'PS295';       /* Name of ICF device entry to use           */
                         /* PS295 is only an example                  */
 userid = ' ';           /* Assume not ID or password protected       */
 password = ' ';
 call qpxxcall(entryname,envid,progname,rc,devent,tpname,
              userid,password);
   .
   .
   .
 call qpxxdlte(-1,envid);          /* Environment complete            */
```

RPG

```
 **      Data structure for return code
I           DS
I                                    B   1   40RC
 **      Declaration and initialization of variables
C                     MOVE *BLANKS   ENTRY 100
C                     MOVE *BLANKS   ID     10
C                     MOVE *BLANKS   PGMNAM 20
C                     MOVE *BLANKS   DEVENT 10
C                     MOVE *BLANKS   TPNAME 64
C                     MOVE *BLANKS   USERID 10
C                     MOVE *BLANKS   PASWRD 10

C                     MOVEL'QRQSRVRX'PGMNAM
C                     MOVEL'CENVIRON'ID
C                     MOVEL'PS295'   DEVENT
C                     MOVEL'EXPERT'  TPNAME
C                     MOVEL'QRQSTRRP'ENTRY
C                     CALL 'QPXXCALL'
C                     PARM           ENTRY
C                     PARM           ID
C                     PARM           PGMNAM
C                     PARM           RC
C                     PARM           DEVENT
C                     PARM           TPNAME
C                     PARM           USERID
C                     PARM           PASWRD
```

Parameters

Note: All of these character parameters must be left-justified and padded to their respective lengths with blanks.

devent — The name of the device entry in the QRQICF ICF file that corresponds to the personal computer where the application resides. This device entry is typically created by the following command:

```
QIWS/QRQICF PGMDEV (dev_name)
                   RMTLOCNAME(rmt_name)
                   CMNTYPE(*APPC) MODE(QPCSUPP)
```

dev_name — The name you want to assign to the device.

rmt_name — The value of the remote control point name field of the controller for the personal computer. This is also the value for LU name field in the OS/2 local APPC logical unit profile.

tpname — The identifier of the PC application to be run. This name is typically not the actual name of the PC program, but an identifier that matches an OS/2 configuration file entry on the personal computer.

userid — An optional 10-character user ID. If provided, it and the password are sent to the remote personal computer for verification. It is normally only required when the remote personal computer enforces conversation security. If not provided (blank), no user ID or password is sent.

password — Optional password. Required if a non-blank user ID is specified.

Return Codes

For return codes, see "AS/400 System Return Codes for the Remote SQL Function" on page 15-64.

Receive Data (QRQRCVDT)

Purpose

Receives a block of data from the PC application program. Once a QRQRCVDT has been issued, no further QRQSNDDT calls may be done until an End of Data return code is received.

While the QRQRCVDT call is outstanding, the PC application may perform SQL operations if needed to create a response for the AS/400 application. These requests are handled transparently by the server without affecting the status of the QRQRCVDT call.

Procedure Declaration

PL/I

```
dcl qpxxcall entry (char(100), char(10), char(20), *,*,*,*,*)
           options(asm);
dcl entryname char(100) static,
    envid     char(10)  static init('EHNRSQLX'), /* Environment name    */
    progname  char(20)  static init('QRQSRVRX'), /* Program name -      */
                                                 /*  must be QRQSRVRX   */

    recv_buf_size  fixed(31) static,             /* Buffer size   */
    rc        fixed bin(31) static;              /* Server return code  */

recv_buf_size = 20;            /* Buffer size (specified by the user)  */
entryname = 'QRQRCVDT';        /* Set entrypoint name                  */
call qpxxcall(entryname,envid,progname,
                          rc,recv_buf_size,recv_buf,recv_actual);
```

RPG

```
 **      Data structure for return code
I           DS
I                                    B   1   40RC
I           DS
I                                    B   1   40BUFLEN
I           DS
I                                    B   1   40LENGTH

C                     MOVE *BLANKS   ENTRY 100
C                     MOVE *BLANKS   ID     10
C                     MOVE *BLANKS   PGMNAM 20
C                     MOVE *BLANKS   BUFFER 80

C                     Z-ADD80        BUFLEN
C                     MOVEL'QRQSRVRX'PGMNAM
C                     MOVEL'CENVIRON'ID
C                     MOVEL'QRQRCVDT'ENTRY
C**
C                     CALL 'QPXXCALL'
C                     PARM           ENTRY
C                     PARM           ID
C                     PARM           PGMNAM
C                     PARM           RC
C                     PARM           BUFLEN
C                     PARM           BUFFER
C                     PARM           LENGTH
```

Parameters

recv_buf_size	Length of the buffer into which the received data is placed. The maximum length that can be received is limited to the value specified by the PC application on its EHNRQSTART or EHNRQACCEPT call.
buffer	Address of the receive buffer.
recv_actual	Returns the actual length of the data received from the AS/400 program.

Return Codes

For return codes, see "AS/400 System Return Codes for the Remote SQL Function" on page 15-64.

Send Data (QRQSNDDT)

Purpose

Sends a block of data to the PC partner program.

Procedure Declaration

PL/I

```
dcl qpxxcall entry (char(100), char(10), char(20), *,*,*,*,*)
            options(asm);
dcl entryname char(100) static,
    envid     char(10)  static init('EHNRSQLX'), /* Environment name          */
    progname  char(20)  static init('QRQSRVRX'), /* Program name -            */
                                                 /*  must be QRQSRVRX         */
    length    fixed(31) static,                  /* Length of data            */
    rc        fixed bin(31) static;              /* Server return code        */
length = 17;                                /* Length of data specified by user  */
entryname = 'QRQSNDDT';
call qpxxcall(entryname,envid,progname,
                      rc,length,buffer);
```

RPG

```
 **      Data structure for return code
I           DS
I                                    B   1   40RC
I           DS
I                                    B   1   40LENGTH

C                     MOVE *BLANKS   ENTRY 100
C                     MOVE *BLANKS   ID     10
C                     MOVE *BLANKS   PGMNAM 20
C                     MOVE *BLANKS   BUFFER 80

C                     MOVEL'QRQSRVRX'PGMNAM
C                     MOVEL'CENVIRON'ID
C                     MOVEL'QRQSNDDT'ENTRY
C                     MOVE 'YourData'BUFFER
C**
C                     Z-ADD8         LENGTH
C                     CALL 'QPXXCALL'
C                     PARM           ENTRY
C                     PARM           ID
C                     PARM           PGMNAM
C                     PARM           RC
C                     PARM           LENGTH
C                     PARM           BUFFER
```

Parameters

length — Length of the data to be sent. The maximum amount of data that can be sent is limited to the value specified by the PC application on its EHNRQSTART or EHNRQACCEPT call.

buffer — Buffer containing the data to be sent.

rc — Returns a 4-byte binary return code.

Return Codes

For return codes, see "AS/400 System Return Codes for the Remote SQL Function" on page 15-64.

Message Descriptions

The following messages can be issued by the host server code.

IWS7101 Abnormal end. Data received was not valid. &&&&&&&&

IWS7102 Communications error encountered by QRQSRVRX.

IWS7103 ACQUIRE of *REQUESTER device failed.

IWS7104 Unable to open ICF file, &&&&&&&&&&&&&&&&&&&&

AS/400 System Return Codes for the Remote SQL Function

Table 15-10. AS/400 System Return Codes for the Remote SQL Function

Hex Value	Description
X'0000'	Successful completion.
X'0002'	Unknown request from the PC application. No further communications are possible. This condition should not occur and most likely is the result of a PC program attempting to use the custom code to access the remote SQL server.
X'0004'	The PC application has ended.
X'0005'	End of data. The personal computer has no more data to send.
X'000C'	Request rejected. Communications with a remote PC application have already been started.
X'000E'	Request rejected. There is no active connection to a remote personal computer.
X'0010'	Request rejected. The connection was initiated by the personal computer and cannot be ended by the AS/400 application.
X'0012'	Data truncation. The receive buffer was too small to contain all of the data sent by the PC application program.
X'0018'	Forced termination. The PC application that initiated this program has ended communications.
X'0019'	The PC application that caused this AS/400 program to be initiated has ended communications. The AS/400 program should perform any appropriate housekeeping and then end.
X'0020'	A communications error occurred, and the link to the target system has been lost. QRQRTVER may be used to obtain the APPC error return codes. A QRQENDRP call should then be done if this is not a PC-initiated program.
X'0022'	Unknown request from the PC application. No further communications can occur.

References

Following is a list of manuals that could provide additional information about the remote SQL function of PC Support/400.

SQL References

Systems Application Architecture Structured Query Language/400 Reference*, SC41-9608
Systems Application Architecture Structured Query Language/400 Programmer's Guide*, SC41-9609

DRDA References

Distributed Relational Database Guide, SC41-0025 (contains a complete bibliography of IBM manuals containing information on DRDA support)

AS/400 Programming

Languages: Systems Application Architecture AD/Cycle* COBOL/400* Reference Summary*, SX09-1209
Application Development Tools: Programming Development Manager User's Guide and Reference, SC09-1339
Languages: Systems Application Architecture AD/Cycle* RPG/400* Reference*, SC09-1349
Languages: Systems Application Architecture AD/Cycle* RPG/400* User's Guide*, SC09-1348

Chapter 16. Data Queues Function High-Level Application Program Interface

This chapter describes the high-level language API for the PC Support data queues function. These API routines may be used in an Extended DOS or OS/2 environment.

The API supports any programming language supported by the operating system.

Sample statements and procedure definitions for C and Pascal are provided in this chapter. An assembler language program could call the functions by sequence, so there are no such definitions given for assembler language.

Sample programs, include files, and other information for using APIs are provided in the PC Support tools folder (QIWSTOOL). See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

Data Queues Overview

A **data queue** is an AS/400 object that is used by AS/400 application programs for communications. Applications can use data queues to pass data between jobs. Multiple AS/400 jobs can send or receive data from a single data queue.

Manipulation of data queues is done using AS/400 CL commands and callable programming interfaces. Thus access to data queues is available to all AS/400 applications regardless of what language that application is written in. The following can be used to work with data queues:

- CRTDTAQ - Creates a data queue and stores it in a specified library
- DLTDTAQ - Deletes the specified data queue from the system
- CALL QSNDDTAQ - Send a message (record) to the specified data queue
- CALL QRCVDTAQ - Read a message (record) from the specified data queue
- CALL QCLRDTAQ - Clear all messages from the specified data queue
- CALL QMHQRDQD - Retrieve a data queue description

With the API for the PC Support data queues function, a PC application can work with AS/400 data queues with the same ease that AS/400 applications can. This extends AS/400 communications to include processes running on a remote PC.

DOS Environment

In the OS/2 operating system, another application (DQSA) is started as a result of the first API call that is made. In DOS, this other application (LOADDQ) must be loaded by the user before using any of the data queues APIs. For LOADDQ, this program runs automatically if data queues is chosen to run at the time of installation. The purpose of this application is to maintain conversations for applications that use data queues. The application is stopped by issuing the stop data queue (QSTPDTAQ) API function call.

Functions

A PC application can use the same data queue functions that are available to AS/400 applications. These include:

- Cancel request to retrieve data from a data queue
- Cancel request to retrieve data from a keyed data queue
- Clear data queue
- Create data queue
- Create keyed data queue
- Delete data queue
- Get message
- Put data to a queue
- Put data to a keyed queue
- Query data queue attributes
- Receive data from a queue
- Receive data from a keyed queue
- Receive data previously requested from a data queue
- Receive data previously requested from a keyed data queue
- Request to receive data from an AS/400 data queue
- Request to receive data from a keyed AS/400 data queue
- Send data to a queue
- Send data to a keyed queue
- Set the data conversion mode for subsequent data queues calls
- Stop data queue

PC command line functions for data queues are described in the *PC Support User's Guide*.

Access to data queues by the PC application program is completely transparent to the AS/400 system. It is as if the data queues are being accessed by AS/400 applications. No changes to the existing AS/400 applications are required.

Note: For information describing the high-level API for the data queues function for programs running in the Windows environment, see Chapter 26, "Data Queues Windows Application Program Interface and DDE Server."

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) on the AS/400 system for the API for the data queues function:

EHNDQ.H	C language include file.
EHNDQ.INC	Pascal language include file.

Data Queues Library Routines

The following sections for the data queues function describe in detail:

- Purpose
- Procedure and data structure declarations
- User-supplied values
- Returned values
- Return codes

Note: All pointers in the following sections are composed of a 32-bit value: a 16-bit segment and a 16-bit offset.

Clear Data Queue (QCLRDTAQ)

Purpose

Clears all messages from a specified remote AS/400 data queue.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                          /* Library/data queue name*/
char far * s_name;                          /* System name            */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QCLRDTAQ( char far * ,  /* Library/data queue name*/
                               char far * ); /* System name            */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ============================================================================
   declare external functions
   ============================================================================ *)
 function QCLRDTAQ(vars q_name:char;       (* Library/data queue name      *)
                   vars s_name:char;       (* System name                  *)
                   )          :integer; extern;
```

Parameters

q_name The name of the data queue to clear. This name consists of the name of the library the queue is located in followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

Additional Information

It is assumed that the data queue exists prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG, (described in "Get Message (QGETMSG)" on page 16-16 can be used to retrieve the message text associated with the error.

Cancel Previous Request (QCNLREQ)

Purpose

This call cancels a previous QRCVREQ when a QRCVDATA has not already been issued and data returned. Any data that has been received is discarded.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                    /* Queue name              */
char far * s_name;                    /* System name             */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QCNLREQ(  char far * ,  /* Queue name            */
                               char far * ); /* System name           */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ======================================================================
   declare external functions
   ====================================================================== *)

 function QCNLREQ(vars q_name:char;       (* Queue name                  *)
                  vars s_name:char        (* System name                 *)
                  )          :integer; extern;
```

Parameters

q_name The name of the data queue to have the request cancelled for. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used. This name must be equal to what was specified in the QRCVREQ call.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

Additional Information

It is assumed that a QRCVREQ call has been issued prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Cancel Previous Keyed Request (QCNLRQKY)

Purpose

This call cancels a previous QRCVRQKY when a QRCVDTKY has not already been issued and data returned. Any data that has been received is discarded.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Queue name              */
char far * s_name;                      /* System name             */
char far * k_buf;                       /* Key buffer              */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QCNLRQKY  char far * ,  /* Queue name              */
                               char far * ,  /* System name             */
                               char far * ); /* Key buffer              */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ============================================================================
   declare external functions
   ============================================================================ *)
 function QCNLRQKY (vars q_name:char;       (* Queue name                  *)
                    vars s_name:char;       (* System name                 *)
                    vars k_buf:char         (* Key buffer                  *)
                   )          :integer; extern;
```

Parameters

q_name The name of the data queue to have the request cancelled for. This name consists of library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used. This name must be equal to what was specified in the QRCVRQKY call.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

k_buf Key of the message specified when QRCVRQKY was issued. This key must be equal to that specified in the QRCVRQKY API.

Additional Information

It is assumed that a QRCVRQKY call has been issued prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Create Data Queue (QCRTDTAQ)

Purpose

Allows the PC application to create non-keyed data queues on the remote AS/400 system.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Library/data queue name*/
char far * s_name;                      /* System name            */
long m_len;                             /* Maximum queue length   */
int seqnce;                             /* Sequence               */
int force;                              /* Force                  */
int auth;                               /* Authority              */
int s_id;                               /* Sender ID              */
char far * t_buf;                       /* Text                   */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

/* Sequence values */
#define DQ_SEQ_LIFO   0
#define DQ_SEQ_FIFO   1

#define extentry _saveregs pascal far

extern int extentry  QCRTDTAQ( char far * ,  /* Library/data queue name*/
                               char far * ,  /* System name            */
                               long ,        /* Maximum queue length   */
                               int  ,        /* Sequence               */
                               int  ,        /* Force                  */
                               int  ,        /* Authority              */
                               int  ,        /* Sender ID              */
                               char far * ); /* Text                   */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;

(* Sequence values *)
       dq_seq_lifo    = 0;
       dq_seq_fifo    = 1;
(* ========================================================================
   declare external functions
   ======================================================================== *)
 function QCRTDTAQ(vars q_name:char;       (* Library/data queue name      *)
                   vars s_name:char;       (* System name                  *)
                        m_len :integer4;   (* Maximum queue length         *)
                        seqnce:integer;    (* Sequence                     *)
                        force :integer;    (* Force                        *)
                        auth  :integer;    (* Authority                    *)
                        s_id  :integer;    (* Sender ID                    *)
                   vars t_buf :char        (* Text                         *)
                   )          :integer; extern;
```

Parameters

q_name
: The name of the data queue to create. This name consists of the name of the library the queue is located in, followed by a forward slash and the name of the data queue. If the library is not specified, a default of *CURLIB is used and the data queue will be created in the current library, if possible.

s_name
: The name of the remote AS/400 system where the data queue is created. A null string indicates that the default system should be used.

m_len
: Identifies the maximum length a message can be in the remote AS/400 data queue. The maximum length that the personal computer can access is 31 744 bytes (31KB).

seqnce
: The messages are enqueued by one of the following methods:

  0 - LIFO: Last in, first out
  1 - FIFO: First in, first out

force
: Specifies whether messages are to be forced to auxiliary storage when they are enqueued. If maintaining data integrity is a concern, this value should be specified, but performance will be impacted.

  0 - No or False
  1 - Yes or True

auth
: Specifies the authority you are giving to users who do not have specific authority to the remote AS/400 data queue.

  0 - ALL
  1 - EXCLUDE
  2 - CHANGE
  3 - USE
  4 - LIBCRTAUT

  Note: To send data to a data queue, a user must have at least CHANGE authority to the queue.

s_id Indicates whether or not the process which enqueues messages will have information about itself entered with the message. This data can be requested when the data is dequeued.

0 - No or False
1 - Yes or True

Note: Unless sender ID information is required, a data queue should be created with this parameter set to the default value of No. There is a slight performance impact when using the send and receive commands on a data queue with sender IDs. Information about the process which enqueued the message is:

- Job Name - 10 characters
- User Name - 10 characters
- Job Number - 6 characters
- User Profile - 10 characters

t_buf Pointer to the description of the remote AS/400 data queue. This string must end with a null (X'00'). The maximum length is 50 characters.

Additional Information

It is assumed that the data queue does not exist prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG, (described in "Get Message (QGETMSG)" on page 16-16 can be used to retrieve the message text associated with the error.

Create Keyed Data Queue (QCRTDQKY)

Purpose

Allows the PC application to create keyed data queues on the remote AS/400 system.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Library/data queue name*/
char far * s_name;                      /* System name            */
long m_len;                             /* Maximum queue length   */
int force;                              /* Force                  */
int auth;                               /* Authority              */
int s_id;                               /* Sender ID              */
char far * t_buf;                       /* Text                   */
int k_len;                              /* Key length             */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QCRTDQKY( char far * ,  /* Library/data queue name*/
                               char far * ,  /* System name            */
                               long ,        /* Max length             */
                               int  ,        /* Force                  */
                               int  ,        /* Authority              */
                               int  ,        /* Sender ID              */
                               char far * ,  /* Text                   */
                               int  );       /* Key length             */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ======================================================================
   declare external functions
   ====================================================================== *)
 function QCRTDQKY(vars q_name:char;       (* Data queue name             *)
                   vars s_name:char;       (* System name                 *)
                        m_len :integer4;   (* Maximum queue length        *)
                        force :integer;    (* Force                       *)
                        auth  :integer;    (* Authority                   *)
                        s_id  :integer;    (* Sender ID                   *)
                   vars t_buf :char;       (* Text                        *)
                        k_len :integer     (* Key length                  *)
                   )          :integer; extern;
```

Parameters

q_name — The name of the data queue to create. This name consists of the name of the library the queue is located in, followed by a forward slash and the name of the data queue. If the library is not specified, a default of *CURLIB is used.

s_name — The name of the remote AS/400 system where the data queue is created. A null string indicates that the default system should be used.

m_len — Identifies the maximum length a message can be in the remote AS/400 data queue. The maximum length that the personal computer can access is 31 744 bytes (31K).

force — Specifies whether messages are to be forced to auxiliary storage when they are enqueued. If maintaining data integrity is a concern, this value should be specified, but performance will be impacted.

0 - No or False
1 - Yes or True

The default is No.

auth — Specifies the authority you are giving to users who do not have specific authority to the remote AS/400 data queue.

0 - ALL
1 - EXCLUDE
2 - CHANGE
3 - USE
4 - LIBCRTAUT

Note: To send data to a data queue, a user must have at least CHANGE authority to the queue.

s_id — Indicates whether or not the process which enqueues messages will have information about itself entered with the message. This data can be requested when the data is dequeued.

0 - No or False
1 - Yes or True

Note: Unless sender ID information is required, a data queue should be created with this parameter set to the default value of No. There is a slight performance impact when using the send and receive commands on a data queue with sender IDs. Information about the process which enqueued the message is:

- Job Name - 10 characters
- User Name - 10 characters
- Job Number - 6 characters
- User Profile - 10 characters

t_buf — The description of the remote AS/400 data queue. This string must end with a null (X'00'). The maximum length is 50 characters.

k_len — Length of the key for the keyed data queue.

Additional Information

It is assumed that the data queue does not exist prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Delete Data Queue (QDLTDTAQ)

Purpose

This function clears all of the messages on the data queue and then deletes the data queue on the remote AS/400 system.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Library/data queue name */
char far * s_name;                      /* System name             */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QDLTDTAQ( char far * ,  /* Library/data queue name*/
                               char far * ); /* System name           */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ========================================================================
   declare external functions
   ======================================================================== *)
 function QDLTDTAQ(vars q_name:char;       (* Library/data queue name      *)
                   vars s_name:char        (* System name                  *)
                   )          :integer; extern;
```

Parameters

q_name — The name of the data queue to delete. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name — The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

Additional Information

It is assumed that the data queue exists prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Get Capability (QDQGTCAP)

Purpose

This call returns the functional level of the data queues module supporting communications to the AS/400 system.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QDQGTCAP ( int far  *);  /* Functional level            */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ============================================================================
   declare external functions
   ============================================================================ *)
 function QDQGTCAP(vars q_cap:integer      (* Functional level             *)
                    )         :integer; extern;
```

Parameters

The integer value that is returned indicates the functional level of the module that the data queues API uses to send requests to and receive requests from the AS/400 system.

Possible values are:

1 Data queues supports the following entry points:

QCLRDTAQ Clear data queue

QCNLREQ	Cancel previous request for data
QCNLRQKY	Cancel previous request for data from a keyed queue
QCRTDTAQ	Create data queue
QCRTDQKY	Create keyed data queue
QDLTDTAQ	Delete data queue
QDQGTCAP	Get capability
QGETMSG	Get message
QPUTDTAQ	Put data to a queue
QPUTDQKY	Put data to a keyed queue
QQRYDTAQ	Query data queue
QRCVDATA	Receive data previously requested from a queue
QRCVDTKY	Receive data previously requested from a keyed queue
QRCVDTAQ	Receive data from a queue
QRCVDQKY	Receive data from a keyed queue
QRCVREQ	Receive request for data
QRCVRQKY	Receive request for data from a keyed queue
QSETMODE	Set data queues mode
QSNDDTAQ	Send data to a queue
QSNDDQKY	Send data to a keyed queue
QSTPDTAQ	Stop data queue

>1 Reserved for future enhancements to data queues.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

Get Message (QGETMSG)

Purpose

When an error occurs on any of the other data queues calls, a message is set up. This call retrieves the message text associated with a previous error. Messages are available in reverse order of their calling order. That is, the most recent message is retrieved first, and so on. Up to 10 messages are available.

The user must allocate a 152-character buffer to hold the message text.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * s_name;                          /* System name            */
char far * m_buf;                           /* 152 byte message buffer*/
long m_len;                                 /* Message length         */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QGETMSG ( char far * ,  /* System name                */
                               char far * ,  /* Message buffer (152 char)  */
                               int far * ) ; /* Message data length address */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ============================================================================
   declare external functions
   ============================================================================ *)
 function QGETMSG(vars s_name :char;       (* System name                 *)
                  vars m_buf  :char;       (* 152 byte message buffer     *)
                  vars m_len  :integer     (* Message length              *)
                  )           :integer; extern;
```

Parameters

s_name — The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

m_buf — A buffer containing the returned error message. A buffer of 152 characters must be allocated.

m_len — Actual length of the error message text received. The following values are valid:

- zero value - there are no messages to receive.
- positive value - the actual length of the error message. More than one message may be available as a result of an error. To determine if there are more messages, your program should read the message buffer until a length of zero is returned.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs on the GETMSG API call, the message buffer contains the text of the error message that has caused the error.

Put Data to a Queue (QPUTDTAQ)

Purpose

Sends messages to a data queue on a remote AS/400 without waiting for a reply. The AS/400 system does not acknowledge each time a message is successfully enqueued. This improves performance.

However, if an error occurs on the QPUTDTAQ call, the message text is not available until the next data queue command other than put data is issued. For example, if a QPUTDTAQ call is issued and fails, and then a send is issued, messages for both the put and the send are available with the GETMSG API call.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Library/data queue name */
char far * s_name;                      /* System name             */
char far * d_buf;                       /* Data buffer             */
long d_len;                             /* Data length             */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QPUTDTAQ( char far * , /* Library/data queue name */
                               char far * , /* System name             */
                               char far * , /* Data buffer             */
                               long );      /* Data length             */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ========================================================================
   declare external functions
   ======================================================================== *)
 function QPUTDTAQ(vars q_name:char;       (* Library/data queue name      *)
                   vars s_name:char;       (* System name                  *)
                   vars d_buf :char;       (* Data buffer                  *)
                        d_len :integer4    (* Data length                  *)
                   )          :integer; extern;
```

Parameters

q_name — The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name — The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

d_buf — A buffer of user-defined data that is to be placed on the specified data queue.

d_len — Integer value indicating the length of the message to be sent.

Additional Information

It is assumed that the data queue exists prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Put Data to a Keyed Queue (QPUTDQKY)

Purpose

Sends messages to a keyed data queue on a remote AS/400 without waiting for a reply. The AS/400 system does not acknowledge each time a message is enqueued. This improves performance.

However, if an error occurs on the QPUTDQKY call, the message text is not available until the next data queue command other than put data is issued. For example, if a put data call is issued and fails, and then a send is issued, messages for both the put and the send are available with the GETMSG API call.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Library/data queue name */
char far * s_name;                      /* System name             */
char far * d_buf;                       /* Data buffer             */
long d_len;                             /* Data length             */
char far * k_buf;                       /* Key buffer              */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QPUTDQKY( char far * ,  /* Library/data queue name*/
                               char far * ,  /* System name            */
                               char far * ,  /* Data buffer            */
                               long       ,  /* Data length            */
                               char far * ); /* Key buffer             */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ============================================================================
   declare external functions
   ============================================================================ *)
 function QPUTDQKY(vars q_name:char;       (* Library/data queue name      *)
                   vars s_name:char;       (* System name                  *)
                   vars d_buf :char;       (* Data buffer                  *)
                        d_len :integer4;   (* Data length                  *)
                   vars k_buf :char        (* Key buffer                   *)
                   )          :integer; extern;
```

Parameters

q_name The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

d_buf A buffer of user-defined data that is to be placed on the specified data queue.

d_len Integer value indicating the length of the message to be sent.

k_buf Key of the message to be enqueued on the remote data queue. This key must be the same length as the key length specified when the data queue was created. This length can be obtained using the Query Data Queue (QQRYDTAQ) routine.

Additional Information

It is assumed that the data queue exists prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Query Data Queue (QQRYDTAQ)

Purpose

Allows the PC application to query the attributes of a data queue on the remote AS/400 system. These are the attributes that the data queue was created with.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Library/data queue name */
char far * s_name;                      /* System name             */
long m_len;                             /* Maximum queue length    */
int seqnce;                             /* Sequence                */
int force;                              /* Force                   */
int s_id;                               /* Sender ID               */
char far * t_buf;                       /* Text                    */
int k_len;                              /* Key length              */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

/* Sequence values */
#define DQ_SEQ_LIFO   0
#define DQ_SEQ_FIFO   1
#define DQ_SEQ_Keyed  2

#define extentry _saveregs pascal far

extern int extentry  QQRYDTAQ( char far * ,  /* Library/data queue name*/
                               char far * ,  /* System name            */
                               long far *,   /* Max length             */
                               int far * ,   /* Sequence               */
                               int far * ,   /* Force                  */
                               int far * ,   /* Sender ID              */
                               char far * ,  /* Text                   */
                               int far * );  /* Key length             */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;

(* Sequence values *)
       dq_seq_lifo    = 0;
       dq_seq_fifo    = 1;
       dq_seq_keyed   = 2;
(* ===================================================================
   declare external functions
   =================================================================== *)
 function QQRYDTAQ(vars q_name:char;       (* Data queue name             *)
                   vars s_name:char;       (* System name                 *)
                   vars m_len :integer4;   (* Maximum queue length        *)
                   vars seqnce:integer;    (* Sequence                    *)
                   vars force :integer;    (* Force                       *)
                   vars s_id  :integer;    (* Sender ID                   *)
                   vars t_buf :char;       (* Text                        *)
                   vars k_len :integer     (* Key length                  *)
                   )          :integer; extern;
```

Parameters

q_name The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name The name of the remote AS/400 system where the data queue is located. A null indicates that the default system should be used.

m_len Returns the maximum size of messages in the remote AS/400 data queue.

seqnce The messages are queued by one of the following methods.

0 - LIFO: Last in, first out
1 - FIFO: First in, first out
2 - KEYED: Ordered by an associated key

force Specifies whether messages are to be forced to auxiliary storage when they are enqueued. If maintaining data integrity is a concern, this value should be specified, but performance will be impacted.

0 - No or False
1 - Yes or True

s_id Indicates whether or not the process which enqueues messages will have information about itself entered with the message.

0 - No or False
1 - Yes or True

Note: Unless sender ID information is required, a data queue should be created with this parameter set to the default value of No. There is a slight performance impact when using the send and receive commands on a data queue with sender IDs. Information about the process which enqueued the message is:

- Job Name - 10 characters
- User Name - 10 characters
- Job Number - 6 characters
- User Profile - 10 characters

t_buf Description of the remote AS/400 data queue.

k_len Length of the key for keyed data queues. This parameter is zero if the queue is LIFO or FIFO.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Receive Data from a Queue (QRCVDTAQ)

Purpose

Retrieves a message from a data queue on a remote AS/400 system. The call waits for the message for the specified amount of time.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Library/data queue name */
char far * s_name;                      /* System name             */
long w_time;                            /* Wait time               */
int s_id;                               /* Sender ID               */
char far * d_buf;                       /* Data buffer             */
long d_len;                             /* Data length             */
char far * s_info;                      /* Sender ID information   */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QRCVDTAQ( char far * ,  /* Library/data queue name*/
                               char far * ,  /* System name           */
                               long ,        /* Wait time             */
                               int  ,        /* Sender ID             */
                               char far * ,  /* Data buffer           */
                               long far * ,  /* Data length           */
                               char far * ); /* Sender ID information  */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ======================================================================
   declare external functions
   ====================================================================== *)
 function QRCVDTAQ(vars q_name:char;       (* Data queue name             *)
                   vars s_name:char;       (* System name                 *)
                        w_time:integer4;   (* Wait time                   *)
                        s_id  :integer;    (* Sender ID                   *)
                   vars d_buf :char;       (* Data buffer pointer         *)
                   vars d_len :integer4;   (* Data length                 *)
                   vars s_info:char        (* Sender information          *)
                   )          :integer; extern;
```

Parameters

q_name — The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name — The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

w_time — The integer value indicating how long to wait for a message to be retrieved from the remote AS/400 data queue.

negative value
: Indicates to wait until a message is available.

zero value
: Indicates not to wait. If a message is available, it is retrieved. If a message is not available, control is returned without waiting.

positive value
: Indicates to wait the specified number of seconds before returning control if a message is not available. Otherwise, the message is returned as soon as it is retrieved.

s_id — Indicates whether or not to return information about the process which enqueued the message.

0 - No or False
1 - Yes or True

Note: If the data queue was created with sender ID = No, the sender ID information field will be blank even if the information is requested on the QRCVDTAQ call.

d_buf — The buffer containing the retrieved data. The application must create a data area large enough to contain the returned data.

d_len — Integer value indicating the length of the retrieved message. If no message is returned, this value is zero. In DOS, this is also a user-supplied value containing the length of the data buffer.

s_info — Information about the process which enqueued the message:

- Job Name - 10 characters
- User Name - 10 characters
- Job Number - 6 characters
- User Profile - 10 characters

Additional Information

It is assumed that the data queue exists prior to this call.

The current mode determines whether this call performs destructive (receive) or nondestructive (peek) reads. See "Set Data Queues Mode (QSETMODE)" on page 16-40 for more information.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Receive Data from a Keyed Queue (QRCVDQKY)

Purpose

Retrieves a message from a keyed data queue on a remote AS/400 system. The call waits for the message for the specified amount of time.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                  /* Library/data queue name */
char far * s_name;                  /* System name             */
long w_time;                        /* Wait time               */
char far * k_buf;                   /* Key buffer              */
char far * s_order;                 /* Search order            */
int  s_id;                          /* Sender ID               */
char far * d_buf;                   /* Data buffer             */
long d_len;                         /* Data length             */
char far * k_ret;                   /* Key return              */
char far * s_info;                  /* Sender ID information   */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QRCVDQKY( char far * ,  /* Library/data queue name*/
                               char far * ,  /* System name            */
                               long ,        /* Wait time              */
                               char far * ,  /* Key buffer             */
                               char far * ,  /* Search order (2 chars) */
                               int  ,        /* Sender ID              */
                               char far * ,  /* Data buffer            */
                               long far * ,  /* Data length            */
                               char far * ,  /* Key returned           */
                               char far * ); /* Sender ID information  */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ========================================================================
   declare external functions
   ======================================================================== *)
 function QRCVDQKY(vars q_name:char;       (* Data queue name              *)
                   vars s_name:char;       (* System name                  *)
                        w_time:integer4;   (* Wait time                    *)
                   vars k_buf :char;       (* Key buffer                   *)
                   vars s_ordr:char;       (* Search order                 *)
                        s_id  :integer;    (* Sender ID                    *)
                   vars d_buf :char;       (* Data buffer                  *)
                   vars d_len :integer4;   (* Data length                  *)
                   vars k_ret :char;       (* Key return                   *)
                   vars s_info:char        (* Sender information           *)
                   )          :integer; extern;
```

Parameters

q_name — The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400system.
2. *CURLIB is also valid as a generic library name.

s_name
: The name of the remote AS/400 system where the data queue is located. A null pointer indicates that the default system should be used.

w_time
: The integer value indicating how long to wait for a message to be retrieved from the remote AS/400 data queue.

  negative value
  : Indicates to wait until a message is retrieved.

  zero value
  : Indicates not to wait. If a message is available, it will be retrieved. If a message is not available, control is returned without waiting.

  positive value
  : Indicates to wait the specified number of seconds before returning control if a message is not retrieved. Otherwise, the message is returned as soon as it is retrieved.

k_buf
: Key of the message to be retrieved from the remote data queue. This key must be the same length as the key length specified when the data queue was created. This length can be obtained using the Query Data Queue (QQRYDTAQ) routine.

s_ordr
: Relational operator used in conjunction with the key to specify which message to select. (For example, >= KEY or < KEY.) The following are valid search orders: EQ, NE, GT, GE, LT, and LE.

s_id
: Indicates whether or not to return information about the process which enqueued the message.

  Note: If the data queue was created with sender ID = No, the sender ID information field will be blank even if the information is requested on the QRCVDQKY call.

  0 - No or False
  1 - Yes or True

d_buf
: The buffer containing the retrieved data. The application must create a data area large enough to contain the returned data.

d_len
: Integer value indicating the length of the retrieved message. In DOS, this is also a user-supplied value containing the length of the data buffer.

k_ret
: Key of the message that was actually retrieved. This may be different than the key specified to search for (for example, if key AA is chosen with the search order of greater than or equal to (GE), the key of the message that is actually retrieved could be AB or anything else greater than AA).

s_info
: Information about the process which enqueued the message:

  - Job Name - 10 characters
  - User Name - 10 characters

- Job Number - 6 characters
- User Profile - 10 characters

Additional Information

It is assumed that the data queue exists prior to this call.

The current mode determines whether this call performs destructive (receive) or nondestructive (peek) reads. See "Set Data Queues Mode (QSETMODE)" on page 16-40 for more information.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Receive Request for Data (QRCVREQ)

Purpose

Requests messages from a specific queue. These messages are retrieved later using QRCVDATA. The return values (such as the data buffer and sender ID information) are filled in whenever the data is sent to the personal computer. The QRCVDATA API call should be issued to get the length of the returned data. This indicates to the application that data has been returned.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                  /* Library/data queue name */
char far * s_name;                  /* System name             */
long w_time;                        /* Wait time               */
int s_id;                           /* Sender ID               */
char far * d_buf;                   /* Data buffer             */
long d_len;                         /* Data length             */
char far * s_info;                  /* Sender ID information   */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QRCVREQ( char far * ,  /* Library/data queue name*/
                              char far * ,  /* System name           */
                              long ,        /* Wait time             */
                              int  ,        /* Sender ID             */
                              char far * ,  /* Data buffer           */
                              long far * ,  /* Data length           */
                              char far * ); /* Sender ID information  */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ======================================================================
   declare external functions
   ====================================================================== *)
 function QRCVREQ(vars q_name:char;       (* Data queue name             *)
                  vars s_name:char;       (* System name                 *)
                       w_time:integer4;   (* Wait time                    *)
                       s_id  :integer;    (* Sender ID                   *)
                  vars d_buf :char;       (* Data buffer pointer         *)
                  vars d_len :integer4;   (* Data length                 *)
                  vars s_info:char        (* Sender information          *)
                  )          :integer; extern;
```

Parameters

q_name The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

w_time The integer value indicating how long to wait for a message to be retrieved from the remote AS/400 data queue.

negative value
Indicates to wait until a message is available.

zero value
Indicates not to wait. If a message is available, it is retrieved. If a message is not available, control is returned without waiting.

positive value
Indicates to wait the specified number of seconds before returning control if a message is not available. Otherwise, the message is returned as soon as it is retrieved.

s_id Indicates whether or not to return information about the process which enqueued the message.

0 - No or False
1 - Yes or True

Note: If the data queue was created with sender ID = No, the sender ID information field will be blank even if the information is requested on the QRCVREQ call.

d_buf The buffer containing the retrieved data. The application must create a data area large enough to contain the returned data.

d_len Integer value indicating the length of the retrieved message. In DOS, this is also a user-supplied value containing the length of the data buffer.

s_info Information about the process which enqueued the message:

- Job Name - 10 characters
- User Name - 10 characters
- Job Number - 6 characters
- User Profile - 10 characters

Additional Information

It is assumed that the data queue exists prior to this call.

If QRCVREQ is called more than once before issuing the corresponding call to QRCVDATA, the following buffers must be unique:

d_buf
d_len
s_info

The current mode determines whether this call performs destructive (receive) or nondestructive (peek) reads. See "Set Data Queues Mode (QSETMODE)" on page 16-40 for more information.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Receive Request for Data from a Keyed Queue (QRCVRQKY)

Purpose

Requests messages from a specific keyed queue. These messages are retrieved later using QRCVDTKY. The return values (such as the data buffer and sender ID information) are filled in whenever the data is sent to the personal computer. The QRCVDTKY API call should be issued to get the length of the returned data. This indicates to the application that data has been returned.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                  /* Library/data queue name */
char far * s_name;                  /* System name             */
long w_time;                        /* Wait time               */
char far * k_buf;                   /* Key buffer              */
char far * s_order;                 /* Search order            */
int  s_id;                          /* Sender ID               */
char far * d_buf;                   /* Data buffer             */
long d_len;                         /* Data length             */
char far * k_ret;                   /* Key return              */
char far * s_info;                  /* Sender ID information   */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QRCVRQKY( char far * ,  /* Library/data queue name*/
                               char far * ,  /* System name            */
                               long ,        /* Wait time              */
                               char far * ,  /* Key buffer             */
                               char far * ,  /* Search order (2 chars) */
                               int  ,        /* Sender ID              */
                               char far * ,  /* Data buffer            */
                               long far * ,  /* Data length            */
                               char far * ,  /* Key returned           */
                               char far * ); /* Sender ID information  */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ========================================================================
   declare external functions
   ======================================================================== *)
 function QRCVRQKY(vars q_name:char;      (* Data queue name              *)
                   vars s_name:char;      (* System name                  *)
                        w_time:integer4;  (* Wait time                    *)
                   vars k_buf :char;      (* Key buffer                   *)
                   vars s_ordr:char;      (* Search order                 *)
                        s_id  :integer;   (* Sender ID                    *)
                   vars d_buf :char;      (* Data buffer pointer          *)
                   vars d_len :integer4;  (* Data length                  *)
                   vars k_ret :char;      (* Key return                   *)
                   vars s_info:char       (* Sender information           *)
                   )          :integer; extern;
```

Parameters

q_name The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

w_time The integer value indicating how long to wait for a message to be retrieved from the remote AS/400 data queue.

negative value
: Indicates to wait until a message is available.

zero value
: Indicates not to wait. If a message is available, it is retrieved. If a message is not available, control is returned without waiting.

positive value
: Indicates to wait the specified number of seconds before returning control if a message is not available. Otherwise, the message is returned as soon as it is retrieved.

k_buf Key of the message to be retrieved from the remote data queue. This key must be the same length as the key length specified when the data queue was created. This length can be obtained using the Query Data Queue (QQRYDTAQ) routine.

s_ordr Relational operator used in conjunction with the key to specify which message to select. (For example, >= KEY or < KEY.) The following are valid search orders: EQ, NE, GT, GE, LT, and LE.

s_id Indicates whether or not to return information about the process which enqueued the message.

0 - No or False
1 - Yes or True

Note: If the data queue was created with sender ID = No, the sender ID information field is blank even if the information is requested on the QRCVRQKY call.

d_buf The buffer containing the retrieved data. The application must create a data area large enough to contain the returned data.

d_len Integer value indicating the length of the retrieved message. In DOS, this is also a user-supplied value containing the length of the data buffer.

k_ret Key of the message that was actually retrieved. This may be different than the key specified to search for (for example, if key AA is chosen with the search order of greater than or equal to (GE), the key of the message that is actually retrieved could be AB or anything else greater than AA).

s_info Information about the process which enqueued the message:

- Job Name - 10 characters
- User Name - 10 characters
- Job Number - 6 characters

- User Profile - 10 characters

Additional Information

It is assumed that the data queue exists prior to this call.

If QRCVRQKY is called more than once before the corresponding call to QRCVDTKY, the following buffers must be unique:

d_buf
d_len
s_info
k_ret

The current mode determines whether this call performs destructive (receive) or nondestructive (peek) reads. See "Set Data Queues Mode (QSETMODE)" on page 16-40 for more information.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Receive Data Previously Requested (QRCVDATA)

Purpose

Determines if a QRCVREQ call has been processed. If data is available, the parameters specified in the QRCVREQ call are filled in. If data is not yet available, the return code is zero, and the length value specified in the QRCVREQ call is also zero.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                    /* Queue name              */
char far * s_name                     /* System name             */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QRCVDATA( char far * ,  /* Queue name            */
                               char far * ); /* System name           */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ================================================================
   declare external functions
   ================================================================ *)
 function QRCVDATA (vars q_name:char;       (* Queue name                *)
                    vars s_name:char        (* System name               *)
                  )          :integer; extern;
```

Parameters

q_name The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

Additional Information

It is assumed that a QRCVREQ call has been issued prior to this call.

The current mode determines whether this call performs destructive (receive) or nondestructive (peek) reads. See "Set Data Queues Mode (QSETMODE)" on page 16-40 for more information.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Receive Data Previously Requested from a Keyed Queue (QRCVDTKY)

Purpose

Determines if a QRCVRQKY call has been processed. If data is available, the parameters specified in the QRCVRQKY call are filled in. If data is not yet available, the return code is zero, and the length value specified in the QRCVRQKY call is also zero.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Queue name              */
char far * s_name;                      /* System name             */
char far * k_buf;                       /* Key buffer              */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QRCVDTKY( char far * ,  /* Queue name             */
                               char far * ,  /* System name            */
                               char far * ); /* Key buffer             */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ========================================================================
   declare external functions
   ======================================================================== *)
 function QRCVDTKY (vars q_name:char;       (* Queue name                 *)
                    vars s_name:char;       (* System name                *)
                    vars k_buf:char         (* Key buffer                 *)
                   )          :integer; extern;
```

Parameters

q_name The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

k_buf Key of the message specified when QRCVRQKY was issued. This key must be equal to that specified in the QRCVRQKY API.

Additional Information

It is assumed that a QRCVRQKY call has been issued prior to this call.

The current mode determines whether this call performs destructive (receive) or nondestructive (peek) reads. See "Set Data Queues Mode (QSETMODE)" on page 16-40 for more information.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Set Data Queues Mode (QSETMODE)

Purpose

Sets the current running mode of data queues. The mode determines if data is converted between ASCII and EBCDIC, and if receive functions perform a destructive (receive) or nondestructive (peek) read.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * s_name;                          /* System name             */
long modeid;                                /* Should data be converted*/
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QSETMODE ( char far * ,    /* System name         */
                                      long);    /* Convert data        */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ======================================================================
   declare external function
   ====================================================================== *)
 function QSETMODE(vars s_name :char;              (* System name  *)
                   vars modeid integer4            (* Convert data *)
                   )          :integer; extern;
```

Parameters

s_name The name of the remote AS/400 system for which the conversation mode is set. A null string indicates that the default system should be used.

modeid A value that indicates whether data should be converted and whether receive functions should perform destructive (receive) or nondestructive (peek) reads.

Setting the first bit on with this parameter causes data to be translated in any later calls. Setting the second bit on with this parameter causes all later calls to be nondestructive (peek) reads. Use the following constant values to set the modeid parameter:

Constant Value	Function
DQ_YES_XLAT	Translate between ASCII and EBCDIC
DQ_NO_XLAT	No translation
DQ_YES_PEEK	Peek data queue
DQ_NO_PEEK	Receive data queue
DQ_XLAT_PEEK	Translate and peek
DQ_XLATN_PEEKY	No translate and peek
DQ_XLATY_PEEKN	Translate and receive
DQ_NO_XLAT_PEEK	No translate and receive

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Send Data to a Queue (QSNDDTAQ)

Purpose

Used to send data to a non-keyed data queue on a remote AS/400 system. The AS/400 system acknowledges each message enqueued to ensure that an error has not occurred.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                      /* Library/data queue name */
char far * s_name;                      /* System name             */
char far * d_buf;                       /* Data buffer             */
long d_len;                             /* Data length             */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QSNDDTAQ( char far * ,  /* Library/data queue name*/
                               char far * ,  /* System name            */
                               char far * ,  /* Data buffer            */
                               long );       /* Data length            */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ========================================================================
   declare external function
   ======================================================================== *)
 function QSNDDTAQ(vars q_name:char;       (* Data queue name             *)
                   vars s_name:char;       (* System name                 *)
                   vars d_buf :char;       (* Data buffer                 *)
                        d_len :integer4    (* Data length                 *)
                   )          :integer; extern;
```

Parameters

q_name — The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name — The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

d_buf — The buffer of user-defined data that is to be placed on the specified data queue.

d_len — Integer value indicating the length of the message to be sent.

Additional Information

It is assumed that the data queue exists prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Send Data to a Keyed Queue (QSNDDQKY)

Purpose

Used to send data to a keyed data queue on a remote AS/400 system. The AS/400 system acknowledges each message enqueued to ensure that an error has not occurred.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * q_name;                    /* Library/data queue name */
char far * s_name;                    /* System name             */
char far * d_buf;                     /* Data buffer             */
long d_len;                           /* Data length             */
char far * k_buf;                     /* Key buffer              */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QSNDDQKY( char far * ,  /* Library/data queue name*/
                               char far * ,  /* System name            */
                               char far * ,  /* Data buffer            */
                               long       ,  /* Data length            */
                               char far * ); /* Key                    */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ======================================================================
   declare external functions
   ====================================================================== *)
 function QSNDDQKY(vars q_name:char;       (* Data queue name             *)
                   vars s_name:char;       (* System name                 *)
                   vars d_buf :char;       (* Data buffer                 *)
                        d_len :integer4;   (* Data length                 *)
                   vars k_buf :char        (* Key buffer                  *)
                   )          :integer; extern;
```

Parameters

q_name — The name of the data queue to access. This name consists of the library name where the queue is located followed by a forward slash and the name of the data queue. If the library is not specified, a default of *LIBL is used.

Notes:

1. For better performance, specify a library name. The use of *LIBL causes more work for the AS/400 system.
2. *CURLIB is also valid as a generic library name.

s_name — The name of the remote AS/400 system where the data queue is located. A null string indicates that the default system should be used.

d_buf — A buffer of user-defined data that is to be placed on the specified data queue.

d_len — Integer value indicating the length of the message to be sent.

k_buf — Key of the message that is being sent to the remote data queue. This key must be the same length as the key length specified when the data queue was created. This length can be obtained using the Query Data Queue (QQRYDTAQ) routine.

Additional Information

It is assumed that the data queue exists prior to this call.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Stop Data Queue (QSTPDTAQ)

Purpose

Allows applications to end the data queue conversations with the remote AS/400 system specified. This involves deallocating the send and receive conversations if no other data queue applications are using them. For the OS/2 environment, when the last remaining pair of conversations is ended, the data queue background task is ended and all associated resources are freed and are returned to the operating system. The DOS background task automatically restarts when the next API call is made. For DOS, only the conversations are ended. Use the RMVPCS function to free resources allocated when LOADDQ was run.

Procedure Declaration

C

Statements to use in application programs:

```
#include <EHNDQ.H>

char far * s_name;                    /* System name          */
```

Declarations provided in include files:

```
#define DQ_OK 0

/* Data Queue Errors */
#define DQ_COMM_ERROR   1
#define DQ_AS400_ERROR  2
#define DQ_PC_ERROR     3

#define extentry _saveregs pascal far

extern int extentry  QSTPDTAQ( char far * );  /* System name          */
```

PASCAL

Statements to use in application programs:

```
{$include: 'EHNDQ.INC'}
```

Declarations provided in include files:

```
 const dq_ok          = 0;

(* Data Queue Errors *)
       dq_comm_error  = 1;
       dq_as400_error = 2;
       dq_pc_error    = 3;
(* ========================================================================
   declare external functions
   ======================================================================== *)
 function QSTPDTAQ(vars s_name:char        (* System name                 *)
                   )          :integer; extern;
```

Parameters

s_name — The name of the remote AS/400 system to stop conversations with. This value is required. The possible values are:

- Specified remote AS/400 system.
- A null value - indicates that the conversations with the default system are to be ended.
- *ALL - indicates that all pairs of conversations are to be ended. This stops all active conversations currently running data queues functions with this application. It is recommended that all active conversations are stopped at the end of the application. This does not affect any other application that is working with data queues on the same AS/400 system.

Return Codes

Return Code	Description
0	APIOK - Successful.
1	COMM_ERROR - A communications error occurred.
2	EXCEPT_ERROR - An exception error occurred on the remote AS/400 system.
3	PC_ERROR - A PC resource error has occurred (such as all system semaphores are in use or are unable to allocate memory).

If an error occurs, the function QGETMSG (described in "Get Message (QGETMSG)" on page 16-16) can be used to retrieve the message text associated with the error.

Chapter 17. Data Transform Library High-Level Application Program Interface

This chapter describes the high-level language API for transforming data. These functions may be used in a DOS or an OS/2 environment.

The APIs support the following languages:

- IBM Pascal/2
- IBM C/2
- IBM Macro Assembler/2

The library routines described in this chapter are written to support large memory models.

Sample statements, procedure definitions, or data structure definitions for C and Pascal are provided in this chapter. An assembler language program could call the functions by sequence, so there are no such definitions given for assembler language.

Sample programs, include files, and other information for using APIs are provided in the PC Support tools folder (QIWSTOOL). See Appendix C, "PC Support Tools Folder" for information about the contents of the tools folder and how to use it.

DOS Environment

This section defines the data transform API for PC programs running in a DOS environment.

Note: For information describing the high-level API for the data transform function for programs running in the Windows environment, see Chapter 27, "Data Transform Windows Application Program Interface."

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the data transform API:

DTXLAT.H	C language include file
DTXLAT_A.INC	Macro Assembler include file
DTXLAT_T.INC	Pascal language type section
DTXLAT_C.INC	Pascal language constant section
DTXLAT_E.INC	Pascal language external procedure declarations

Functions

The following functions are provided:

- 2-byte integer to 6-byte ASCII numeric
- 4-byte integer to 11-byte ASCII numeric
- 6-byte ASCII numeric to 2-byte integer
- 11-byte ASCII numeric to 4-byte integer
- Hexadecimal fields to equivalent ASCII characters
- ASCII characters to hexadecimal fields
- Packed decimal to packed decimal

- DOS random to packed decimal
- Packed decimal to DOS random
- Packed decimal to ASCII numeric
- ASCII numeric to packed decimal
- Zoned decimal to zoned decimal
- Zoned decimal to DOS random
- DOS random to zoned decimal
- Zoned decimal to ASCII numeric
- ASCII numeric to zoned decimal
- All printable characters from ASCII to EBCDIC
- All printable characters from EBCDIC to ASCII
- EBCDIC to EBCDIC
- Get ASCII-to-EBCDIC table
- Get EBCDIC-to-ASCII table

An entry point is created for each function. The sections on each function define the parameter list format, input and output, and return codes for each entry point.

OS/2 environment

IBM OS/2 EE Communications Manager (CM) provides an APPC interface for PC programs running in an OS/2 environment through dynamic link library (DLL) routines. This section describes the data transform DLL routines.

Include Files

The following files are provided in the PC Support tools folder (QIWSTOOL) for the data transform API:

DTXLAT.H	C language include file
DTXLAT_A.INC	Macro Assembler include file
DTXLAT_T.INC	Pascal language type section
DTXLAT_C.INC	Pascal language constant section
DTXLAT_E.INC	Pascal language external procedure declarations

Functions

The functions provided for OS/2 are the same as those provided for DOS.

Data Transform Library Routines

The following sections for the data transform function describe in detail:

- Purpose
- Procedure and data structure declarations
- User-supplied values
- Returned values
- Return codes

2-Byte Integer to 6-Byte ASCII Numeric (ITOASC)

Purpose

The ITOASC function converts a 2-byte integer, pointed to by the supplied parameter s, to a 6-byte ASCII string pointed to by the returned parameter t. The 2-byte integer is assumed to be stored in memory with the high-byte of a word first.

This function assumes that only 2 bytes are translated (-32 768 to 32 767).

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t; /* A pointer to the target */
char far *s; /* A pointer to the source */
```

Declarations provided in include files:

```
unsigned pascal itoasc(t,s);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file                }
{$include: 'dtxlat_c.inc'} { Type section include file                    }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ITOASC(t:ADSMEM,s:ADSMEM) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source }
```

Parameters

s A pointer to the 2-byte integer to be transformed.

t Returns a pointer to the 6-byte ASCII string.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion

4-Byte Integer to 11-Byte ASCII Numeric (LTOASC)

Purpose

The LTOASC function converts a 4-byte integer, pointed to by the supplied parameter s, to an 11-byte ASCII string pointed to by the returned parameter t. The 4-byte integer is assumed to be stored in memory with the high-word first and the high-byte of a word first.

This function assumes that only 4 bytes are translated (-2 147 483 648 to 2 147 483 647).

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t; /* A pointer to the target */
char far *s; /* A pointer to the source */
```

Declarations provided in include files:

```
unsigned pascal ltoasc(t,s);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION LTOASC(t:ADSMEM,s:ADSMEM) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source }
```

Parameters

s A pointer to the 4-byte integer to be transformed.

t Returns a pointer to the 11-byte ASCII string.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion

6-Byte ASCII Numeric to 2-Byte Integer (ASCTOI)

Purpose

The ASCTOI function converts a 6-byte ASCII string, pointed to by the supplied parameter s, to a 2-byte integer pointed to by the returned parameter t. Leading and trailing space characters are ignored. Spaces between the sign (+ or –) and the first digit are ignored. The 2-byte integer is stored in memory with the high-byte of a word first.

This function assumes that exactly 6 ASCII numeric characters are translated.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t; /* A pointer to the target */
char far *s; /* A pointer to the source */
```

Declarations provided in include files:

```
unsigned pascal asctoi(t,s);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ASCTOI(t:ADSMEM,s:ADSMEM) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source }
```

Parameters

s A pointer to the 6-byte ASCII string to be transformed.

t Returns a pointer to the 2-byte integer.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
X'0001'	DTX_OVERFLOW - Overflow error
Other values	Offset of the first untranslated character plus one

11-Byte ASCII Numeric to 4-Byte Integer (ASCTOL)

Purpose

The ASCTOL function converts an 11-byte ASCII string, pointed to by the supplied parameter s, to a 4-byte integer pointed to by the returned parameter t. Leading and trailing space characters are ignored. Spaces between the sign (+ or –) and the first digit are ignored. The 4-byte integer is stored in memory with the high-word first and the high-byte of a word first.

This function assumes that exactly 11 ASCII numeric characters are translated.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t; /* A pointer to the target */
char far *s; /* A pointer to the source */
```

Declarations provided in include files:

```
unsigned pascal asctol(t,s);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file                }
{$include: 'dtxlat_c.inc'} { Type section include file                    }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ASCTOL(t:ADSMEM,s:ADSMEM) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source }
```

Parameters

s A pointer to the 11-byte ASCII string to be transformed.

t Returns a pointer to the 4-byte integer.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
X'0001'	DTX_OVERFLOW - Overflow error
Other values	Offset of the first untranslated character plus one

Hexadecimal Data to ASCII Characters (HEXTOASC)

Purpose

The HEXTOASC function converts hexadecimal data, pointed to by the supplied parameter s, to ASCII characters pointed to by the returned parameter t for each half-byte.

This function assumes that the length of the hexadecimal data is from 1 to 256 bytes.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target     */
char far *s;  /* A pointer to the source     */
int len;      /* Bytes of the input hex data */
```

Declarations provided in include files:

```
unsigned pascal hextoasc(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file                }
{$include: 'dtxlat_c.inc'} { Type section include file                    }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION HEXTOASC(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Bytes of the input hex data}
```

Parameters

len — Number of bytes of the input hexadecimal data.

s — A pointer to the hexadecimal data to be transformed. The length of the hexadecimal data must be greater than or equal to the value specified for the parameter len.

t — Returns a pointer to the ASCII numeric data. The length of the output ASCII data is len*2.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion

ASCII Characters to Hexadecimal Data (ASCTOHEX)

Purpose

The ASCTOHEX function converts ASCII characters, pointed to by the supplied parameter s, to hexadecimal data pointed to by the returned parameter t by converting two ASCII bytes into one hexadecimal byte.

This function assumes that the ASCII numeric data is twice as long as the hexadecimal data, and the length is from 1 to 256 bytes. The ASCII character must be from 0 to 9 or from A to F.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target      */
char far *s;  /* A pointer to the source      */
int len;      /* Bytes of the output hex data */
```

Declarations provided in include files:

```
unsigned pascal asctohex(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ASCTOHEX(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Bytes of the output hex data }
```

Parameters

len — Number of bytes of the output hexadecimal data to be transformed.

s — A pointer to the ASCII numeric data to be transformed. The length of the ASCII data must be greater than or equal to len*2.

t — Returns a pointer to the hexadecimal data with length len.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
Other values	Offset of the first untranslated character plus one

Packed Decimal to Packed Decimal (PAKTOPAK)

Purpose

The PAKTOPAK function copies a packed decimal number, pointed to by the parameter s, to the area pointed to by the parameter t, byte by byte. The conversion continues until hexadecimal data that is not valid is encountered, or until the number of bytes specified by len are copied.

This function assumes that the packed decimal number is from X'0' to X'9' (but does not include the sign half-byte of the last half-byte) and the length is from 1 to 16 bytes.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target */
char far *s;  /* A pointer to the source */
int len;      /* Number of bytes         */
```

Declarations provided in include files:

```
unsigned pascal paktopak(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION PAKTOPAK(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Number of bytes }
```

Parameters

len	Number of bytes of the input packed decimal number to be copied.
s	A pointer to the packed decimal number to be copied.
t	Returns a pointer to the packed decimal number with length specified by len.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - No untranslatable characters
Other values	Offset of the first untranslated character plus one

DOS Random to Packed Decimal (DOSTOPAK)

Purpose

The DOSTOPAK function converts a packed decimal number in ASCII format, pointed to by the parameter s, to a packed decimal number in EBCDIC format pointed to by the parameter t.

This function assumes that the packed decimal number is from X'0' to X'9' and the sign half-byte, the low-nibble, is X'B' or X'3'. The length must be from 1 to 16 bytes.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target */
char far *s;  /* A pointer to the source */
int len;      /* Number of bytes         */
```

Declarations provided in include files:

```
unsigned pascal dostopak(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION DOSTOPAK(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Number of bytes }
```

Parameters

len — Number of bytes of the input packed decimal number to be transformed.

s — A pointer to the packed decimal number in ASCII format to be transformed.

t — Returns a pointer to the packed decimal number with length len in EBCDIC format.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
Other values	Offset of the first untranslated character plus one

Packed Decimal to DOS Random (PAKTODOS)

Purpose

The PAKTODOS function converts a packed decimal number in EBCDIC format, pointed to by the parameter s, to a packed decimal number in ASCII format pointed to by the parameter t. The X'D' or X'B' of sign half-byte, or last low-nibble, converts to X'B'; anything else converts to X'3'.

This function assumes that the packed decimal number is from X'0' to X'9' (but does not include the sign half-byte of the last half-byte) and the length is from 1 to 16 bytes.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target */
char far *s;  /* A pointer to the source */
int len;      /* Number of bytes         */
```

Declarations provided in include files:

```
unsigned pascal paktodos(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file              }
{$include: 'dtxlat_c.inc'} { Type section include file                  }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION PAKTODOS(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Number of bytes }
```

Parameters

len — Number of bytes of the input packed decimal number to be transformed.

s — A pointer to the packed decimal number in EBCDIC format to be transformed.

t — Returns a pointer to the packed decimal number with length len in ASCII format.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
Other values	Offset of the first untranslated character plus one

Packed Decimal to ASCII Numeric (PAKTOASC)

Purpose

The PAKTOASC function converts a packed decimal number, pointed to by the parameter s, to ASCII numeric data format pointed to by the parameter t with a decimal point at the position specified by decimal_position. The X'D' or X'B' of the sign half-byte converts to a negative number; anything else converts to a positive number in the ASCII numeric data format.

This function assumes that the packed decimal number is from X'0' to X'9' (but does not include the sign half-byte of the last half-byte) and the length is from 1 to 16 bytes. Also, 0 <= decimal_position < (len * 2).

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;              /* A pointer to the target                */
char far *s;              /* A pointer to the source                */
int len;                  /* Bytes of the input packed decimal data */
int decimal_position;     /* Place of decimal point                 */
```

Declarations provided in include files:

```
unsigned pascal paktoasc(t,s,len,decimal_position);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION PAKTOASC(t:ADSMEM,s:ADSMEM,len:INTEGER2,
                  decimal_position:INTEGER2) : WORD;
{char t              --  A pointer to the target
 char s              --  A pointer to the source
 int len             --  Bytes of the input packed decimal
 int decimal_position --  Place of decimal point }
```

Parameters

len	Number of bytes of the input packed decimal number.
decimal_position	Place of decimal point in the output ASCII data.
s	A pointer to the packed decimal number to be transformed. The length of the packed decimal number must be greater than or equal to len.
t	Returns a pointer to the ASCII numeric data. If decimal_position is zero, the output data length is len*2. If decimal_position is not zero, the output data length is len*2+1.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
Other values	Offset of the first untranslated character plus one

ASCII Numeric to Packed Decimal (ASCTOPAK)

Purpose

The ASCTOPAK function converts an ASCII number, pointed to by the parameter s, to a packed decimal number pointed to by the parameter t. Decimal_position specifies the position of the decimal point in the ASCII data. Leading and trailing spaces are ignored. Spaces between the sign (+ or –) and the first digit are ignored.

This function assumes that the ASCII numeric data is from 1 to 16 bytes long and 0 <= decimal_position < (len * 2).

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;              /* A pointer to the target                */
char far *s;              /* A pointer to the source                */
int len;                  /* Bytes of the output packed decimal data */
int decimal_position;     /* Place of decimal point                 */
```

Declarations provided in include files:

```
unsigned pascal asctopak(t,s,len,decimal_position);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ASCTOPAK(t:ADSMEM,s:ADSMEM,len:INTEGER2,
                  decimal_position:INTEGER2) : WORD;
{char t               --  A pointer to the target
 char s               --  A pointer to the source
 int len              --  Bytes of the output packed decimal data
 int decimal_position --  Place of decimal point }
```

Parameters

len	Number of bytes of the output packed decimal data to be transformed.
decimal_position	Place of decimal point of ASCII data.
s	A pointer to the ASCII numeric data to be transformed. If decimal_position is zero, the data length must be greater than or equal to len*2. If decimal_position is not zero, the data length must be greater than or equal to len*2 + 1 and must contain a decimal point. All data must be digits (0 to 9), signs (+ or –), or blanks.
t	Returns a pointer to the packed decimal number with length len.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
X'0001'	DTX_OVERFLOW - Overflow error
Other values	Offset of the first untranslated character plus one

Zoned Decimal to Zoned Decimal (ZONTOZON)

Purpose

The ZONTOZON function copies zoned decimal data, pointed to by the parameter s, to the area pointed to by the parameter t, byte by byte. The conversion continues until a byte that is not valid is encountered or the number of bytes specified by len are copied.

This function assumes that the left half-byte in the zoned decimal is X'F' (but does not include the last half-byte) and the right half-byte in the zoned decimal is from X'0' to X'9'. The length must be from 1 to 31 bytes.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target */
char far *s;  /* A pointer to the source */
int len;      /* Number of bytes         */
```

Declarations provided in include files:

```
unsigned pascal zontozon(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ZONTOZON(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Number of bytes }
```

Parameters

len	Number of bytes of the input zoned decimal data to be copied.
s	A pointer to the zoned decimal data to be copied.
t	Returns a pointer to the zoned decimal data with length len.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - No untranslatable characters

Return Code	Description
Other values	Offset of the first untranslated character plus one

Zoned Decimal to DOS Random (ZONTODOS)

Purpose

The ZONTODOS function converts zoned decimal data in EBCDIC format, pointed to by the parameter s, to zoned decimal data in ASCII format pointed to by the parameter t. The left half-byte (X'F') in the EBCDIC format is converted to X'3' in the left half-byte of ASCII format except for the last left half-byte (sign). The X'D' or X'B' of the sign (the last one) converts to X'B'; anything else converts to X'3'.

This function assumes that the left half-byte in the zoned decimal is X'F' (but does not include the last half-byte) and the right half-byte in the zoned decimal is from X'0' to X'9'. The length must be from 1 to 31 bytes.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target */
char far *s;  /* A pointer to the source */
int len;      /* Number of bytes         */
```

Declarations provided in include files:

```
unsigned pascal zontodos(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ZONTODOS(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Number of bytes }
```

Parameters

len	Number of bytes of the input zoned decimal data to be transformed.
s	A pointer to the zoned decimal data in EBCDIC format to be transformed.
t	Returns a pointer to the zoned decimal data with length len in ASCII format.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
Other values	Offset of the first untranslated character plus one

DOS Random to Zoned Decimal (DOSTOZON)

Purpose

The DOSTOZON function converts zoned decimal data in ASCII format, pointed to by the parameter s, to zoned decimal data in EBCDIC format pointed to by the parameter t. The left half-byte (X'3') in the ASCII format is converted to X'F' in the left half-byte of EBCDIC format, except for the last left half-byte (sign). The last left half-byte (sign) is not converted.

This function assumes that the left half-byte in the DOS random data is X'3' (but does not include the last half-byte which is X'3' or X'B') and the right half-byte in the DOS random data is from X'0' to X'9'. The length must be from 1 to 31 bytes.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target */
char far *s;  /* A pointer to the source */
int len;      /* Number of bytes         */
```

Declarations provided in include files:

```
unsigned pascal dostozon(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION DOSTOZON(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Number of bytes }
```

Parameters

len	Number of bytes of the input zoned decimal data to be transformed.
s	A pointer to the zoned decimal data in ASCII format to be transformed.
t	Returns a pointer to the zoned decimal data with length len in EBCDIC format.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
Other values	Offset of the first untranslated character plus one

Zoned Decimal to ASCII Numeric (ZONTOASC)

Purpose

The ZONTOASC function converts zoned decimal data, pointed to by the parameter s, to ASCII numeric data pointed to by the parameter t, with a decimal point at the position specified by decimal_position. The X'D' or X'B' of the sign half-byte converts to a negative number; anything else converts to a positive number in the ASCII numeric data.

This function assumes that the high byte in the zoned decimal is X'F' except for the last byte, the low byte in the zoned decimal is from X'0' to X'9', and the length is from 1 to 31 bytes. Also, 0 <= decimal_position < len.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;              /* A pointer to the target              */
char far *s;              /* A pointer to the source              */
int len;                  /* Bytes of the input zoned decimal area */
int decimal_position;     /* Place of decimal point               */
```

Declarations provided in include files:

```
unsigned pascal zontoasc(t,s,len,decimal_position);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ZONTOASC(t:ADSMEM,s:ADSMEM,len:INTEGER2,
                  decimal_position:INTEGER2) : WORD;
{char t              --  A pointer to the target
 char s              --  A pointer to the source
 int len             --  Bytes of the input zoned decimal data
 int decimal_position --  Place of decimal point }
```

Parameters

len	Number of bytes of the input zoned decimal data.
decimal_position	Place of the decimal point of the output ASCII numeric data.
s	A pointer to the zoned decimal to be transformed. The length of the zoned decimal must be greater than or equal to len.
t	Returns a pointer to the ASCII numeric data. If decimal_position is zero, the output data length is len+1. If decimal_position is not zero, the output data length is len+2.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
Other values	Offset of the first untranslated character plus one

ASCII Numeric to Zoned Decimal (ASCTOZON)

Purpose

The ASCTOZON function converts ASCII numeric data, pointed to by the parameter s, to zoned decimal data pointed to by the parameter t. Leading and trailing spaces are ignored. Spaces between the sign (+ or –) and the first digit are ignored.

This function assumes that the ASCII numeric data is from 1 to 33 bytes and 0 <= decimal_position <= len.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;              /* A pointer to the target                */
char far *s;              /* A pointer to the source                */
int len;                  /* Bytes of the output zoned decimal data */
int decimal_position;     /* Place of decimal point                 */
```

Declarations provided in include files:

```
unsigned pascal asctozon(t,s,len,decimal_position);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ASCTOZON(t:ADSMEM,s:ADSMEM,len:INTEGER2,
                  decimal_position:INTEGER2) : WORD;

{char t              --  A pointer to the target
 char s              --  A pointer to the source
 int len             --  Bytes of the output zoned decimal data
 int decimal_position --  Place of decimal point }
```

Parameters

len	Number of bytes of the output zoned decimal data to be transformed.
decimal_position	Place of the decimal point in the input ASCII data.
s	A pointer to the ASCII numeric data to be transformed. If decimal_position is zero, the data length must be greater than or equal to len + 1. If decimal_position is not zero, the data length must be greater than or equal to len + 2 and must contain a decimal point. All data must be digits (0 to 9), signs (+ or –), or blanks.
t	Returns a pointer to the zoned decimal with length len.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
X'0001'	DTX_OVERFLOW - Overflow error
Other values	Offset of the first untranslated character plus one

ASCII Characters to EBCDIC Characters (ASCTOEBC)

Purpose

The ASCTOEBC function converts all ASCII characters, pointed to by the parameter s, to EBCDIC characters pointed to by the parameter t, byte by byte. If t is NULL, the result is placed in the area pointed to by s. If len <= 0, the characters are translated until a NULL is encountered, or until the number of bytes specified in len are translated.

This function requires the PC Support router to be loaded so the API can access the ASCII-to-EBCDIC table.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target                    */
char far *s;  /* A pointer to the source                    */
int len;      /* Number of ASCII characters to be converted */
```

Declarations provided in include files:

```
unsigned pascal asctoebc(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION ASCTOEBC(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Number of ASCII characters to be converted }
```

Parameters

len — Number of bytes of the ASCII characters to be converted to EBCDIC characters.

s — A pointer to the ASCII characters to be converted.

t — Returns a pointer to the output EBCDIC characters with length len.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
X'FEFF'	DTX_ROUTER_NOT_LOADED - PC Support router is not loaded

EBCDIC Characters to ASCII Characters (EBCTOASC)

Purpose

The EBCTOASC function converts all EBCDIC characters, pointed to by the parameter s, to ASCII characters pointed to by the parameter t, byte by byte.

If t is NULL, the result is placed in the area pointed to by s. If len <= 0, the characters are translated until a NULL is encountered, or until len bytes are translated.

This function requires the PC Support router to be loaded so the API can access the EBCDIC-to-ASCII table.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target                    */
char far *s;  /* A pointer to the source                    */
int len;      /* Number of EBCDIC characters to be converted */
```

Declarations provided in include files:

```
unsigned pascal ebctoasc(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION EBCTOASC(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Number of EBCDIC characters to be converted }
```

Parameters

len — Number of bytes of the EBCDIC characters to be converted to ASCII characters.

s — A pointer to the EBCDIC characters to be converted.

t — Returns a pointer to the output ASCII characters with length len.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - Successful completion
X'FEFF'	DTX_ROUTER_NOT_LOADED - PC Support router is not loaded

EBCDIC Characters to EBCDIC Characters (EBCTOEBC)

Purpose

The EBCTOEBC function copies the EBCDIC string, pointed to by the parameter s, to the EBCDIC string pointed to by the parameter t, byte by byte. The conversion continues until a character outside the range from X'40' to X'FF' is encountered, or until the number of bytes specified by len are copied.

This function assumes that the length of the EBCDIC character string is from 1 to 4096 bytes.

Procedure Declaration

C

Statements to use in application programs:

```
#include <dtxlat.h>

char far *t;  /* A pointer to the target                  */
char far *s;  /* A pointer to the source                  */
int len;      /* Number of EBCDIC characters to be copied */
```

Declarations provided in include files:

```
unsigned pascal ebctoebc(t,s,len);
```

PASCAL

Statements to use in application programs:

```
{$include: 'dtxlat_t.inc'} { Constant section include file               }
{$include: 'dtxlat_c.inc'} { Type section include file                   }
{$include: 'dtxlat_e.inc'} { External procedure declarations include file }
```

Declarations provided in include files:

```
FUNCTION EBCTOEBC(t:ADSMEM,s:ADSMEM,len:INTEGER2) : WORD;
{char t  -- A pointer to the target
 char s  -- A pointer to the source
 int len -- Number of EBCDIC characters to be copied }
```

Parameters

len — Number of bytes of the input EBCDIC characters to be copied.

s — A pointer to the EBCDIC characters to be copied.

t — Returns a pointer to the EBCDIC characters with length len.

Return Codes

Return Code	Description
X'0000'	DTX_SUCCESS - No untranslatable characters
Other values	Offset of the first untranslated character plus one

Get EBCDIC-to-ASCII Table (GETE2A)

Purpose

Queries the PC Support router to get the EBCDIC-to-ASCII translation table.

Procedure Declaration

C

```
#include <DTXLAT.H>
char far * pascal gete2a(void)
```

PASCAL

```
{$include: 'DTXLAT_C.INC'} { Constant section include file               }
{$include: 'DTXLAT_T.INC'} { Type section include file                   }
{$include: 'DTXLAT_E.INC'} { External procedure declarations include file }
FUNCTION QRYE2A() : ADSMEM;
```

Parameters

Memory address pointer
: Returns a pointer to the EBCDIC-to-ASCII translation table.

Null pointer
: PC Support router is not loaded.

Get ASCII-to-EBCDIC Table (GETA2E)

Purpose

Queries the PC Support router to get the ASCII-to-EBCDIC translation table.

Procedure Declaration

C

```
#include <DTXLAT.H>
char far * pascal geta2e(void)
```

PASCAL

```
{$include: 'DTXLAT_C.INC'} { Constant section include file                }
{$include: 'DTXLAT_T.INC'} { Type section include file                    }
{$include: 'DTXLAT_E.INC'} { External procedure declarations include file }
FUNCTION QRYA2E( ) : ADSMEM;
```

Parameters

Memory address pointer
: Returns a pointer to the ASCII-to-EBCDIC translation table.

Null pointer
: PC Support router is not loaded.

Chapter 18. OS/2 Virtual Printer Application Program Interface

The OS/2 virtual printer API gives OS/2 applications access to AS/400 printers.

This chapter describes the individual routines, data structures, and return codes that make up the OS/2 virtual printer API. The descriptions refer to predefined constants, data structures, and function prototypes that can be found in the EHNVPOS2.H include file. This file is available in the PC Support tools folder (QIWSTOOL). For more information about the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

The virtual print APIs are the same for both the OS/2 and Windows environments. They have also been enabled for both SBCS and DBCS. Because of the differences between these environments, certain parameters or fields will not be used in some situations.

- A timeout value is not used in the OS/2 environment.
- Characters per line, lines per page, and ASCII character set are not used in the either DBCS environment.
- A target operating system is not used in either the OS/2 or Windows environments.

OS/2 Virtual Printer API Routines

The following discussions of each OS/2 virtual printer API routine describe in detail:

- Purpose
- Procedure Declaration
- Parameters
- Return codes

EHNVP_AssignVP

Purpose

This function assigns a network printer to an AS/400 system.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_AssignVP(HWND hWnd,
        LPVP_ASSIGN lpAssignData,
        char far *  lpszLocalName);
```

Parameters

hWnd	Identifies the current window of the application.
lpAssignData	Points to a VP_ASSIGN structure. This structure specifies the options to use when assigning the virtual printer. It is also used to return the ID associated with the assigned printer. For a description of the VP_ASSIGN structure, see "VP_ASSIGN" on page 18-15.
lpszLocalName	Points to a null-terminated character string that identifies the local device to be redirected. Valid local device names are LPT1 through LPT9.

Notes:

1. Up to nine virtual printers may be assigned at the same time.
2. If szPrtFile is specified, the szPrtFileLib field identifies the library where the printer file exists. If the library is not specified, *USERLIBL is used.
3. If the System Name field of the Device Data field is null, the default system is used.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

EHNVP_BuildList

Purpose

This function builds one of the following lists:

- Printers on an AS/400 system
- All print files (or all print files matching a specified wildcard value) in the libraries of the library list of the caller
- All print files (or all print files matching a specified wildcard value) in a specified library
- All libraries in the user library list (or matching a specified wildcard value)

This function must be used before the EHNVP_QueryListHead and EHNVP_GetListItem functions can be used.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_BuildList(
        HWND           hWnd,
        unsigned short wListType,
        LPVP_BUILDLIST lpListParms);
```

Parameters

hWnd — Identifies the current window of the application.

wListType — Specifies the type of list request. Possible values are:

- EHNVP_LTPRTLIST (X'0000') printer list.
- EHNVP_LTLIBLIST (X'0001') library list.
- EHNVP_LTPRTFLIST (X'0002') printer files list.

lpListParms — Points to a VP_BUILDLIST structure. This structure specifies the options to use when building the list. It is also used to return the information about the generated list. For a description of the VP_BUILDLIST structure see "VP_BUILDLIST" on page 18-17.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

EHNVP_ChgTransTable

Purpose

The function changes the translation table used by a virtual printer.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_ChgTransTable(HWND           hWnd,
                                      unsigned short wVPid,
                                      char far       *lpcTransTable);
```

Parameters

hWnd — Identifies the current window of the application.

wVPid — The ID for the virtual printer to be changed.

lpcTransTable — A far pointer to a 256 byte translation table used for converting ASCII to EBCDIC.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

EHNVP_DosAPI

Purpose

This function will handle calls written to the DOS Extended Virtual print API.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_DosAPI(HWND           hWnd,
                               LPVP_DOSAPIREQ lpDosApiReq);
```

Parameters

hWnd — Identifies the current window of the application.

lpDosApiReq — A far pointer to a request block for the extended DOS virtual print API. This is the data that would be referred to by the EX:BX registers if the call was being made through the shared interrupt under DOS.

Return Codes

The return value will be from the DOS low-level API.

Additional Information

This API should be used when an application is run under both DOS and OS/2 operating systems. The call will be made at the same place as the code to generate the shared interrupt under DOS.

See Chapter 8, "Virtual Printer Function Low-Level Application Program Interface" for additional information on the DOS low-level virtual print APIs.

EHNVP_GetErrorText

Purpose

This function retrieves any error text from virtual printer network errors and clears the error text buffer of the DLL. The error for which the text is retrieved is one of the following:

- The last virtual print API request that had a return code of EHNVP_NETERROR (X'0007')
- The last EHNVP_GetAsnStat request that returned a status with the EHNVP_ERRPENDING bit turned on

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_GetErrorText(
        HWND       hWnd,
        char far * lpszErrBuffer);
```

Parameters

hWnd — Identifies the current window of the application.

lpszErrBuffer — Points to a character buffer that is used to store the retrieved error text. This buffer must be 514 bytes in length. The error text is stored in this buffer as a null-terminated character string.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

EHNVP_GetListItem

Purpose

This function retrieves the next item in a list built by the EHNVP_BuildList API.

Note: EHNVP_BuildList must be called before calling this API.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_GetListItem(
        HWND            hWnd,
        unsigned short  wListID,
        char far *      lpszItemBuf);
```

Parameters

hWnd — Identifies the current window of the application.

wListID — Specifies the list ID returned in the wListID field of the VP_BUILDLIST structure from the call to EHNVP_BuildList.

lpszItemBuffer — Points to a buffer where the list item is to be stored. The buffer must be at least as long as the usRecLength parameter in bytes (returned in the VP_BUILDLIST structure on the EHNVP_BuildList call). The list item is stored in the buffer as a null-terminated character string. The characters up to the first blank contain the name of the printer, print file, or printer file library. Any additional characters provide more information about the item.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22. If the return value is EHNVP_BADLISTID (X'0013'), the ID is not valid or the list has been damaged.

EHNVP_GetRedirList

Purpose

This function, when called repeatedly, returns status information for a list of AS/400 network printers. The network printer is identified by the value specified for the wIndex parameter. If an entry in the current list of redirected printers is found, the current status of the AS/400 network printer and its assigned parameter values are returned.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_GetRedirList(
        HWND           hWnd,
        unsigned short wIndex,
        LPVP_REDIRINFO lpRedirItem);
```

Parameters

hWnd — Identifies the current window of the application.

wIndex — Specifies the list position of the redirected printer device for which status information is needed. A value of 0 indicates the first device, a value of 1 indicates the second device, and so on.

lpRedirItem — Points to a VP_REDIRINFO structure. This structure is used to return information about the network printer. For more information about the VP_REDIRINFO structure, see "VP_REDIRINFO" on page 18-18.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22. This function returns EHNVP_ENDOFLIST when there are no more redirected devices.

EHNVP_QueryAsnParms

Purpose

This function returns detailed status information regarding the network printer assignment for the specified printer ID.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_QueryAsnParms(
        HWND           hWnd,
        unsigned short wVPid,
        LPVP_ASNSTATUS lpStatusData);
```

Parameters

hWnd — Identifies the current window of the application.

lpwVPID — Specifies the ID of the virtual printer to query. This is the ID returned when the printer was assigned. This value can also be obtained by calling EHNVP_QryAsnStatus or EHNVP_GetRedirList.

lpStatusData — Points to a VP_ASNSTATUS structure. This structure is used to return the status of the network printer. For a description of the VP_ASNSTATUS structure, see "VP_ASNSTATUS" on page 18-15.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

EHNVP_QueryAsnStatus

Purpose

This function returns the ID and current status of the specified printer (if a device is assigned as a virtual printer).

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_QueryAsnStatus(
        HWND                  hWnd,
        char far *            lpszLocalDevice,
        unsigned short far *  lpwVPid,
        unsigned short far *  lpwVPStatus);
```

Parameters

hWnd — Identifies the current window of the application.

lpszLocalDevice — Points to a null-terminated character string that identifies the local device being queried. Valid local device names are LPT1 through LPT9.

lpwVPID — Points to an integer variable that is used to return the virtual printer ID. This ID can be used as input to the EHNVP_QueryAsnParms call to get the full printer assignment information.

lpwVPStatus — Points to an integer variable where the status of the local device is to be stored. The following bits may be turned on:

EHNVP_ASSIGNED (X'0001')
EHNVP_ERRPENDING (X'0004')
EHNVP_SUSPENDED (X'0008')
EHNVP_DATAPASS (X'0010')

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

EHNVP_QueryCapability

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_QueryCapability(HWND hWnd );
```

Parameters

hWnd Identifies the current window of the application.

Return Codes

The return value is the functional level of the OS/2 virtual printer API. EHNVP_CAPLVL1 is defined in EHNVPOS2.H. The original version of this DLL always returns a value of EHNVP_CAPLVL1 (X'0000') on this call.

EHNVP_QueryErrorText

Purpose

This function retrieves error text for virtual printer network errors. The error for which the text is retrieved is one of the following:

- The last virtual printer API request had a return code of EHNVP_NETERROR (X'0007').
- The last EHNVP_GetAsnStat request that returned a status with the EHNVP_ERRPENDING bit turned on.

The error message buffer is not cleared by this call. If you call this function twice, the same message text is returned both times.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_QueryErrorText(
        HWND       hWnd,
        char far * lpszErrBuffer);
```

Parameters

hWnd Identifies the current window of the application.

lpszErrBuffer Points to a character buffer that is used to store the retrieved error text. This buffer must be 514 bytes in length. The error text is stored in this buffer as a null-terminated character string.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

EHNVP_QueryListHead

Purpose

This function retrieves the heading of a list built by the EHNVP_BuildList function.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_QueryListHead(
        HWND           hWnd,
        unsigned short wListID,
        char far *     lpszHeadBuf);
```

Parameters

hWnd	Identifies the current window of the application.
wListID	Specifies the list ID returned in the wListID field of the VP_BUILDLIST structure from the call to EHNVP_BuildList.
lpszHeadBuffer	Points to a buffer where the heading of the list is to be stored. This buffer must be at least as long as the usRecLength parameter in bytes (returned by EHNVP_BUILDLIST structure on the EHNVP_BuildList call). The heading is stored in the buffer as a null-terminated character string. This heading contains a text description of the content of each item in the list. A programmer may choose to use this for display purposes.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22. If the return value is EHNVP_BADLISTID, the ID is not valid or the list has been damaged.

EHNVP_QueryVPStatus

Purpose

This function returns the status of the virtual printer program.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_QueryVPStatus(
        HWND          hWnd,
        LPVP_VPSTATUS lpVPStatus);
```

Parameters

hWnd Identifies the current window of the application.

lpVPStatus Points to a VP_VPSTATUS structure. This structure is used to store the status of the virtual printer program. For a description of the VP_VPSTATUS structure, see "VP_VPSTATUS" on page 18-19.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

EHNVP_ReleaseVP

Purpose

This function releases an assigned AS/400 network printer.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_ReleaseVP(
        HWND           hWnd,
        unsigned short wVPid);
```

Parameters

hWnd Identifies the current window of the application.

wVPID Specifies the ID of the virtual printer to be released. This is the ID returned when the printer was assigned. This value can also be obtained by calling EHNVP_QueryAsnStatus or EHNVP_GetRedirList.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

EHNVP_ResetParms

Purpose

This function resets some of the parameters for a currently assigned printer.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_ResetParms(
        HWND           hWnd,
        unsigned short wVPid,
        LPVP_ASNPARMS  lpNewParms);
```

Parameters

hWnd — Identifies the current window of the application.

wVPID — Specifies the ID of the virtual printer for which the parameters are to be reset. This is the ID returned when the printer was assigned. This value can also be obtained by calling EHNVP_GetRedirList.

lpNewParms — Points to a VP_ASNPARMS structure. This structure is used to specify the new parameter values. For a description of the VP_ASNPARMS structure, see "VP_ASNPARMS" on page 18-12.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

Additional Information

The following fields may be changed by this API:

- usCopies
- fDeferPrint
- fAppFormData
- usACharSet
- cUntransChar
- usTOS (DBCS environment only)
- usFormFeed

EHNVP_VerifyAsnDevice

Purpose

This function verifies that a printer, printer file library, and printer file are valid. If the objects are valid, the function returns information about them.

Procedure Declaration

```
#include "EHNVPOS2.H"

USHORT FAR PASCAL EHNVP_VerifyAsnDevice(HWND hWnd,
        LPVP_VERIFYREQ lpDeviceData,
        LPVP_VERIFYOUT lpVerifyOutput);
```

Parameters

hWnd — Identifies the current window of the application.

lpDeviceData — Points to a VP_VERIFYREQ structure. This structure is used to identify the AS/400 system name, printer, library and printer file to verify. For a description of the VP_VERIFYREQ structure, see "VP_VERIFYREQ" on page 18-19.

Notes:

1. If szPrtFile is specified, the szPrtFileLib field identifies the library where the printer file exists. If the library is not specified, *USERLIBL is used.
2. If the szSystem field is null, the default system is used.
3. On input, the szOutQue and szOutQueLib parameters are reserved and should be set to NULL.

lpVerifyOutput Points to a VP_VERIFYOUT structure. This structure is to be used to store the printer type default, and possible values for assigned options, if the specified objects are valid. For a description of the VP_VERIFYOUT structure, see "VP_VERIFYOUT" on page 18-20.

Notes:

1. Both the szOutQue and szOutQueLib fields are updated with the output queue and output queue library to which the data will be spooled on the AS/400 system. These values should be passed back on an Assign request.
2. The output queue name and the output queue library name are not provided if you are attempting to assign a virtual printer to an AS/400 system running in a version or release of the OS/400 program before Version 2 Release 2.

Return Codes

For return codes, see "Return Codes for the OS/2 Virtual Printer API" on page 18-22.

OS/2 Virtual Printer API Structures

VP_ASNPARMS

Description

This structure is used by the EHNVP_ResetParms and EHNVP_AssignVP API routines. The structure is defined as follows:

```
typedef struct _VP_ASNPARMS {
   unsigned short  usSize;        //(in)  - length in bytes
   char szPrinterType[EHNVP_PRTTYPELEN + 1];  // Printer type
   unsigned short  usDataType;    //Data type
   unsigned short  usCopies;      //number of Copies
   unsigned short  usTimeout;     //printer time out in seconds
   BOOL            fDeferPrint;   //defer printing
   unsigned short  usCPI;         //default CPI*10
   unsigned short  usCPL;         //default Chars Per Line
   unsigned short  usLPI;         //default LPI*10
   unsigned short  usPageLen;     //default Page Length
   unsigned short  usLPP;         //default LinesPerPage
   BOOL            fAppFormData;  //Application formatted data
   unsigned short  usACharSet;    //ASCII Character Set
   BYTE            bUntransChar;  //Untranslatable Character
   char            achReserved[4];
   unsigned short  usTOS;         //Target Operating System
   unsigned short  usFormFeed;    //Append Form Feed
   } VP_ASNPARMS;
```

Field	Description
usSize	Identifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
szPrinterType	Specifies the printer type (for example, 3812). This is a null-terminated character string.
usDataType	Identifies the data type the virtual printer will be handling. Possible values are: EHNVP_DTSCS (X'0001') EHNVP_DTCONVRT (X'0002') EHNVP_DTFFT (X'0003') EHNVP_DTASCII (X'0004') EHNVP_DTAFPDS (X'0005')
usCopies	Identifies the number of copies requested. Possible values are 1 to 255. If the usDataType field is EHNVP_DFTFT, the allowed values are 1 to 99.
usTimeOut	Identifies the printer time-out value associated with the printer. The time-out value is the number of seconds the virtual printer is to wait after it stops receiving data and before it closes the output file. A value of zero (0) indicates no time-out. Any other value represents the value in seconds. Possible values are 0 to 255.
fDeferPrint	Indicates whether the output file on the AS/400 system must be closed before printing can start. Possible values are True and False.
usCPI	Identifies the number of characters per inch multiplied by 10 (for example, 16.7 would be 167). The possible values are returned on the EHNVP_VerifyAsnDevice API routine.
usCPL	Identifies the number of characters per line. Possible values are 1 to 378.
usLPI	Identifies the number of lines per inch to print. Possible values are returned by the EHNVP_VerifyAsnDevice API routine.

usPageLen — Identifies the length of the paper in lines. Possible values are 1 to 255.

usLPP — Identifies the number of lines printed per page. Possible values are 1 to the value specified for the usPageLen parameter.

fAppFormData — Identifies whether the data was formatted by an application. Possible values are True or False. If True, the virtual printer function assumes the formatting is correct and allows the following values to be changed by the data stream:

- characters per inch (CPI)
- lines per inch (LPI)
- lines per page (LPP)
- Page Length

If False, the virtual printer function ignores any attempts to change these values.

usACharSet — Identifies the ASCII character set to use. Character set 2 has printable characters at code points from X'80' to X'9F'. Character set 1 has printer controls in this range. Possible values are:

EHNVP_CHARSET1 (X'0001')
EHNVP_CHARSET2 (X'0002')

bUntransChar — Identifies the untranslatable character substitute character. This character is substituted for characters that are not found in the ASCII-to-EBCDIC table.

usTOS — Indicates the target operating system on which your printer emulator runs when you print personal computer code data. The possible values are:

EHNVP_TOSDOS (X'0001')
EHNVP_TOSOS2 (X'0002')
EHNVP_TOSDOSV (X'0003')

Note: This field is ignored in SBCS data.

usFormFeed — Identifies whether the virtual printer should add a form feed to the end of each job. This field is used only when you are also using data type 4 for SBCS data or data type 3 for DBCS data. The possible values are:

EHNVP_FFYES (X'0001')
EHNVP_FFNO (X'0002')
EHNVP_FFCON (X'0003')

Note: If the value is ENHVP_FFCON, the virtual printer adds a form feed to the end of a job only if the last byte of the job is not a form feed character.

Notes:

1. For the EHNVP_AssignVP API, all of these fields should be filled in by the caller.
2. For the EHNVP_ResetParms API, the usSize, fDeferPrint, usTimeOut, and usCopies fields should be filled in by the caller. All other fields are ignored.

VP_ASNSTATUS

Description

This structure is used by the EHNVP_QueryAsnParms API routine. The structure is defined as follows:

```
typedef struct _VP_ASNSTATUS {
   unsigned short usSize;
   WORD           wAnsStatus;
   VP_ASSIGN    AssignInfo;
   VP_PRTSTAT   PrintStats;
} VP_ASNSTATUS;
```

Field	Description
usSize	Identifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
wAnsStatus	Returns flags that indicate the assign status for this virtual printer. The following bits may be turned on: EHNVP_ASSIGNED (X'0001') EHNVP_ERRPENDING (X'0004') EHNVP_SUSPENDED (X'0008') EHNVP_DATAPASS (X'0010') If the EHNVP_ASSIGNED bit is off, none of the other bits are valid.
AssignInfo	Returns the assignment information.
PrintStats	Returns the printer statistics.

VP_ASSIGN

Description

This structure is used by the EHNVP_AssignVP and EHNVP_QueryAsnParms API routines.

```
typedef struct _VP_ASSIGN {
   unsigned short      usSize;
   WORD                wVPID;
   VP_VERIFYREQ        DeviceData;
   VP_ASNPARMS         AssignParms;
} VP_ASSIGN;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
wVPID	Specifies the ID of the virtual printer. This field is filled in by the virtual print program when the device is assigned successfully.
DeviceData	Contains information on where the local port is to be redirected.

AssignParms This structure contains the assign parameters. Some of the parameters are ignored based on the value given for the usDataType field.

The following table shows which parameters must be specified when assigning a virtual printer for each of the possible data types (SCS, convert ASCII to SCS, final form text, ASCII, and advanced function printing data stream).

Table 18-1. Assign Parameters Used with Data Types

Parameter	Data Type 1	Data Type 2	Data Type 3	Data Type 4	Data Type 5
usTimeOut	X	X	X	X	X
usCopies	X	X	X	X	X
fDeferprint	X	X		X	X
usCPI	X	X			
usCPL	X	X			
usLPI	X	X			
usPageLen	X	X			
usLPP	X	X			
fAppFormData		X			
usACharSet		X			
cUntransChar		X			

Notes:

1. The usSize field for the AssignParms parameter must be filled in by the caller with the size of the entire VP_ASNPARMS structure in bytes, regardless of the data type.
2. For the EHNVP_AssignVP API, the wVPID field is filled in as a result of the API call. All other fields should be filled in by the caller.
3. For the EHNVP_QueryAsnParms API, the usSize field should be filled in by the caller. All other fields are filled in as a result of the API call.

VP_BUILDLIST

Description

This structure is used by the EHNVP_BuildList API routine. The structure is defined as follows:

```
typedef struct _VP_BUILDLIST  {
   unsigned short   usSize;
   WORD             wListID;
   unsigned short   usRecLength;
   char szSystem[EHNCL_L_SYSNAME + 1];
   char szPrtFileLib[EHNCL_L_LIBNAME + 1];
   char szPrtFile[EHNCL_L_FILENAME + 1];
} VP_BUILDLIST;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
wListID	Is used to return an ID of the list that is needed for the EHNVP_QueryListHead and EHNVP_GetListItem APIs.
usRecLength	Is used to return the maximum length of records in the list. This is the size of the buffer needed when calling EHNVP_QueryListHead and EHNVP_GetListItem.
szSystem	Specifies a null-terminated AS/400 system name. If the name has a length of zero, the default system is used.
szPrtFileLib	Specifies the name of the library to search for printer files. This field is ignored if the list type is not EHNVP_LTPRTFLIST or EHNVP_LTLIBLIST.
szPrtFile	Specifies a generic printer file name used to filter which printer files should be put on the list. This field is ignored if the list type is not EHNVP_LTPRTFLIST.

VP_DOSAPIREQ

Description

This structure is used by the EHNVP_DosAPI routine. It will point to the same information referred to by the ES:BS registers in the virtual print API for extended DOS.

```
typedef struct _VP_DOSAPIREQ  {
   unsigned short   usSize;
   WORD             wReqType;
   char far         *lpszErrText;
   void far         *lpReqStruct;
} VP_DOSAPIREQ;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.

wReqType	Specifies the API being requested. See Chapter 8, "Virtual Printer Function Low-Level Application Program Interface" for a list of possible values.
lpszErrText	Specifies a far pointer, in selector:offset format, to a 513 byte buffer that will be used to hold the error text for the request.
lpReqStruct	Specifies a far pointer, in selector:offset format, to the DOS API request structure. The content of the structure is dependent on the value in the wReqType field and is defined in Chapter 8, "Virtual Printer Function Low-Level Application Program Interface."

VP_PRTSTAT

Description

This structure is used by the EHNVP_QueryAsnParms API routine. The structure is defined as follows:

```
typedef struct _VP_PRTSTAT  {
   unsigned short   usSize;
   unsigned short   usPrintedFiles;
   unsigned short   usPages;
   unsigned short   usUntransChars;
} VP_PRTSTAT;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
usPrintedFiles	Returns the number of files that have been printed through this virtual printer since it was assigned.
usPages	Returns the number of pages in the last job.
usUntransChars	Returns the number of untranslatable characters in the last job.

VP_REDIRINFO

Description

This structure is used by the EHNVP_GetRedirList API routine. The structure is defined as follows:

```
typedef struct _VP_REDIRINFO  {
   unsigned short usSize;
   WORD           wVPid;
   WORD           wAnsStatus;
   char szLocalName[EHNVP_MAXLOCNAME + 1];
   VP_VERIFYREQ DeviceData;
} VP_REDIRINFO;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.

wVPID	Returns the ID of the local printer to be used with other API calls.
wAnsStatus	Returns flags indicating the current status of the device. Possible values can be one or more of the following: • EHNVP_ASSIGNED (X'0001') • EHNVP_ERRPENDING (X'0004') • EHNVP_SUSPENDED (X'0008') • EHNVP_DATAPASS (X'0010')
szLocalName	Returns the local device that is redirected. The name is returned as a null-terminated character string.
DeviceData	Is used to return the remote device to which the local device is redirected.

VP_VPSTATUS

Description

This structure is used by the EHNVP_QueryVPStatus API routine. The structure is defined as follows:

```
typedef struct _VP_VPSTATUS {
   unsigned short usSize;
   unsigned short usNumVPs;
   unsigned short usAPILevel;
   char           achVerRelLev[EHNVP_VRLsize];
}  VP_VPSTATUS;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
usNumVPs	Returns the number of assigned virtual printers.
usAPILevel	Returns the level of support for the API. For Version 2 Release 2, this is 1.
achVerRelLev	Returns the 6 ASCII characters that identify the version, release, and level of the virtual print Windows API program.

VP_VERIFYREQ

Description

This structure is used by the EHNVP_GetRedirList, EHNVP_AssignVP, and EHNVP_VerifyAsnDevice API routines. The structure is defined as follows:

```
typedef struct _VP_VERIFYREQ {
   unsigned short usSize;
   char szSystem[EHNCL_L_SYSNAME + 1];
   char szPrinter[EHNCL_L_PRTNAME + 1];
   char szPrtFileLib[EHNCL_L_LIBNAME + 1];
   char szPrtFile[EHNCL_L_FILENAME + 1];
   char szOutQueLib[EHNCL_L_LIBNAME + 1];
   char szOutQue[EHNCL_L_QUENAME + 1];
} VP_VERIFYREQ;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
szSystem	Identifies the name of the AS/400 system. This name is a null-terminated character string.
szPrinter	Identifies the name of the AS/400 printer device. This name is a null-terminated character string.
szPrtFileLib	Identifies the name of the library for the szPrtFile field. This name is a null-terminated character string.
szPrtFile	Identifies the name of the printer file to use for printing. This name is a null-terminated character string.
szOutQuelib	Identifies the name of the library containing the output queue identified in the szOutQue field. This name is a null-terminated character string.
szOutQue	Identifies the name of the output queue on the AS/400 system where data is being sent. This name is a null-terminated character string.

Notes:

1. For the EHNVP_AssignVP API, all of these fields should be filled in by the caller.
2. For the EHNVP_GetRedirList API, the usSize field should be filled in by the caller. All other fields are filled in as a result of the API call.

VP_VERIFYOUT

Description

This structure is used by the EHNVP_VerifyAsnDevice API routine. The structure is defined as follows:

```
typedef struct _VP_VERIFYOUT {
   unsigned short  usSize;              //(in)  - length in bytes
   VP_VERIFYREQ    DeviceData;          //actual data
   unsigned short  usDefCPI;            //default CPI
   unsigned short  usDefCPL;            //default page width
   unsigned short  usDefLPI;            //default LPI
   unsigned short  usDefPageLen;        //default paper length
   unsigned short  usDefLPP;            //default lines per page
   unsigned short  usDefCopies;         //default copies
   BOOL            fDefDefer;           //default defer value
   unsigned short  usDefTimeOut;        //default time out
   BOOL            fDefAppFormData;     //default application data
   unsigned short  usDefACharSet;       //default ASCII Character Set
   unsigned short  ausValidCPIs-EHNVP_MAXCPIVAL-;
   unsigned short  ausValidLPIs-EHNVP_MAXLPIVAL-;
   unsigned short  ausValidDataTypes-EHNVP_MAXDATATYPE-;
   char szPrinterType-EHNVP_PRTTYPELEN + 1-;  // Printer type
   BYTE            bDefUntransChar;     //default untranslatable character
   unsigned short  usDefTOS;            //default Target Operating System
   unsigned short  usDefFormFeed;       //default Append Form Feed
} VP_VERIFYOUT;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
DeviceData	Returns the actual names of the system, printer file library, and printer file.
usDefCPI	Returns the default characters per inch multiplied by 10.
usDefCPL	Returns the default for the number of characters that are printed on a line.
usDefLPI	Returns the default for the number of lines per inch multiplied by 10.
usDefPageLen	Returns the default for the paper length in lines.
usDefLPP	Returns the default for the number of lines that are printed on a page.
usDefCopies	Returns the default for the number of copies to be made for each job.
fDefDefer	Returns the default for whether the job should not be printed on the AS/400 system until the output file is closed. A nonzero value indicates that the job should be deferred.
usDefTimeOut	Contains the default timeout value.
fDefAppFormData	Returns the default value for the fAppFormData field.
usDefACharSet	Returns the default value for the usACharSet field.
ausValidCPIs	Returns the valid characters per inch (CPI) values multiplied by 10. If there are less valid CPI values than the array has room for, the extra elements of the array will contain 0's.
ausValidLPIs	Returns the valid lines per inch (LPI) values multiplied by 10. If there are less valid LPI values than the array has room for, the extra elements of the array will contain 0s.

ausValidDataTypes	Returns the valid data type values. Each element in the array may be: • 0 (indicating an unused element) • EHNVP_DTSCS (X'0001') SCS • EHNVP_DTCONVRT X'0002') ASCII to SCS • EHNVP_DTFFT (X'0003') final form text • EHNVP_DTASCII (X'0004') ASCII • EHNVP_DTAFPDS (X'0005') advanced function printing data stream
szPrinterType	Returns the printer type (for example, 3812). This is returned as a null-terminated character string.
bDefUntransChar	Returns the default value for the bUntransChar field.
usDefTOS	Returns the default value for the usTOS.
usDefFormFeed	Returns the default value for the usFormFeed field.

Return Codes for the OS/2 Virtual Printer API

Functions in the virtual printer Windows API use the following return codes. These are constants defined in EHNVPOS2.H.

Table 18-2. Return Codes for the OS/2 virtual printer API

Name	Hex Value	Description
EHNVP_SUCCESS	0000	Command completed successfully
EHNVP_INVALIDPARM	0001	Invalid parameter
EHNVP_VPNOTSTARTED	0002	Virtual printer not started
EHNVP_VPBACKLEVEL	0003	API not available
EHNVP_NOERRORTEXT	0004	No error text available
EHNVP_NOTREDIR	0005	Device not redirected
EHNVP_ALREADYREDIR	0006	Device already redirected
EHNVP_NETERROR	0007	Error from virtual printer
EHNVP_ROUTERERROR	0008	Error with router DLL
EHNVP_FUNCTIONBUSY	0009	Function busy
EHNVP_ENDOFLIST	0012	End of the list
EHNVP_BADLISTID	0013	Corrupted list
EHNVP_BADVPID	0014	Virtual printer ID is not valid
EHNVP_BADLOCALNAME	0015	Local name is not valid
EHNVP_BADUSsize	0016	usSize field is too small
EHNVP_WINDOWSERROR	0100	Windows error
EHNVP_MEMERROR	0101	General memory error
EHNVP_DOSMEMERROR	0102	Memory error below 640K
EHNVP_BADENVIRONMENT	0103	Invalid Windows environment
EHNVP_UNEXPECTEDNETERROR	FFFF	Internal error

Part 3. PC Support Windows Application Program Interfaces

Chapter 19. PC Support/400 Windows Application Program Interface Overview

The PC Support Windows APIs offer a range of interfaces for AS/400 client applications as well as client/server applications. These interfaces provide access to the following AS/400 services:

Command
Database
Display
File
Print
Program to program
Translation

Command services enable work stations to run batch commands on the AS/400 system. These services are provided by the remote command function.

Database and file services enable work stations to retrieve and update data that reside on the AS/400 system. These services are provided by the shared folders function, transfer function, and remote SQL.

Display services enable applications to monitor screen output and send key inputs from and to an AS/400 application. The display services are provided by the RUMBA/400 function. RUMBA/400 provides both a Windows DDE interface and an EHLLAPI (Emulator High-Level Language API) interface that can be accessed from Windows applications. The RUMBA/400 tools for DDE and EHLLAPI can be purchased from Wall Data Incorporated. For additional information, call 1-800-48-RUMBA (U.S. and Canada only). These interfaces are not discussed in this manual.

Print services enable work stations to print to AS/400 printers. These are provided by the virtual print function.

Program-to-program services enable applications on the work station to communicate with AS/400 applications as well as applications running on other work stations. Program-to-program services are provided by the router (APPC) function and the data queues function.

Translation services provide a set of APIs that translate from PC to AS/400 formats and from AS/400 to PC formats. These services are provided by the data translation function.

Note: Except for program-to-program services, AS/400 programs do not need to be written for a user application to access the AS/400 system.

Command Services

The following sections describe the PC Support Windows APIs in more detail to give a better understanding of which APIs can be used to accomplish various tasks.

Submit Remote Command

Submit remote command API allows work stations to start noninteractive programs and procedures on the AS/400 system and to receive completion messages. Any AS/400 batch command or program may be run. Up to 10 reply messages can be sent by the AS/400 command or program per command request. However, program data cannot be returned through this interface. Router (APPC) or data queues should be used if program data need to be returned.

Database Services

Transfer Function

Transfer function can be used to retrieve specific records from AS/400 databases using AS/400 Query. It uses an SQL-like statement to describe the desired query. The performance of this API is affected by the complexity of the query, although in general it is faster than Remote SQL. Unlike Remote SQL, however, records cannot be updated individually. Updates are performed to the entire database file. Transfer function also provides a list capability for libraries, files and members. This capability is not available through Remote SQL. Applications that only require query or list capability or that require favorable download performance should use this API.

Remote SQL

Remote SQL provides full SQL capability including record update and insert and commit support. This API uses AS/400 Dynamic SQL and performance is equivalent to Dynamic SQL. Generally, performance is not as good as transfer function. Applications that need to perform record updates or that need full function SQL should use this API. Unlike transfer function, Remote SQL does not have the capability to retrieve a list of files or libraries from the AS/400 system. If this capability is needed, both APIs can be used simultaneously.

For a more detailed discussion on SQL, refer to the *SQL/400* Programmer's Guide* or the *SQL/400* Reference*.

File Services

Shared Folders

Shared folders offers PC applications access to AS/400 folders as network drives. Applications written for standard DOS services can store and retrieve data without modifications. Shared folders offers these applications the increased capacity, security, and sharing capability of the AS/400 system. The data stored in a shared folder need to be extracted and translated before being used by an AS/400 application. To share data between AS/400 and PC applications, the Hierarchical File System API on the AS/400 system can be used. For more information on the Hierarchical File System API, refer to the *System Programmer's Interface Reference*.

In addition to the standard APIs supplied by DOS, shared folders also provides a set of routines to help manage assigning and releasing shared folders drives. The shared folders API allows up to 26 drive assignments and does not restrict the drive letter range (A to Z) in any way. It is specific to shared folders and, therefore, does not interfere with the assignments of other network drives. A programmer can

use this API to hide the details of whether data is being accessed from a local drive or from a shared folder drive.

A set of network redirector APIs is also provided for Windows programmers. This set of APIs may be used with shared folders and any other network program that supports these functions to assign (redirect) and release (cancel redirection) drives. Therefore, a programmer can use this API to write a generic application to access both shared folders drives and drives connected to other networks.

For a more detailed discussion on the DOS network redirection functions, refer to the *DOS Technical Reference*.

Transfer Function

Transfer function is primarily a database access API, although it can be used for file transfer. Data transferred through transfer function can be stored in AS/400 format on the AS/400 system or in PC format on the personal computer. All data translations can be performed by the transfer function or the application program can do the translations using the translation services.

Print Services

Virtual Print

Virtual print offers PC applications access to AS/400 printers as network printers. Applications written to the Windows GDI (Graphics Device Interface) can print to an AS/400 printer or to PC printers attached to an AS/400 system without modifications. For more information about virtual printers, see the *PC Support/400: DOS User's Guide* and the *PC Support/400 DOS Installation and Administration Guide*.

Note: The supported data stream depends on the printers.

In addition to the Windows GDI, virtual print also provides a set of APIs. Virtual printers may be assigned and released using the virtual print API. The printer setup may also be changed, after assign time, to be more compatible with different applications being run.

Up to nine printers (LPT1 through LPT9) may be supported through the virtual print API instead of three supported through standard DOS services.

Like shared folders, virtual print also supports the network redirector interface. This enables an application to be written for multiple networks using a single set of interfaces. However, printer setup cannot be modified through the redirector interface. Also, only a subset of printer redirector services are supported by virtual print.

For a more detailed discussion on the DOS network redirection functions, refer to the *DOS Technical Reference*

Program-to-Program Services

Router (APPC)

The router API provides a method to write cooperative processing applications between PCs and AS/400 systems. The router is an implementation of the SNA LU 6.2 protocol, however, the API insulates the programmer from low-level communications programming and hardware connectivity types. Since the application pro-

grammers need to write both the AS/400 and PC programs when using the router API, the router API offers the maximum flexibility. Almost anything that can be accessed by the host application can be extended to the partner PC application. The router API also offers the best performance among the APIs. However, since the router API is a communications API, a certain degree of complexity is involved. Applications that need services that are not provided by other PC Support APIs and that are performance critical should definitely consider this set of APIs.

Data Queues

The data queues API allows AS/400 applications and PC applications to pass information to each other through a data queue, which resides on the AS/400 system. The format of the data passed in a data queue is defined by the server and client applications and is not restricted in any way. Since data queues is the method used by AS/400 applications to exchange data, it is one of the easiest interfaces for AS/400 applications.

Data queues is asynchronous. Requesters can place information (or work to be done) on a receiver's queue without the receiving application being active. The receiving application can be started later (after hours) if desired.

Data queues is also an easy way for work stations to communicate with each other. Since a data queue is just a queue that resides on the AS/400 system, work stations can put and get data through shared queues.

PC applications exchanging data with AS/400 applications may want to use the translation services to get the data into AS/400 formats when sending data, and into PC formats when receiving data.

Client/server applications that can place requests and responses in queues are natural applications for data queues. Applications that require synchronization may want to use the router API instead.

Translation Services

Data Translation

Because an AS/400 system stores data in different formats than personal computers, translation services may be needed when sending and receiving data to and from the AS/400 system. Translation services are provided for text and numerous numeric formats.

PC Support/400 DLLs

The PC Support Windows Application Program Interfaces (APIs) are implemented as a set of dynamic link libraries (DLLs). A DLL is a set of routines that are bound to the application at run time rather than at link time. Several applications can use the routines simultaneously, yet only one copy of the routine is loaded into memory.

The PC Support Windows DLLs are:

- EHNAPPC (router)
- EHNDQW (data queues)
- EHNDTW (data translation)
- EHNNETW (network redirector)
- EHNRQW (remote SQL)

- EHNSFW (shared folders)
- EHNSRW (submit remote)
- EHNTFW (transfer)
- EHNVPW (virtual printer)
- EHNCLN1 (internal)
- EHNCLN2 (internal)
- EHNCLP1 (internal)
- EHNCL01 (internal)
- EHNCL02 (internal)
- EHNHAW (internal)
- EHNRTRW (internal)
- PCSMOND (internal)

PC Support Tools Folder

The PC Support tools folder (QIWSTOOL) contains a number of tools and files to assist an application programmer in the development of programs which use the PC Support Windows APIs. These include:

- Include files

  Each of the PC Support functions provides an include file which contains pre-defined constants, data structures, and function prototypes. These files can be included in the application program.

- Sample programs

  Each of the PC Support functions provide a sample program to demonstrate how the API routines can be implemented.

- Monitoring tool

  The PC Support monitor traces the PC Support APIs similar to the way the SPY program (provided with the Windows Software Development Kit) monitors Windows messages. This tool can be used by application programmers for problem analysis.

For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

Multitasking Issues

Microsoft Windows is a nonpreemptive multitasking environment. Task switching does not occur until the application running is ready to give up control. Therefore, applications that have processing that may take a long time to complete should yield control to allow other tasks to run. If this is not done, the system appears locked until processing is complete. This can be observed with some Windows applications that access the disks. While applications are accessing the disk, no other applications get control and the user cannot give another application focus.

Before control can be transferred to another task, the Windows application must call one of the Windows APIs that yields control. However, before Windows performs the task switch, it requires the message queue of the foreground task to be empty. Therefore, if messages are on the queue when Yield(), GetMessage(), PeekMessage(), or WaitMessage() is called, the task switch is not done.

Because the messages belong to the application, PC Support APIs cannot discard the messages on behalf of the application to allow task switching. In order to

enable task switching during PC Support API calls, an additional parameter is added to calls that may take longer to complete. The new parameter is a callback address that allows the application to get control while the API call is processing. With this callback routine, the application could process the messages in its queue, thus enabling task switching.

Note: Because messages are processed by applications while a PC Support API is in progress, the application may process messages that lead to a call to an API that is in progress (called by another application). PC Support APIs are not reentrant and return API busy if it is already in progress. When a busy return code is received, the application should either delay the request and allow task switching to occur, or fail the request. If this is not done, a deadlock results because the call in progress would never complete until the second call gives up control.

When an application is about to call an API with a callback procedure, it should disable controls (menu items, push buttons, and so on) that may lead to another call to the same API. This reduces the chance of getting a busy return code.

Note: Due to its complexity, the callback implementation should only be used by experienced Windows programmers. An inexperienced programmer should pass a NULL pointer as the callback address. The NULL pointer disables the callback capability. Applications that do not implement the callbacks should handle the API busy condition as described earlier.

Implementation of the Callback Procedure

The callback procedure should contain code to remove messages from the message queue of the application. This is accomplished by the PeekMessage() function or the GetMessage() function. Once the messages are removed from the queue, they are translated and dispatched to the window the message is intended for. This is repeated until no more messages are on the queue. Task switching occurs when a PeekMessage() or GetMessage() is called while no messages are on the queue.

The following is a sample of the callback function:

```
int _far _pascal CallBackFunction(void)
{
  MSG  msg;

  .
  .
  While (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    .
    .
  }

}
```

The callback function must also be exported in the .DEF file of the application.

```
.
.
.
EXPORTS
    .
    .
    CallBackFunction
    .
    .
```

Calling a PC Support/400 API

Before calling an API that supports callbacks, the application should disable controls that lead to another call to the same API. This is done to prevent reentrancy of the API and of the application code itself. Additionally, MakeProcInstance() should be called to create an instance thunk for the callback function. The instance thunk ensures that the callback function uses the correct data segment. When the API call completes, FreeProcInstance should be called to free the instance thunk.

The following is an application that calls an API which supports the callback function:

```
FARPROC lpCallBackPointer;

.
.
/* disable controls that may lead to this code path */

/* hInst is the instance handle */
lpCallBackPointer = MakeProcInstance(CallBackFunction,
                                     hInst);

/* call transfer API with call back */
while (EHNTF_STF(hwndApp,
                 lpCallBackPointer,         /* call back address */
                 &RouterConvId,
                 Function,
                 Buffer,
                 BufferSize,
                 RequestLength,
                 SystemName,
                 Translation,
                 ReturnAddrPtr,
                 ReturnLengthPtr,
                 ReturnSecondPtr) == EHNTF_WAIT_REQUEST_STILL_ACTIVE)
{
  While (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
    .
    .
  }
}
/* free instance thunk */
FreeProcInstance(lpCallBackPointer);

/* enable controls that may lead to this code path */
.
.
```

When a busy return code is received, the application needs to relinquish control before trying the operation again. If it does not relinquish control, the outstanding request never completes.

Note: The code that relinquishes control is the same as the callback function.

The application may choose to display a dialog box that enables a user to cancel a request that is waiting for the busy request to complete. Another alternative is to fail any busy request and have the user try the operation again.

Variable-Naming Conventions

All variables declared in the include files are named using Hungarian notation prefixes. The following describes the set of Hungarian notation prefixes used in the PC Support/400 DLL include files. The prefixes can be combined with one another as well as with a base type.

- Prefixes

  a Array

hp	Huge pointer
lp	Long pointer
p	Near pointer

- Base Types

b	Byte
c	Character
ch	Character
dw	Double word
f	Flag or boolean
fn	Function
h	Handle
ln	Long integer
n	Integer
sz	Null-terminated string
us	Unsigned short integer
w	Word

- Examples:

lpszName	Is a long pointer to a null-terminated character string.
lpachDriveTable	Is a long pointer to an array of characters.

String Format

All strings passed to and from the PC Support APIs must be from the ANSI character set.

Window Handles

All of the PC Support/400 APIs require a window handle that is used internally by PC Support/400. If you do not wish to pass a window handle, you can pass NULL. In this case, the PC Support/400 APIs retrieve the window handle of the current task. This may cause problems if Windows becomes a preemptive multitasking environment. In such an environment, the current task may no longer be your task.

Requirements for Using Windows APIs

To use the Windows APIs, you need the following:

- Extended DOS Version 2 Release 2 or later of PC Support must be loaded. Specifically, this means the following must be loaded:
  - The PC Support Extended Memory Manager (PCSXI)
  - The PC Support Router (STARTRTR)
  - The PC Support Windows program (PCSWIN)

  Note: Other programs may also be required to use a specific API. See the introduction to each Windows API chapter for further details.

- You must be running with Windows in either standard or enhanced mode.
- The WINDOWS.H file that comes with the Microsoft Windows Software Development Kit should be included in your application program.
- Your application program must be linked with the PC Support import library EHNAPIW.LIB.
- The directory name where PC Support Windows DLLs are located must be listed in your path statement. The DLLs are shipped in the QIWSFL2 and QIWSFL2D folders. To get these DLLs downloaded to your PCS subdirectory,

run the PC Support configurator (CFGPCS) and indicate that you want to run Windows.

Chapter 20. Router Windows Application Program Interface

The router Windows API provides a method to write cooperative processing applications between personal computers and AS/400 systems. It insulates the programmer from low-level communications programming and hardware connectivity types. Application programmers need to write both the AS/400 and PC programs when using the router API. Almost anything that can be accessed by the host application can be extended to the partner PC application. This API can be used for performance-critical applications.

Before using the router Windows API, review the concepts described in the following chapters:

- Chapter 1, "Introduction to the DOS Router APPC Interface"
- Chapter 2, "Development of a Router Transaction Program"
- Chapter 3, "Introduction to Router Verb Types"
- Chapter 4, "Router Service Verbs and Basic Conversation Verbs"

This chapter describes the individual routines, data structures, and return codes that make up the router Windows API. The descriptions make reference to predefined constants, data structures, and function prototypes which can be found in the file EHNAPPC.H in the PC Support tools folder (QIWSTOOL). The PC Support tools folder also contains a sample program which demonstrates how to use the router Windows API. For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

For information on the requirements for developing and running applications that use the PC Support Windows APIs, see "Requirements for Using Windows APIs" on page 19-9.

Additional Information

Allocating Buffers in Extended Memory

When running in Windows standard or enhanced mode, the application's data structures and buffers may be in extended memory. The router DLL copies the data to conventional memory and makes the switch to the necessary mode to call the router. The application is protected from this, but you should be aware that the router DLL may not be able to get enough conventional memory to satisfy the send or receive request. If this happens, the application should try the request again with smaller buffers.

Performance Considerations

To improve the performance of your application, the send and receive buffers can be allocated in conventional memory. Otherwise, the DLL copies the data between conventional and extended memory for each send and receive operation.

Router Verbs

For detail on the router verbs, refer to Chapter 4, "Router Service Verbs and Basic Conversation Verbs."

Router API Routines

The following discussions of each router Windows API routine describe in detail:

- Purpose
- Procedure declaration
- Parameters
- Return codes

EHNAPPC_Allocate

Purpose

This function starts a conversation with a partner transaction program.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_Allocate(
                HWND              hWnd,
                unsigned          nBufferLength,
                ConversationType  bType,
                SyncLevelEnum     bSynchLevel,
                LPSTR             lpszLocationName,
                LPSTR             lpszTpn,
                int               nPipLength,
                LPVOID            lpPipData,
                LPDWORD           lpdwConversation);
```

Parameters

hWnd

Identifies the current window of the application.

nBufferLength

Identifies the length of the communications buffer to be allocated by the router. If the size is less than 271, then 271 bytes are allocated. The space is allocated out of the PCSWIN buffer pool. Use the EHNAPPC_GetCapabilities function to determine the best length to use.

bType

Identifies the type of conversation to allocate. Possible values are:

EHNAPPC_BASIC (0)
EHNAPPC_MAPPED (1)

bSynchLevel

Identifies the synchronization level between the local and partner programs. Possible values are:

EHNAPPC_SYNCLEVELNONE (0)
EHNAPPC_SYNCLEVELCONFIRM (1)

lpszLocationName

Points to a null-terminated character string that specifies the host system name. If this pointer is set to NULL, the default system is used.

lpszTpn

Points to a null-terminated character string that specifies the partner program name. If the first character is less than 0x40, then ASCII-to-EBCDIC translation is not done.

nPipLength	Identifies the length of the program initialization parameters (PIP) data. If this variable is 0, no PIP data is sent.
lpPipData	Points to the PIP data. The PIP data must be in GDS format, and must be in EBCDIC. For more information about PIP data, see page 4-10.
lpdwConversation	Points to a doubleword variable that is used to return a handle to be used on subsequent calls. The handle is a unique value for each conversation.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_Confirm

Purpose

This function requests a confirmation that all data sent so far has been received by the partner.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_Confirm(
                HWND      hWnd,
                DWORD     dwConversation,
                LPBYTE    lpRequestToSendRcvd);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.
lpRequestToSendRcvd	Points to a variable which is used to store whether the partner transaction program issued a REQUEST_TO_SEND verb. A value of TRUE indicates the partner transaction program issued a REQUEST_TO_SEND verb.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_Confirmed

Purpose

This function sends a confirmation to a partner that has requested confirmation.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_Confirmed(
                HWND      hWnd,
                DWORD     dwConversation);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_Deallocate

Purpose

This function deallocates an allocated conversation.

Procedure Declaration

```
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_Deallocate(
                HWND            hWnd,
                DWORD           dwConversation,
                DeallocateEnum  bType);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.
bType	Identifies the type of deallocation the router is to perform. Possible values are: EHNAPPC_DEALLOCATESYNCLEVEL (0) EHNAPPC_DEALLOCATEFLUSH (1) EHNAPPC_DEALLOCATEABEND (2)

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_ExtendedAllocate

Purpose

This function starts a conversation with a partner transaction program and overrides the security values specified when the router was started.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_ExtendedAllocate(
                HWND              hWnd,
                unsigned          nBufferLength,
                ConversationType  bType,
                SyncLevelEnum     bSynchLevel,
                LPSTR             lpszLocationName,
                LPSTR             lpszTpn,
                LPSTR             lpszModeName,
                SecurityType      bSecurityType,
                LPSTR             lpszUserId,
                LPSTR             lpszPassword,
                int               nPipLength,
                LPVOID            lpPipData,
                LPDWORD           lpdwConversation);
```

Parameters

hWnd
Identifies the current window of the application.

nBufferLength
Identifies the buffer length to be allocated by the router. If the size is less than 271, then 271 bytes are allocated. The space is allocated out of the PCSWIN buffer pool.

bType
Identifies the type of conversation to allocate. Possible values are:

EHNAPPC_BASIC (0)
EHNAPPC_MAPPED (1)

bSynchLevel
Identifies the synchronization level between the local and partner programs. Possible values are:

EHNAPPC_SYNCLEVELNONE (0)
EHNAPPC_SYNCLEVELCONFIRM (1)

lpszLocationName
Points to a null-terminated character string that specifies the host system name. If this pointer is set to NULL, the default system is used.

lpszTpn
Points to a null-terminated character string that specifies the partner program name. If the first character is less than X'40', then ASCII-to-EBCDIC translation is not done.

lpszModeName	Must be a null pointer.
bSecurityType	Identifies the security type to use. Possible values are: EHNAPPC_SECURITY_NONE (0) EHNAPPC_SECURITY_SAME (1) EHNAPPC_SECURITY_PGM (2)
lpszUserId	Points to a null-terminated character string containing the user ID. The maximum length is 10 characters.
lpszPassword	Points to a null-terminated character string containing the password. The maximum length is 10 characters.
nPipLength	Identifies the length of the PIP data. If this variable is 0, no PIP data is sent.
lpPipData	Points to the PIP data. The PIP data must be in GDS format, and must be in EBCDIC. For more information about PIP data, see page 4-10.
lpdwConversation	Points to a doubleword variable which is used to return a handle to be used on subsequent calls.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_Flush

Purpose

This function causes the router to send any data it may have in its buffers.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_Flush(
                HWND          hWnd,
                DWORD         dwConversation);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_GetAttributes

Purpose

Returns attributes of the specified conversation, including the LU names of the local and partner transaction programs, the level of processing synchronization, and any user ID provided for security.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern unsigned far pascal EHNAPPC_GetAttributes(
                HWND            hWnd,
                DWORD           dwConversation,
                LPBYTE          lpbSyncLevel,
                LPSTR           lpszModeName,
                LPSTR           lpszLuName,
                LPSTR           lpszPluName,
                LPSTR           lpszUserId);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned by EHNAPPC_Allocate or EHNAPPC_Extended Allocate.
lpbSyncLevel	Points to a byte variable that is used to return the synchronization level.
lpszModeName	Points to a null-terminated character string that is used to return the 8-character mode name.
lpszLuName	Points to a null-terminated character string that is used to return the LU of the local transaction program. This LU name was configured in the RTLN value of a PC Support configuration file (CONFIG.PCS).
lpszPluName	Points to a null-terminated character string that is used to return the name of the partner LU.
lpszUserId	Points to a null-terminated character string that is used to return the user ID that was used to establish this connection.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_GetCapabilities

Purpose

This function fills in a data structure indicating the capabilities of the router currently loaded. Refer to "Query_Router_Capabilities" on page 4-4 for information on how to use the Capabilities verb.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern unsigned far pascal EHNAPPC_GetCapabilities(
                HWND            hWnd,
                LPSTR           lpList);
```

Parameters

hWnd — Identifies the current window of the application.

lpList — Points to a capabilities list that is used to retrieve the capability information. A capabilities list consists of a header followed by a variable number of capability structures. On input, the list specifies the capabilities to be queried. On output, it contains the capability information.

Note: For additional structure information, see "appcrtrcap_hdr" on page 20-17, "appcrtrcap_mult" on page 20-18, and "appcrtrcap_query" on page 20-18.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_GetDefaultSystem

Purpose

This function returns the default system name that the router is connected to.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern unsigned int far pascal EHNAPPC_GetDefaultSystem(
                HWND            hWnd,
                LPSTR           lpszDefSysName);
```

Parameters

hWnd — Identifies the current window of the application.

lpszDefSysName — Points to a character buffer that is used to return the default system name. The system name is stored in this buffer as a null-terminated character string.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_IsRouterLoaded

Purpose

This function determines whether or not the router is loaded in memory.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern bool far pascal EHNAPPC_IsRouterLoaded(
                HWND           hWnd);
```

Parameters

hWnd	Identifies the current window of the application.

Return Codes

The return code is FALSE (0) if:

- The PC Support router is not loaded.
- A PC Support router is loaded that is not supported by the router API.
- The PCSWIN program is not loaded.

Otherwise, the return value is TRUE (1).

EHNAPPC_PrepareToReceive

Purpose

This function prepares the program to receive data. Using this function followed by EHNAPPC_ReceiveImmediate is the same as using EHNAPPC_ReceiveAndWait.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_PrepareToReceive(
                HWND      hWnd,
                DWORD     dwConversation);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_QueryConvState

Purpose

This function returns the state of the specified conversation.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern unsigned int far pascal EHNAPPC_QueryConvState(
                HWND        hWnd,
                DWORD       dwConversation);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.

Return Codes

The return value indicates the current state of the conversation. Possible values are:

EHNAPPC_RESET_STATE (0)
EHNAPPC_SEND_STATE (1)
EHNAPPC_RECEIVE_STATE (2)
EHNAPPC_RCVD_CONF_STATE (3)
EHNAPPC_RCVD_CONF_SEND_STATE (4)
EHNAPPC_RCVD_CONF_DEALL_STATE (5)
EHNAPPC_PEND_DEALLOCATE_STATE (6)
EHNAPPC_INVALID_STATE (7)

EHNAPPC_QueryUserId

Purpose

This function returns the user ID used to connect to the specified system.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern unsigned far pascal EHNAPPC_QueryUserId(
                HWND        hWnd,
                LPSTR       lpszLocationName,
                LPSTR       lpszUserId);
```

Parameters

hWnd	Identifies the current window of the application.
lpszLocationName	Points to a null-terminated character string containing the system name to be queried. Specify NULL to query the user ID for the default system.
lpszUserId	Points to a null-terminated character string that is used to return the user ID for the specified system.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_QuerySystems

Purpose

This function returns the names of the systems the router is connected to.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern unsigned far pascal EHNAPPC_QuerySystems(
                HWND         hWnd,
                LPINT        lpSysCount,
                LPSYSSTRUC   lpSys);
```

Parameters

hWnd	Identifies the current window of the application.
lpSysCount	Points to an integer variable which is used to return the number of systems connected.
lpSys	Points to an AS400_Sys structure that is used to return the names of the systems. The default system is the first system in the structure. For a description of the AS400_Sys structure, see "AS400_Sys" on page 20-17.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_ReceiveAndWait

Purpose

This function waits for information to arrive on the conversation, then receives the information.

Note: When this function is called, all other Windows applications are suspended until it completes. Use the *Receive Immediate* function instead of this function to avoid suspending other applications.

Procedure Declaration

```
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_ReceiveAndWait(
                      HWND          hWnd,
                      DWORD         dwConversation,
                      FillEnum      bFill,
                      int           nMaxLength,
                      LPVOID        lpReceiveData,
                      LPBYTE        lpWhatReceived,
                      LPBYTE        lpRequestToSendRcvd,
                      LPWORD        lpReceiveDataLength );
```

Parameters

hWnd
Identifies the current window of the application.

dwConversation
Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.

bFill
Indicates the form in which the program is to receive data. Possible values are:

EHNAPPC_BUFFER (0) (fill the buffer)
EHNAPPC_LL (1) (receive a complete or truncated logical record)

nMaxLength
Indicates the largest amount of data that can be accepted.

lpReceiveData
Points to a buffer where the data is to be received.

lpWhatReceived
Indicates what has been received by the router. Possible values are:

EHNAPPC_DATA (0)
EHNAPPC_DATACOMPLETE (1)
EHNAPPC_DATAINCOMPLETE (2)
EHNAPPC_RECEIVEDCONFIRM (3)
EHNAPPC_RECEIVEDCONFIRMSEND (4)
EHNAPPC_RECEIVEDCONFIRMDEALLOC (5)
EHNAPPC_RECEIVEDSEND (6)

lpRequestToSendRcvd
Points to a variable that is used to store whether the partner transaction program issued a REQUEST_TO_SEND verb. A value of TRUE (1) indicates the partner transaction program issued a REQUEST_TO_SEND verb.

lpReceiveDataLength
Points to a variable that is used to return the amount of data received by the router.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_ReceiveImmediate

Purpose

This function checks to see if something has been received. If so, the data is returned.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_ReceiveImmediate(
                        HWND            hWnd,
                        DWORD           dwConversation,
                        FillEnum        bFill,
                        int             nMaxLength,
                        LPVOID          lpReceiveData,
                        LPBYTE          lpWhatReceived,
                        LPBYTE          lpRequestToSendRcvd,
                        LPWORD          lpReceiveDataLength );
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.
bFill	Indicates the form in which the program is to receive data. Possible values are: EHNAPPC_BUFFER (0) (fill the buffer) EHNAPPC_LL (1) (receive a complete or truncated logical record)
nMaxLength	Indicates the largest amount of data that can be accepted.
lpReceiveData	Points to a buffer where the data is to be received.
lpWhatReceived	Identifies what has been received by the router. Possible values are: EHNAPPC_DATA (0) EHNAPPC_DATACOMPLETE (1) EHNAPPC_DATAINCOMPLETE (2) EHNAPPC_RECEIVEDCONFIRM (3) EHNAPPC_RECEIVEDCONFIRMSEND (4) EHNAPPC_RECEIVEDCONFIRMDEALLOC (5) EHNAPPC_RECEIVEDSEND (6)
lpRequestToSendRcvd	Points to a variable which is used to store whether the partner transaction program issued a REQUEST_TO_SEND verb. A value of TRUE (1) indicates the partner transaction program issued a REQUEST_TO_SEND verb.
lpReceiveDataLength	Points to a variable that is used to return the amount of data received by the router.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_RemoteProgramStart

Note: This API is listed with the router Windows APIs but cannot be used with any other router Windows APIs.

Purpose

This function allows Windows applications to start a program on a remote AS/400 system.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern word far pascal EHNAPPC_RemoteProgramStart(
                         HWND             hWnd,
                         LPSTR            lpszHostSystemName,
                         LPSTR            lpszHostProgramName,
                         LPSTR            lpszHostLibraryName,
                         char FAR        *lpchPipData,
                         WORD             wPipDataLength);
```

Parameters

hWnd	Identifies the current window of the application.
lpszHostSystemName	Points to a null-terminated character string that contains the name of the remote system. The maximum length of this string is 8 characters. If this pointer is null, the default system name is used.
lpszHostProgramName	Points to a null-terminated character string that contains the name of the host program to be started.
lpszHostLibraryName	Points to a null-terminated character string that contains the library path of the host program. If this pointer is null, the library list of the user is searched.
lpchPipData	Points to the program initialization parameter (PIP) data area for the host program. If this pointer is null, no PIP data is sent. For more information about PIP data, see page 4-10.
wPipDataLength	Contains the length of the PIP data.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_RqsToSend

Purpose

This function requests that the partner give up control of the conversation. The router places the conversation in Send state when the local transaction program subsequently receives EHNAPPC_RECEIVEDSEND (6) in the lpWhatReceived parameter of a Receive verb from the partner transaction program.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_RqsToSend(
                        HWND            hWnd,
                        DWORD           dwConversation);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_SendData

Purpose

This function sends data to the partner transaction program.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_SendData(
                HWND      hWnd,
                DWORD     dwConversation,
                int       nSendDataLength,
                LPVOID    lpSendDataBuffer,
                LPBYTE    lpRequestToSendRcvd);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.
nSendDataLength	Identifies the length of the data in the send buffer.
lpSendDataBuffer	Identifies the address of the send buffer. For information on how the data should be formatted in the send buffer, see the description of the SEND_DATA verb on page 4-38.

lpRequestToSendRcvd	Points to a variable that is used to store whether the partner transaction program issued a REQUEST_TO_SEND verb. A value of TRUE indicates the partner transaction program issued a REQUEST_TO_SEND verb.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

EHNAPPC_SendError

Purpose

This function indicates to the partner transaction program that some error has been found. After using this function, the local program is in receive state.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNAPPC.H"
extern int far pascal EHNAPPC_SendError(
                HWND      hWnd,
                DWORD     dwConversation,
                LPBYTE    lpRequestToSendRcvd);
```

Parameters

hWnd	Identifies the current window of the application.
dwConversation	Identifies the conversation handle returned from either EHNAPPC_Allocate or EHNAPPC_ExtendedAllocate.
lpRequestToSendRcvd	Points to a variable that is used to store whether the partner transaction program issued a REQUEST_TO_SEND verb. A value of TRUE indicates the partner transaction program issued a REQUEST_TO_SEND verb.

Return Codes

For return codes, see "Return Codes for the Router Windows API" on page 20-19.

Router Windows API Structures

AS400_Sys

Description

This structure is used to store the names of the systems the router is connected to.

```
struct AS400_sys
   {
   unsigned char EHNAPPC_SysName[EHNAPPC_MAX_SYSTEMS]
                                [EHNAPPC_SYSNAME_SYSNAME_LENGTH];
   };
```

Fields	Description
EHNAPPC_SysName	Is used to store the name of a connected system. System names are returned as null-terminated strings. The first system returned in the array is the default system (EHNAPPC_MAX_SYSTEMS = 32 and EHNAPPC_SYS_NAME = 10).

appcrtrcap_hdr

Description

This is the structure of the router capability list header.

```
struct appcrtrcap_hdr
   {
   unsigned char rc;
   unsigned char opcode;
   unsigned int length;
   };
```

Field	Description
rc	Is used to store the overall return code of the capabilities request.
opcode	Signals the get capabilities request. Its value must be EHNAPPC_OC_CAPABILITIES (0x17).
length	Identifies the length of the entire capabilities list. The length includes the size of the header plus the size of each capability structure. Refer to "Query_Router_Capabilities" on page 4-4 for information on how to use the Capabilities verb.

appcrtrcap_mult

Description

This is the capability structure used to determine the optimal communications buffer multiplier. The size of the communications buffer should be a multiple of this optimal communications buffer multiplier to make the most efficient use of DOS memory.

```
struct appcrtrcap_mult
   {
   unsigned int length;
   unsigned char identifier;
   unsigned char rc;
   unsigned int data;
   };
```

Field	Description
length	Identifies the length of this capability structure.
identifier	Signals the optimal communications buffer multiplier. Its value must be EHNAPPC_CAP_OPTIMAL_COM_SIZE (X'02'). Refer to "Query_Router_Capabilities" on page 4-4 for information on how to use the Capabilities verb.
rc	Is used to store the return code of this capability request.
data	Is used to return the optimal communications buffer multiplier.

appcrtrcap_query

Description

This is the capability structure used to query if the router supports the specified capability.

```
struct appcrtrcap_query
   {
   unsigned int length;
   unsigned char identifier;
   unsigned char rc;
   unsigned char data;
   };
```

Field	Description
length	Identifies the length of this capabilities structure.
identifier	Identifies the function to be queried. Possible values are: EHNAPPC_CAP_QUERY_CONV_STATE (3) EHNAPPC_CAP_EXT_ALLOCATE (4)
rc	Is used to store the return code of this capability request.

data — Is used to return whether or not the specified function is supported. Refer to "Query_Router_Capabilities" on page 4-4 for information on how to use the Capabilities verb.

Return Codes for the Router Windows API

Functions in the router Windows API use the following return codes. These are constants defined in EHNAPPC.H.

Table 20-1 (Page 1 of 2). Return Codes for the Router Windows API

Return Code	Hex Value	Description
EHNAPPC_OK	0	Command completed successfully.
EHNAPPC_DEALLOCNORMAL	1	Deallocation normal.
EHNAPPC_PROGRAMERRNOTRUNCATION	2	Program error; no truncation.
EHNAPPC_PROGRAMERRTRUNCATION	3	Program error; truncation.
EHNAPPC_PROGRAMERRPURGING	4	Program error; purging.
EHNAPPC_RESOURCEFAILURERETRY	5	Resource failure retry.
EHNAPPC_RESOURCEFAILURENORETRY	6	Resource failure no retry.
EHNAPPC_UNSUCCESSFUL	7	Unsuccessful.
EHNAPPC_APPCBUSY	8	APPC busy.
EHNAPPC_PARMCHKINVALIDVERB	14	Parameter check; invalid verb.
EHNAPPC_PARMCHKINVALIDCONVERID	15	Parameter check; invalid conversation ID.
EHNAPPC_PARMCHKBUFFERCROSSSEG	16	Parameter check; buffer crossed segment.
EHNAPPC_PARMCHKTPNAMELENGTH	17	Parameter check; transaction program name length.
EHNAPPC_PARMCHKINVCONVERTYPE	18	Parameter check; invalid conversation type.
EHNAPPC_PARMCHKBADSYNCLVLALLOC	19	Parameter check; bad synchronization level allocate.
EHNAPPC_PARMCHKBADRETURNCTRL	1A	Parameter check; bad return control.
EHNAPPC_PARMCHKPIPTOOLONG	1B	Parameter check; PIP data too long.
EHNAPPC_PARMCHKBADPARTNERNAME	1C	Parameter check; bad partner name.
EHNAPPC_PARMCHKCONFNOTALLOWED	1D	Parameter check; confirm not allowed.
EHNAPPC_PARMCHKBADDEALLOCTYPE	1E	Parameter check; bad deallocation type.
EHNAPPC_PARMCHKPREPTORCVTYPE	1F	Parameter check; prepare to receive type.
EHNAPPC_PARMCHKBADFILLTYPE	20	Parameter check; bad fill type.
EHNAPPC_PARMCHKRECMAXLEN	21	Parameter check; receive maximum length.
EHNAPPC_PARMCHKUNKNOWNSECTYPE	22	Parameter check; unknown security type.
EHNAPPC_PARMCHKRESFLDNOTZERO	23	Parameter check; reserved field not zero.
EHNAPPC_STATECHKNOTINCONFSTAT	28	State check; not in confirmed state.
EHNAPPC_STATECHKNOTINRECEIVE	29	State check; not in receive.
EHNAPPC_STATECHKREQSNDBADSTAT	2A	State check; request to send bad state.
EHNAPPC_STATECHKSNDINBADSTATE	2B	State check; send in bad state.

Table 20-1 (Page 2 of 2). Return Codes for the Router Windows API

Return Code	Hex Value	Description
EHNAPPC_STATECHKSNDERRBADSTAT	2C	State check; send error bad state.
EHNAPPC_ALLOCERRNORETRY	32	Allocation error; no retry.
EHNAPPC_ALLOCERRRETRY	33	Allocation error; retry.
EHNAPPC_ALLOCERRPGMNOTAVAILNR	34	Allocation error; program not available no retry.
EHNAPPC_ALLOCERRTPNNOTRECOG	35	Allocation error; transaction program name not recognized.
EHNAPPC_ALLOCERRPGMNOTAVAILR	36	Allocation error; program not available retry.
EHNAPPC_ALLOCERRSECNOTVALID	37	Allocation error; security not valid.
EHNAPPC_ALLOCERRCONVTYP	38	Allocation error; conversation type mismatch.
EHNAPPC_ALLOCERRPIPNOTALLOWED	39	Allocation error; PIP data not allowed.
EHNAPPC_ALLOCERRPIPNOTCORRECT	3A	Allocation error; PIP data not correct.
EHNAPPC_ALLOCERRSYNCHLEVEL	3B	Allocation error; synchronization level not supported.
EHNAPPC_DEALLOCABENDPROGRAM	46	Deallocation abend program.
EHNAPPC_INSUFFICIENTMEMORY	47	Insufficient memory.
EHNAPPC_MEMORYALLOCERROR	48	Memory allocation error.
EHNAPPC_TOOMANYCONVERSATIONS	49	Too many conversations.
EHNAPPC_CONVTABLEFULL	4A	Conversion table full.
EHNAPPC_ROUTERNOTINSTALLED	4B	Router not installed.
EHNAPPC_ROUTERWRONGLEVEL	4C	Router at wrong level.
EHNAPPC_PCSWINNOTLOADED	4D	PCSWIN not loaded.
EHNAPPC_PCSWINOUTOFMEMORY	4E	PCSWIN out of memory.
EHNAPPC_INVALIDUSERIDLEN	4F	Invalid user ID length.
EHNAPPC_INVALIDPASSWORDLEN	50	Invalid password length.
EHNAPPC_INVALIDUNAME	51	Invalid LU Length.
EHNAPPC_UNDEFINED	63	Undefined.

Chapter 21. Transfer Function Windows Application Program Interface

The transfer function API provides access to AS/400 databases using SQL syntax. It can retrieve individual records, however, it only operates on complete database files when performing updates. This API is useful for applications that require host query capability or that desire favorable performance for updating or downloading entire files.

This chapter describes the individual routines and return values that make up the transfer function API. The descriptions make reference to predefined constants, data structures, and function prototypes which can be found in the file EHNTFW.H on the PC Support tools folder (QIWSTOOL). The PC Support tools folder also contains a sample program which demonstrates how to use the transfer function Windows API (WTFRSAMP on QIWSTOOL). For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

For information on the requirements for developing and running applications that use the PC Support Windows APIs, see Part 3, "PC Support Windows Application Program Interfaces" on page 18-23. In addition, see "Transfer Function Overview" on page 5-1 for detailed information on the transfer function and the syntax of available transfer statements.

Transfer Function Windows API Routines

The transfer function API contains only one entry point (EHNTF_STF). The bFunction parameter is used to differentiate between the different possible functions to be performed. Depending on the function specified, only certain parameters may make sense and, therefore, only these parameters are required to have valid values when the procedure is called.

The following discussions of each transfer function Windows API routine describe in detail:

- Purpose
- Procedure declaration
- Parameters
- Return codes

Open Transfer Request (bFunction = EHNTF_SEND_REQUEST)

Purpose

This function defines the transfer request criteria for records which are to be transferred to or from an AS/400 system. It also specifies which system the transfer request is dealing with. If the request is successful, a transfer request is now open and a router conversation ID is associated with this transfer request. The router ID should be used for subsequent calls to send or retrieve data for this transfer request.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNTFW.H"
int FAR PASCAL EHNTF_STF(HWND            hWindowHandle,
                         FARPROC         lpProcCallBack,
                         LPDWORD         lpdwRtrConvId,
                         BYTE            bFunction,
                         char far       *lpachBufferAddress,
                         int             nBufferLength,
                         int             nRequestLength,
                         char far       *lpachSystemAddress,
                         BYTE            bConversion,
                         char far * far *lplpachReturnAddress,
                         LPINT           lpnReturnLength,
                         LPINT           lpnReturnCodeSecond)
```

Parameters

hWindowHandle

Identifies the current window of the application.

lpProcCallBack

Identifies the address of a routine within the calling application that the transfer function DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy transfer request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: This parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine. The message box indicates the personal computer is not hung.

lpdwRtrConvId

Points to a double-word variable containing a router conversation ID of zero. If the request is successful, this variable is used to store the router conversation ID associated with this transfer request. This value should be saved and used for all subsequent work with this transfer request.

bFunction

Identifies the open transfer request function. Its value must be EHNTF_SEND_REQUEST (01).

lpachBufferAddress

Points to a buffer that contains the actual transfer request. The transfer request may be one of the following types: SELECT, REPLACE, EXTRACT, or OPTIONS. For the syntax of each of these commands, see "Transfer Requests" on page 5-5. If the request fails, this buffer may be used to store error message text.

nBufferLength	Identifies the length of the transfer request buffer. This value must be 4096.
nRequestLength	Identifies the length of the transfer request.
lpachSystemAddress	Points to a character string of the form: `SYSTEM(system name)` where SYSTEM is a keyword and system name is the name of the system the request is to be sent to. If the pointer is null or the string is `SYSTEM()`, the default system is used.
bConversion	Is ignored for this request.
lplpachReturnAddress	Points to a pointer variable that is used to store the address within the transfer request buffer where error message text was placed. The value of this pointer is only valid if the return code from this request is nonzero and the return length parameter (lpnReturnLength) is also nonzero.
lpnReturnLength	Points to an integer variable that is used to return the number of templates associated with the transfer request. The number of templates is the number of fields in each record that satisfies the criteria of the transfer request. If the request fails, this variable is used to store the length of the error message text returned.
lpnReturnCodeSecond	Points to an integer variable that is used to return one of the following: • Zero, if the return code is zero. • The number of templates that can be retrieved, if the return code contains a warning. A warning is any return code with a value less than X'2000'. • A Windows return code, if the return code is EHNTF_OPERATING_SYSTEM_ERROR (X'5400'). • The position in the request of the character that cannot be translated, if the return code is EHNTF_XFER_REQ_CHAR_UNXLATABLE (X'2002'). • The position in the request where the error occurred, if the return code is EHNTF_AS400_ERROR_OFFSET_GIVEN (X'3000').

Return Codes

For return codes, see "Return Codes for the Transfer Function Windows API" on page 21-15.

Retrieve Templates (bFunction = EHNTF_RETRIEVE_TEMPLATES)

Purpose

This function returns a template associated with a transfer request. A template contains the description of a field within a record that satisfies the criteria of the transfer request. This description includes the field name, length, and data type. If the field contains binary, decimal, or packed decimal data, the description also includes the maximum number of digits the field can hold. This function only returns one template per call. To retrieve all of the templates for a transfer request, this routine should be called multiple times until a return code of EHNTF_EOF (X'1FFF') is returned.

For additional information about the format of the retrieve template, see "Retrieve the Templates Function (AL=02)" on page 5-23.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNTFW.H"
int FAR PASCAL EHNTF_STF(HWND            hWindowHandle,
                         FARPROC         lpProcCallBack,
                         LPDWORD         lpdwRtrConvId,
                         BYTE            bFunction,
                         char far       *lpachBufferAddress,
                         int             nBufferLength,
                         int             nRequestLength,
                         char far       *lpachSystemAddress,
                         BYTE            bConversion,
                         char far * far *lplpachReturnAddress,
                         LPINT           lpnReturnLength,
                         LPINT           lpnReturnCodeSecond)
```

Parameters

hWindowHandle

Identifies the current window of the application.

lpProcCallBack

Identifies the address of a routine within the calling application that the transfer function DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy transfer request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: This parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine. The message box indicates the personal computer is not hung.

lpdwRtrConvId	Points to the router conversation ID for the transfer request being used. This must be the same value passed back on the open of the transfer request (bFunction = EHNTF_SEND_REQUEST).
bFunction	Identifies the retrieve templates function. Its value must be EHNTF_RETRIEVE_TEMPLATES (02).
lpachBufferAddress	Points to a buffer to receive the templates. The length of this buffer is 4096 bytes.
nBufferLength	Is ignored for this request.
nRequestLength	Is ignored for this request.
lpachSystemAddress	Is ignored for this request.
bConversion	Is ignored for this request.
lplpachReturnAddress	Points to a pointer variable that is used to store the address, within the transfer request buffer, where the retrieved template was stored. For additional information about the format of the retrieve template, see "Retrieve the Templates Function (AL=02)" on page 5-23. If the request fails, the variable may be used to store the address within the transfer request buffer where error message text was placed. This only happens when the return code from this request is nonzero and the return length parameter (lpnReturnLength) is also nonzero.
lpnReturnLength	Points to an integer variable that is used to return the number of bytes in the retrieved template. If the request fails with a return code of EHNTF_AS400_ERROR_OFFSET_GIVEN (X'3000') or EHNTF_AS400_ERROR_NO_OFFSET (X'3100'), this variable is used to store the length of the error message text returned.
lpnReturnCodeSecond	Points to an integer variable that is used to return one of the following: • Zero, if the return code is zero • A Windows return code, if the return code is EHNTF_OPERATING_SYSTEM_ERROR (X'5400')

Return Codes

For return codes, see "Return Codes for the Transfer Function Windows API" on page 21-15.

Retrieve Records (bFunction = EHNTF_RETRIEVE_RECORDS)

Purpose

This function retrieves a record from the AS/400 system that satisfies the criteria of the transfer request. This function also allows the caller to specify the format in which the record should be retrieved. This function returns only one record per call. To retrieve all of the records that satisfy the transfer request, this routine should be called multiple times until a return code of EHNTF_EOF (X'1FFF') is returned.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNTFW.H"
int FAR PASCAL EHNTF_STF(HWND            hWindowHandle,
                         FARPROC         lpProcCallBack,
                         LPDWORD         lpdwRtrConvId,
                         BYTE            bFunction,
                         char far       *lpachBufferAddress,
                         int             nBufferLength,
                         int             nRequestLength,
                         char far       *lpachSystemAddress,
                         BYTE            bConversion,
                         char far * far *lplpachReturnAddress,
                         LPINT           lpnReturnLength,
                         LPINT           lpnReturnCodeSecond)
```

Parameters

hWindowHandle

Identifies the current window of the application.

lpProcCallBack

Identifies the address of a routine within the calling application that the transfer function DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy transfer request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: This parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine. The message box indicates the personal computer is not hung.

lpdwRtrConvId

Points to the router conversation ID for the transfer request being used. This must be the same value passed back on the open (bFunction = EHNTF_SEND_REQUEST) of the transfer request.

bFunction

Identifies the retrieve records function. Its value must be EHNTF_RETRIEVE_RECORDS (03).

lpachBufferAddress

Points to a buffer to receive the record. The length of this buffer is 4096 bytes.

nBufferLength

Is ignored for this request.

nRequestLength

Is ignored for this request.

lpachSystemAddress

Is ignored for this request.

bConversion

Identifies the type of conversion to be done on the record being retrieved. Valid values are:

- EHNTF_NO_CONVERSION (X'00') No conversion is done.
- EHNTF_DOS_RANDOM (X'01') Records are changed to the type used when writing a DOS random file. This is the default.
- EHNTF_DOS_RANDOM_EXCEPT (X'02') Records are changed to the type used when writing a DOS random file. Untranslatable EBCDIC values are changed to X'3F' or X'FF'.
- EHNTF_DOS_RANDOM2 (X'11') Records are changed to the type used when writing a DOS random type 2 file.
- EHNTF_DOS_RANDOM2_EXCEPT (X'12') Records are changed to the type used when writing a DOS random type 2 file. Untranslatable EBCDIC values are changed to X'3F' or X'FF'.
- EHNTF_NO_CONVERSION_NULL_VAR (X'80') Null conversions are performed. Null and variable-length fields are supported.
- EHNTF_DOS_RANDOM_NULL_VAR (X'81') Records are changed to the type used when writing a DOS random file. Null and variable-length fields are supported.
- EHNTF_DOS_RANDOM_E_NULL_VAR (X'82') Records are changed to the type used when writing a DOS random file. Untranslatable EBCDIC values are changed to X'3F' or X'FF'. Null and variable-length fields are supported.
- EHNTF_DOS_RANDOM2_NULL_VAR (X'91') Records are changed to the type used when writing a DOS random type 2 file. Null and variable-length fields are supported.
- EHNTF_DOS_RANDOM2_NULL_VAR (X'92') Records are changed to the type used when writing a DOS random type 2 file. Untranslatable EBCDIC values are changed to X'3F' or X'FF'. Null and variable-length fields are supported.

For more information on conversion types, see "Retrieve the Records Function (AL=03)" on page 5-25.

lplpachReturnAddress

Points to a pointer variable that is used to store the address, within the transfer request buffer, where the retrieved record was placed. If the request fails, the variable may be used to store the address, within the transfer request buffer, where error message text was placed. This only happens when the return code from this request is nonzero and the return length parameter (lpnReturnLength) is also nonzero.

lpnReturnLength

Points to an integer variable that is used to return the number of bytes in the retrieved record. If the request fails with a return code of

EHNTF_AS400_ERROR_OFFSET_GIVEN (0x3000) or EHNTF_AS400_ERROR_NO_OFFSET (0x3100), this variable is used to store the length of the error message text returned.

lpnReturnCodeSecond — Points to an integer variable that is used to return one of the following:

- Zero, if the return code is zero
- The number of untranslatable characters, if the return code is EHNTF_UNXLATABLE_DATA (X'0300')
- A Windows return code, if the return code is EHNTF_OPERATING_SYSTEM_ERROR(X'5400')

Return Codes

For return codes, see "Return Codes for the Transfer Function Windows API" on page 21-15.

Close Request (bFunction = EHNTF_CLOSE_REQUEST)

Purpose

This function closes the AS/400 file associated with a transfer request. This function should be used after all data has been sent to or retrieved from the AS/400 system. A transfer request can also be closed by retrieving all records until an end-of-file condition is met. This function does not end the conversation associated with the AS/400 system for the transfer request; therefore, the router conversation ID is still active.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNTFW.H"
int FAR PASCAL EHNTF_STF(HWND            hWindowHandle,
                         FARPROC         lpProcCallBack,
                         LPDWORD         lpdwRtrConvId,
                         BYTE            bFunction,
                         char far       *lpachBufferAddress,
                         int             nBufferLength,
                         int             nRequestLength,
                         char far       *lpachSystemAddress,
                         BYTE            bConversion,
                         char far * far *lplpachReturnAddress,
                         LPINT           lpnReturnLength,
                         LPINT           lpnReturnCodeSecond)
```

Parameters

hWindowHandle — Identifies the current window of the application.

lpProcCallBack — Identifies the address of a routine within the calling application that the transfer function DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy transfer request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also

must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: This parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine. The message box indicates the personal computer is not hung.

lpdwRtrConvId	Points to the router conversation ID for the transfer request being used. This must be the same value passed back on the open (bFunction = EHNTF_SEND_REQUEST) of the transfer request.
bFunction	Identifies the close transfer request function. Its value must be EHNTF_CLOSE_REQUEST (04).
lpachBufferAddress	Points to a buffer to receive any error message text. The length of the buffer must be 4096 bytes.
nBufferLength	Is ignored for this request.
nRequestLength	Is ignored for this request.
lpachSystemAddress	Is ignored for this request.
bConversion	Is ignored for this request.
lplpachReturnAddress	Points to a pointer variable that is used to store the address, within the transfer request buffer, where error message text was placed. The value of this pointer is only valid if the return code from this request is nonzero and the return length parameter (lpnReturnLength) is also nonzero.
lpnReturnLength	Points to an integer variable that is used to store the length of the error message text returned, if the request fails with a return code of EHNTF_AS400_ERROR_OFFSET_GIVEN (X'3000') or EHNTF_AS400_ERROR_NO_OFFSET (X'3100').
lpnReturnCodeSecond	Points to an integer variable that is used to return one of the following: • Zero, if the return code is zero • A Windows return code, if the return code is EHNTF_OPERATING_SYSTEM_ERROR (X'5400')

Return Codes

For return codes, see "Return Codes for the Transfer Function Windows API" on page 21-15.

End All Transfer Request Conversations (bFunction = EHNTF_END_ALL_REQ_CONVERSATIONS)

Purpose

This function immediately closes all transfer request files for a specified window and ends all conversations associated with the transfer requests. This function should only be used if there is more than one conversation active for the window. If only one conversation is active, the end transfer request (bFunction = EHNTF_END_ONE_REQ_CONVERSATION) function should be used.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNTFW.H"
int FAR PASCAL EHNTF_STF(HWND            hWindowHandle,
                         FARPROC         lpProcCallBack,
                         LPDWORD         lpdwRtrConvId,
                         BYTE            bFunction,
                         char far       *lpachBufferAddress,
                         int             nBufferLength,
                         int             nRequestLength,
                         char far       *lpachSystemAddress,
                         BYTE            bConversion,
                         char far * far *lplpachReturnAddress,
                         LPINT           lpnReturnLength,
                         LPINT           lpnReturnCodeSecond)
```

Parameters

WindowHandle

Identifies the current window of the application.

lpProcCallBack

Identifies the address of a routine within the calling application that the transfer function DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy transfer request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, "Multitasking Issues" on page 19-5.

Note: This parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine. The message box indicates the personal computer is not hung.

lpdwRtrConvId

Is ignored for this request.

bFunction

Identifies the end all transfer request conversations function. Its value must be EHNTF_END_ALL_REQ_CONVERSATIONS (05).

lpachBufferAddress	Points to a buffer to receive any error message text. The length of this buffer must be 4096 bytes.
nBufferLength	Is ignored for this request.
nRequestLength	Is ignored for this request.
lpachSystemAddress	Is ignored for this request.
bConversion	Is ignored for this request.
lplpachReturnAddress	Points to a pointer variable that is used to store the address, within the transfer request buffer, where error message text was placed. The value of this pointer is only valid if the return code from this request is nonzero and the return length parameter (lpnReturnLength) is also nonzero.
lpnReturnLength	Points to an integer variable that is used to store the length of the error message text returned, if the request fails with a return code of EHNTF_AS400_ERROR_OFFSET_GIVEN (X'3000') or EHNTF_AS400_ERROR_NO_OFFSET (X'3100').
lpnReturnCodeSecond	Points to an integer variable that is used to return one of the following: • Zero, if the return code is zero • A Windows return code, if the return code is EHNTF_OPERATING_SYSTEM_ERROR (X'5400')

Return Codes

For return codes, see "Return Codes for the Transfer Function Windows API" on page 21-15.

Send Records (bFunction = EHNTF_SEND_RECORDS)

Purpose

This function sends a record to the AS/400 system. It also allows the caller to specify the format of the record being sent. This function sends only one record per call. To send multiple records, this routine should be called multiple times. When all records have been sent, the transfer request should be closed by calling the close transfer request (bFunction = EHNTF_CLOSE_REQUEST) routine.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNTFW.H"
int FAR PASCAL EHNTF_STF(HWND           hWindowHandle,
                         FARPROC        lpProcCallBack,
                         LPDWORD        lpdwRtrConvId,
                         BYTE           bFunction,
                         char far      *lpachBufferAddress,
                         int            nBufferLength,
                         int            nRequestLength,
                         char far      *lpachSystemAddress,
                         BYTE           bConversion,
                         char far * far *lplpachReturnAddress,
                         LPINT          lpnReturnLength,
                         LPINT          lpnReturnCodeSecond)
```

Parameters

hWindowHandle

Identifies the current window of the application.

lpProcCallBack

Identifies the address of a routine within the calling application that the transfer function DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy transfer request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: This parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine. The message box indicates the personal computer is not hung.

lpdwRtrConvId

Points to the router conversation ID for the transfer request being used. This must be the same value passed back on the open (bFunction = EHNTF_SEND_REQUEST) of the transfer request.

bFunction

Identifies the send records function. Its value must be EHNTF_SEND_RECORDS (06).

lpachBufferAddress

Points to a buffer containing the record to be sent. The length of this buffer must be 4096 bytes.

nBufferLength

Is ignored for this request.

nRequestLength

Identifies the length of the record to send.

lpachSystemAddress

Is ignored for this request.

bConversion

Identifies the type of conversion to be done on the record being sent. Valid values are:

- EHNTF_NO_CONVERSION (X'00') No conversion is done.
- EHNTF_DOS_RANDOM (X'01') The records are converted from personal computer DOS random format to system format.
- EHNTF_DOS_RANDOM2 (X'11') The records are converted from personal computer DOS random type 2 format to system format.
- EHNTF_NO_CONVERSION_NULL_VAR (X'80') Null conversions are performed. Null and variable-length fields are supported.
- EHNTF_DOS_RANDOM_NULL_VAR (X'81') The records change in the same way the interactive or automatic transfer function program changes records when read from a DOS random file.
- EHNTF_DOS_RANDOM2_NULL_VAR (X'91') The records change in the same way interactive or automatic transfer function programs change records when read from a DOS random type 2 file.

For more information on conversion types, see "Retrieve the Records Function (AL=03)" on page 5-25.

lplpachReturnAddress

Points to a pointer variable that is used to store the address, within the transfer request buffer, where error message text was placed. The value of this pointer is only valid if the return code from this request is nonzero and the return length parameter (lpnReturnLength) is also nonzero.

lpnReturnLength

Points to an integer variable that is used to store the length of the error message text returned, if the request fails with a return code of EHNTF_AS400_ERROR_OFFSET_GIVEN (X'3000') or EHNTF_AS400_ERROR_NO_OFFSET (X'3100').

lpnReturnCodeSecond

Points to an integer variable that is used to return one of the following:

- Zero, if the return code is zero
- The number of characters that cannot be translated, if the return code is EHNTF_UNXLATABLE_PC_TO_400_DATA (X'0302')
- The position of the first character that cannot be translated, if the return code is EHNTF_NO_XFER_UNXLAT_NUM_DATA (X'2003')
- The position in which a digit range error occurred, if the return code is EHNTF_RANGE_ERROR_NUMERIC_DATA (0x2004)
- A Windows return code, if the return code is EHNTF_OPERATING_SYSTEM_ERROR (X'5400')

Return Codes

For return codes, see "Return Codes for the Transfer Function Windows API" on page 21-15.

End Transfer Request Conversation (bFunction = EHNTF_END_ONE_REQ_CONVERSATION)

Purpose

This function closes a transfer request file and ends the AS/400 conversation associated with the request.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNTFW.H"
int FAR PASCAL EHNTF_STF(HWND            hWindowHandle,
                         FARPROC         lpProcCallBack,
                         LPDWORD         lpdwRtrConvId,
                         BYTE            bFunction,
                         char far       *lpachBufferAddress,
                         int             nBufferLength,
                         int             nRequestLength,
                         char far       *lpachSystemAddress,
                         BYTE            bConversion,
                         char far * far *lplpachReturnAddress,
                         LPINT           lpnReturnLength,
                         LPINT           lpnReturnCodeSecond)
```

Parameters

hWindowHandle

Identifies the current window of the application.

lpProcCallBack

Identifies the address of a routine within the calling application that the transfer function DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy transfer request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: This parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine. The message box indicates the personal computer is not hung.

lpdwRtrConvId

Points to the router conversation ID for the transfer request being used. This must be the same value passed back on the open (bFunction = EHNTF_SEND_REQUEST) of the transfer request.

bFunction	Identifies the end transfer request conversation function. Its value must be EHNTF_END_ONE_REQ_CONVERSATION (X'07').
lpachBufferAddress	Points to the active transfer request buffer. This should be the same buffer used on the open (bFunction = EHNTF_SEND_REQUEST) of the transfer request being used.
nBufferLength	Is ignored for this request.
nRequestLength	Is ignored for this request.
lpachSystemAddress	Is ignored for this request.
bConversion	Is ignored for this request.
lplpachReturnAddress	Points to a pointer variable that is used to store the address, within the transfer request buffer, where error message text was placed. The value of this pointer is only valid if the return code from this request is nonzero and the return length parameter (lpnReturnLength) is also nonzero.
lpnReturnLength	Points to an integer variable that is used to store the length of the error message text returned, if the request fails with a return code of EHNTF_AS400_ERROR_OFFSET_GIVEN (X'3000') or EHNTF_AS400_ERROR_NO_OFFSET (X'3100').
lpnReturnCodeSecond	Points to an integer variable that is used to return one of the following: • Zero, if the return code is zero • A Windows return code, if the return code is EHNTF_OPERATING_SYSTEM_ERROR (X'5400')

Return Codes

For return codes, see "Return Codes for the Transfer Function Windows API."

Return Codes for the Transfer Function Windows API

Table 21-1 (Page 1 of 3). Return Codes for the Transfer Function Windows API

Return Code	Hex Value	Description
EHNTF_OK	0000	The previous operation was successful.
EHNTF_REQ_ALREADY_ENDED	0103	Transfer request is already ended.
EHNTF_UNXLATABLE_DATA	0300	Untranslatable data found in transferred record.
EHNTF_UNXLATABLE_PC_TO_400_DATA	0302	Untranslatable data in record to be transferred to AS/400 system.
EHNTF_AS400_WARNING	0400	Warning detected by the AS/400 system.
EHNTF_EOF	1FFF	End of file.

Table 21-1 (Page 2 of 3). Return Codes for the Transfer Function Windows API

Return Code	Hex Value	Description
EHNTF_MAX_REQ_ACTIVE	2000	Maximum number of transfer requests active.
EHNTF_XFER_LENGTH_INVALID	2001	Transfer request length not valid.
EHNTF_XFER_REQ_CHAR_UNXLATABLE	2002	Transfer request character cannot be translated.
EHNTF_NO_XFER_UNXLAT_NUM_DATA	2003	Record not transferred to the AS/400 system because of untranslatable numeric data.
EHNTF_RANGE_ERROR_NUMERIC_DATA	2004	Numeric data does not fit into range specified by digits.
EHNTF_BAD_APP_RECORD_LENGTH	2005	Record length given by your application program is incorrect.
EHNTF_INCORRECT_NULL_MAP_OFFSET	2007	Null map offset provided by application is not correct.
EHNTF_FIELD_LENGTH_OVER_4096	2008	Total length of record exceeds 4096 bytes.
EHNTF_WAIT_REQUEST_STILL_ACTIVE	2009	A previous request to the DLL has not finished transferring data.
EHNTF_XFER_REQ_NOT_OPENED	2010	Transfer request not opened.
EHNTF_CANNOT_RETRIEVE_TEMPLATES	2011	Templates cannot be retrieved now.
EHNTF_CANNOT_GET_REC_PC_TO_400	2012	Cannot get records on a PC-to-AS/400 request.
EHNTF_CANNOT_SEND_REC_400_TO_PC	2013	Cannot send records on an AS/400-to-PC request.
EHNTF_INCORRECT_RTR_CONV_ADDR_PARM	2019	Incorrect address specified for the router conversation ID parameter.
EHNTF_INCORRECT_FUNCTION_REQUEST	2020	Incorrect function requested.
EHNTF_INCORRECT_BUFFER_LENGTH	2021	Incorrect buffer length.
EHNTF_INCORRECT_BUFFER_ADDRESS	2022	Incorrect buffer address.
EHNTF_INCORRECT_BUFFER_ADDRESS_PARM	2023	Transfer request buffer overlaps a previous transfer request buffer.
EHNTF_INCORRECT_RETURN_ADDRESS_PARM	2024	Incorrect address specified for the buffer address parameter.
EHNTF_INCORRECT_RETURN_LENGTH_PARM	2025	Incorrect address specified for the return address parameter.
EHNTF_INCORRECT_RETURN_LENGTH_PARM	2026	Incorrect address specified for the return length parameter.
EHNTF_INCORRECT_SEC_RET_COCE_PARM	2027	Incorrect address specified for the secondary return code parameter.
EHNTF_INCORRECT_SYSTEM_ADDRESS_PARM	2028	Incorrect address specified for the system name parameter.
EHNTF_PCSWIN_COMM_BUFFER_TOO_SMALL	2029	PCSWIN communications buffer space exhausted.
EHNTF_OPTIONS_NOT_SUPPORTED	2030	Options statement not supported on this system.
EHNTF_OPTIONS_NOT_ALL_SUPPORTED	2031	Options not fully supported on this system.

Table 21-1 (Page 3 of 3). Return Codes for the Transfer Function Windows API

Return Code	Hex Value	Description
EHNTF_FUNCTION_AND_STYLE_INVALID	2032	Conversion format not allowed with STYLE(NEW) in effect.
EHNTF_STYLE_KEYWORD_NOT_SUPPORT	2033	STYLE keyword not supported on this system.
EHNTF_AS400_ERROR_OFFSET_GIVEN	3000	Error detected by AS/400 system; statement offset provided.
EHNTF_AS400_ERROR_NO_OFFSET	3100	Error detected by AS/400 system; no statement offset provided.
EHNTF_ROUTER_NOT_STARTED	5042	PC Support router has not been started.
EHNTF_CONNECT_TO_SYSTEM_FAILED	5048	Connection failed to system.
EHNTF_SYS_NAME_INCORR_OR_INACTIVE	5050	System name is incorrect or inactive.
EHNTF_SYS_PROGRAM_NOT_FOUND	5052	System program not found.
EHNTF_SYS_PROGRAM_ENDED_UNEXPECT	5054	System program ended unexpectedly.
EHNTF_SYS_SECURITY_ERROR	5056	Security error occurred for system.
EHNTF_SYS_NOT_SUPPORTED	5058	System is not a supported system.
EHNTF_WRONG_PROGRAM_VERSION	5060	&&&&&&&&& must be Version &.& Release &.& Modification &&.
EHNTF_SYS_CONTACT_ENDED	5062	Contact with system ended.
EHNTF_SYS_CONTACT_INTERRUPTED	5064	Contact with system temporarily interrupted.
EHNTF_SYS_RESOURCE_FAILURE	5066	Resource failure on system.
EHNTF_COV_ENDED_UNEXPECTEDLY	5067	Conversation was ended unexpectedly.
EHNTF_INSUFFICIENT MEMORY	5071	Not enough memory.
EHNTF_PCSWIN_NOT_FOUND	5310	PCSWIN not loaded.
EHNTF_INSUFFICIENT_MEMORY	5311	DLL does not support Windows real mode.
EHNTF_OPERATING_SYSTEM_ERROR	5400	Operating system error.
EHNTF_UNDEFINED_ERROR	9999	Undefined error.

Chapter 22. Submit Remote Command Windows Application Program Interface

The submit remote command API allows work stations to start noninteractive programs and procedures on the AS/400 system and to receive completion messages. Any AS/400 batch command or program may be run. Up to ten reply messages can be sent by the AS/400 command or program. However, program data cannot be returned through this interface.

This chapter describes the individual routines and return codes which make up the submit remote command Windows API. The descriptions make reference to predefined constants and function prototypes which can be found in the file EHNSRW.H in the PC Support tools folder (QIWSTOOL). The PC Support tools folder also contains a sample program which demonstrates how to use the submit remote command Windows API. For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

For information on the requirements for developing and running applications that use the PC Support Windows APIs, see "Requirements for Using Windows APIs" on page 19-9.

Submit Remote Command Windows API Routines

The following discussions of each submit remote command Windows API routine describe in detail:

- Purpose
- Procedure declaration
- Parameters
- Return codes

EHNSR_GetMessage

Purpose

This function should be called after EHNSR_SubmitCommand. It extracts a message from the reply message buffer that was passed on the EHNSR_SubmitCommand call. Each time this API is called, the next message is retrieved.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSRW.H"
word far pascal EHNSR_GetMessage(
                HWND            hWnd,
                LPSTR           lpszSysName,
                LPWORD          lpwSevCode,
                LPSTR           lpszMsgID,
                LPSTR           lpszMsg,
                WORD            wMsgLen,
                LPSTR           lpszMsgBuf,
                WORD            lpwMsgBufLen);
```

Parameters

hWnd	Identifies the current window of the application.
lpszSysName	Points to a null-terminated character string which specifies a system name. The maximum length of this string is 8 bytes (excluding the null terminator).
lpwSevCode	Points to a variable that is used to return the severity code of the message to which the EHNSR_SubmitCommand was submitted.
lpszMsgID	Points to a character buffer that is used to return the message identifier. The message identifier is returned as a null-terminated character string. This buffer must have a length of 8 bytes.
lpszMsg	Points to a character buffer that is used to return the message text. The message text is returned as a null-terminated character string. If the reply message buffer is empty, an empty string is returned.
wMsgLen	Identifies the size of the message text buffer pointed to by the lpszMsg parameter.
lpszMsgBuf	Points to the reply message buffer that was passed on the EHNSR_SubmitCommand call.
lpwMsgBufLen	Identifies the size of the reply message buffer that was passed on the EHNSR_SubmitCommand call.

Return Codes

For return codes, see "Return Codes for the Submit Remote Command Windows API" on page 22-4. If the buffer is too small to hold the entire message, the message will be truncated. In this situation EHNSR_MSGTRUNCATE(002B) is returned even though the API has been executed successfully.

EHNSR_StopConversation

Purpose

This function stops the conversation with the remote system. If no conversation is active, no action is taken and the return code indicates the call was successful.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSRW.H"
word far pascal EHNSR_StopConversation(
                HWND            hWnd,
                LPSTR           lpszSysName);
```

Parameters

hWnd	Identifies the current window of the application.
lpszSysName	Points to a null-terminated character string that specifies the name of the system for which the conversation should be ended. The maximum length of this string is 8 bytes (excluding the null terminator).

Return Codes

For return codes, see "Return Codes for the Submit Remote Command Windows API" on page 22-4.

EHNSR_SubmitCommand

Purpose

This function submits a command to a remote system. The first time this routine is called, a conversation is allocated. The conversation continues until EHNSR_StopConversation is called or until a communication problem occurs. If a communication problem occurs, the next call to EHNSR_SubmitCommand causes another attempt to allocate a conversation.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSRW.H"
word far pascal EHNSR_SubmitCommand(
                HWND            hWnd,
                LPSTR           lpszSysName,
                LPSTR           lpszCommand,
                LPSTR           lpszMsgBuf,
                WORD            wMsgBufLen,
                FARPROC         lpProcCallBack);
```

Parameters

hWnd	Identifies the current window of the application.
lpszSysName	Points to a null-terminated character string containing the name of the remote system. The maximum length of this string is 8 bytes (excluding the null terminator).
lpszCommand	Points to a null-terminated character string containing the command to submit. The maximum length of this string is 2000 bytes (excluding the null terminator).
lpszMsgBuf	Points to a buffer where reply messages will be placed. The maximum size of this buffer is 64 KB minus 1. See "Reply Message Buffer" on page 13-3 for information about the format of this buffer.
wMsgBufLen	Identifies the size of the reply message buffer.
lpProcCallBack	Identifies the address of a routine within the calling application that the submit remote command DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy submit remote command request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: This parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the submit remote command request is long, a message box should be displayed before calling this routine. The message box indicates the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Submit Remote Command Windows API."

Return Codes for the Submit Remote Command Windows API

Functions for the submit remote command Windows API use the following return codes. The constants are defined in EHNSWR.H.

Table 22-1 (Page 1 of 3). Return Codes for the Submit Remote Command Windows API

Name	Hex Value	Description
EHNSR_OK	0000	Command completed successfully.
EHNSR_WARNING	000A	Warning issued.
EHNSR_ERROR	0014	Error found.

Table 22-1 (Page 2 of 3). Return Codes for the Submit Remote Command Windows API

Name	Hex Value	Description
EHNSR_ACCESSDAMAGE	0015	DDM access damaged file.
EHNSR_SEVEREERROR	0016	Severe error found.
EHNSR_PERMANENTDAMAGE	0017	Damage occurred to permanent objects.
EHNSR_TARGETDAMAGE	0018	DDM damaged for this session.
EHNSR_CMDSTRINGSYNTAX	001F	Command string greater than 2000 bytes.
EHNSR_SYSNAMESYNTAX	0029	System name greater than 8 bytes.
EHNSR_MSGTRUNCATE	002B	Reply message buffer too small.
EHNSR_INACTIVECONVERSATION	002D	No active conversation to remote system.
EHNSR_INVALIDPOINTER	0030	Input pointer to data buffer is NULL.
EHNSR_CMDNOTSUPPORT	0031	DDM command not supported.
EHNSR_CONTACTEND	0032	Contact with remote system ended.
EHNSR_CONVERSATIONEND	0033	Conversation ended unexpectedly.
EHNSR_TARGETDDMEND	0034	DDM unexpectedly ended.
EHNSR_NOSESSIONAVAIL	0035	Conversation allocation failed.
EHNSR_SYSNAMENOTFOUND	0036	Remote system name not found.
EHNSR_NORESOURCEAVAIL	0037	Insufficient system resource.
EHNSR_TEMPINTERRUPT	0038	Contact with system interrupted.
EHNSR_NODDMONSYSTEM	0039	DDM not on remote system.
EHNSR_SECURITYINVALID	003A	User ID and/or password is not valid.
EHNSR_ROUTERNOTLOADED	0046	Router is not loaded.
EHNSR_NOMEMORY	0048	Not enough memory.
EHNSR_INVALIDMSGBUFFER	0049	Reply message buffer not valid.
EHNSR_INVALIDCMDBUFFER	004A	Command string buffer not valid.
EHNSR_INVALIDSYSTEMBUFFER	004B	Remote system buffer not valid.
EHNSR_REQUESTINTERRUPT	0052	System interrupted the request.
EHNSR_INVALIDCONVSTATE	0053	Conversation state not valid.
EHNSR_PARMCHECKERROR	0054	Parameter checking error.
EHNSR_ALLOCATIONERROR	0055	Allocation unexpected error.
EHNSR_UNEXPECTRETURNCODE	0056	Unexpected primary return code.
EHNSR_NOTINREQUESTSTATE	0057	Conversation not in request state.
EHNSR_INVALIDMSGCODEPOINT	0058	Message code point not valid.
EHNSR_WRONGDATALENGTH	005B	Wrong object data length.
EHNSR_UNEXPECTEDERROR	0063	Unexpected error.
EHNSR_NOMOREMESSAGES	FF00	All messages have been processed.
EHNSR_NOROUTERTOSYSTEM	FF01	Router not connected to any system or not connected to specified system.
EHNSR_INVALIDWINDOWHANDLE	FF02	Window handle does not exist or NULL.
EHNSR_INTERNALERROR	FF03	Internal error.

Table 22-1 (Page 3 of 3). Return Codes for the Submit Remote Command Windows API

Name	Hex Value	Description
EHNSR_ENVIRONMENTNOTSUPPORT	FF04	Environment not supported.

Chapter 23. Shared Folders Windows Application Program Interface

The shared folders Windows API provides a set of routines to help manage assigning and releasing shared folders drives. It is specific to shared folders and, therefore, does not interfere with the assignments of other network drives. This API allows up to 26 drive assignments and does not restrict the drive letter range in any way.

This chapter describes the individual routines and return codes which make up the shared folders Windows API. The descriptions make reference to predefined constants, data structures, and function prototypes which can be found in the file EHNSFW.H in the PC Support tools folder (QIWSTOOL). The PC Support tools folder also contains a sample program which demonstrates how to use the shared folders Windows API. For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

For information on the requirements for developing and running applications that use the PC Support Windows APIs, see "Requirements for Using Windows APIs" on page 19-9. To use the shared folders Windows API, the PC Support Shared Folders program (STARTFLR) must also be loaded.

Shared Folders Windows API Routines

The following discussions of each shared folders Windows API routine describe in detail:

- Purpose
- Procedure declaration
- Parameters
- Return codes

EHNSF_AssignFlrDrive

Purpose

This function assigns a drive to an AS/400 folder.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSFW.H"
word far pascal EHNSF_AssignFlrDrive(
                HWND            hWnd,
                char            cDrive,
                LPSTR           lpszSystemName,
                LPSTR           lpszFolderName);
```

Parameters

hWnd	Identifies the current window of the application.
cDrive	Identifies the personal computer drive letter to be assigned.
lpszSystemName	Points to a null-terminated character string identifying the host system. If this is a null pointer or the host name is a zero-length string, the default system is used.
lpszFolderName	Points to a null-terminated character string identifying the folder to assign. If this is a null pointer or the folder name is a zero-length string, the drive is assigned to all folders on the system.

Return Codes

For return codes, see "Return Codes for the Shared Folders Windows API" on page 23-6.

EHNSF_FindAvailDrive

Purpose

This function returns the first available nonlocal drive letter that can be assigned as an AS/400 shared folders drive.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSFW.H"
char far pascal EHNSF_FindAvailDrive(
                HWND           hWnd,
                char far      *lpcDrive;)
```

Parameters

hWnd	Identifies the current window of the application.
lpcDrive	Points to a character variable which is used to return the drive letter.

Return Codes

For return codes, see "Return Codes for the Shared Folders Windows API" on page 23-6.

EHNSF_GetCapability

Purpose

This function returns the functional level of the shared folders Windows DLL.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSFW.H"
word far pascal EHNSF_GetCapability(
                HWND           hWnd);
```

Parameters

hWnd — Identifies the current window of the application.

Return Codes

The return value is the functional level of the shared folders Windows DLL. Possible values are:

0 The shared folders windows DLL cannot be run in the current environment. Some possible reasons are:
- Shared folders is not running.
- Shared folders is running, but it is not at a level which supports any of the function provided by this DLL. Shared folders must be at Version 2 Release 2 or higher.
- Windows is running in real mode.

1 The shared folders Windows DLL contains support for the following entry points:

EHNSF_AssignFlrDrive
EHNSF_FindAvailDrive
EHNSF_GetCapability
EHNSF_GetFlrDesc
EHNSF_QueryAssignedFlrDrive
EHNSF_QueryDriveStatus
EHNSF_ReleaseFlrDrive

> 1 These values are reserved for future enhancements of the shared folders Windows DLL.

EHNSF_GetFlrDesc

Purpose

This function returns the description of an AS/400 folder.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSFW.H"
word far pascal EHNSF_GetFlrDesc(
                HWND          hWnd,
                char          cDrive,
                LPSTR         lpszFolderName,
                LPSTR         lpszDescBuf);
```

Parameters

hWnd — Identifies the current window of the application.

cDrive — Identifies the personal computer drive letter to be queried.

lpszFolderName — Points to a null-terminated character string that identifies the name of the folder for which the description is requested.

lpszDescBuf — Points to a character buffer that is used to return the folder description. The buffer must be at least 45 bytes in length. The description is returned in the buffer as a null-terminated character string.

Return Codes

For return codes, see "Return Codes for the Shared Folders Windows API" on page 23-6.

EHNSF_QueryAssignedFlrDrive

Purpose

This function queries a drive letter to determine whether it is assigned to an AS/400 folder. If the drive is assigned, this routine optionally returns the name of the folder and the name of the system associated with the assigned drive.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSFW.H"
word far pascal EHNSF_QueryAssignedFlrDrive(
                HWND          hWnd,
                char          cDrive,
                LPSTR         lpszSystemName,
                LPSTR         lpszFolderName);
```

Parameters

hWnd — Identifies the current window of the application.

cDrive — Identifies the personal computer drive letter to be queried.

lpszSystemName — Points to a character buffer that is used to return the name of the system the drive is assigned to. The buffer must be at least 128 bytes in length. The system name is returned in the buffer as a null-terminated character string. If the pointer is null, the system name is not returned.

lpszFolderName — Points to a character buffer that is used to return the name of the folder the drive is assigned to. The buffer must be at least 128 bytes in length. The folder name is returned in the buffer as a null-terminated character string. If the pointer is null, the folder name is not returned.

Return Codes

For return codes, see "Return Codes for the Shared Folders Windows API" on page 23-6.

EHNSF_QueryDriveStatus

Purpose

This function determines if each drive in the range A: to Z: is in use by another network driver and is:

- Is in use by another network driver and is unavailable to be assigned as a shared folders drive
- Available to be assigned as a shared folders drive
- Already assigned as a shared folders drive

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSFW.H"
word far pascal EHNSF_QueryDriveStatus(
                HWND           hWnd,
                char far      *lpcDriveTable);
```

Parameters

hWnd — Identifies the current window of the application.

lpcDriveTable — Points to a 26-character array that is used to return the status each drive. Possible values are:

EHNSF_CANNOTASSIGN (X'30')

The drive letter is in use by another network driver and is not available to be assigned as a shared folders drive.

EHNSF_CANASSIGN (X'31')

The drive letter is available to be assigned as a shared folders drive.

EHNSF_ASSIGNED (X'32')

The drive letter is already assigned as a shared folders drive.

Return Codes

For return codes, see "Return Codes for the Shared Folders Windows API."

EHNSF_ReleaseFlrDrive

Purpose

This function releases a drive which is assigned to an AS/400 folder.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNSFW.H"
word far pascal EHNSF_ReleaseFlrDrive(
                HWND            hWnd,
                char            cDrive);
```

Parameters

hWnd	Identifies the current window of the application.
cDrive	Identifies the personal computer drive letter to be released. If this value is EHNSF_RELEASEALL(*), all assigned drives are released.

Return Codes

For return codes, see "Return Codes for the Shared Folders Windows API." If all drives are to be released and at least one drive cannot be released, the return code is for the last drive that cannot be successfully released.

Return Codes for the Shared Folders Windows API

Functions in the shared folders Windows API use the following return codes. These are constants defined in EHNSFW.H.

Table 23-1. Return Codes for the Shared Folders Windows API

Name	Hex Value	Description
EHNSF_SUCCESS	0000	Command completed successfully.
EHNSF_INVALIDFUNCTION	0001	Invalid function.
EHNSF_FILENOTFOUND	0002	File not found.
EHNSF_PATHNOTFOUND	0003	Path not found.
EHNSF_ACCESSDENIED	0005	Access denied.
EHNSF_ENVIRONMENTERROR	000A	Shared folders API does not support the current environment (Windows may be running in real mode).
EHNSF_INVALIDDRIVE	000F	Invalid drive.
EHNSF_NOMOREFILES	0012	No more files.
EHNSF_GENERALFAILURE	001F	General failure.
EHNSF_NETNOTSUPPORTED	0032	Network request not supported.
EHNSF_NOROUTER	0033	Router not found.
EHNSF_SYSTEMNAMENOTFOUND	0035	System name not found.
EHNSF_UNEXPECTNETERROR	003B	Unexpected network error.
EHNSF_NETACCESSDENIED	0041	Network access denied.
EHNSF_FOLDERNAMENOTFOUND	0043	Folder name not found.
EHNSF_NETNAMELIMIT	0044	Name too long.
EHNSF_TEMPCANNOTASSIGN	0048	Assign function temporarily suspended.
EHNSF_TOOMANYASSIGNED	0054	Maximum number of drives already assigned.
EHNSF_ALREADYASSIGNED	0055	Drive already assigned.
EHNSF_INVALIDPASSWORD	0056	Invalid password.
EHNSF_INVALIDPARAMETER	0057	Invalid parameter.
EHNSF_INVALIDHOSTLEVEL	005A	Host system does not support function.
EHNSF_BUFFEROVERFLOW	006F	Buffer overflow.
EHNSF_INVALIDNAME	007B	Invalid name.
EHNSF_INVALIDLEVEL	007C	DLL is not at a functional level to support this API.

Chapter 24. Virtual Printer Windows Application Program Interface

The virtual print API offers PC applications access to AS/400 printers. Virtual printers may be assigned and released using this API. The API supports up to nine printer assignments.

This chapter describes the individual routines, data structures, and return codes which make up the virtual printer Windows API. The descriptions make reference to predefined constants, data structures, and function prototypes which can be found in the files EHNVPWIN.H and EHNCLHST.H in the PC Support tools folder (QIWSTOOL). The PC Support tools folder also contains a sample program which demonstrates how to use the virtual printer Windows API. For more information about the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

For information on the requirements for developing and running applications that use the PC Support Windows APIs, see "Requirements for Using Windows APIs" on page 19-9.

To use the virtual printer function Windows DLL, the virtual print program VPRT.EXE must be loaded before Windows is started.

Virtual Printer Windows API Routines

The following discussions of each virtual printer Windows API routine describe in detail:

- Purpose
- Procedure declaration
- Parameters
- Return codes

EHNVP_AssignVP

Purpose

This function assigns a network printer to an AS/400 system.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_AssignVP(
                    HWND              hWnd,
                    LPVP_ASSIGN       lpAssignData,
                    LPSTR             lpszLocalName);
```

Parameters

hWnd	Identifies the current window of the application.
lpAssignData	Points to a VP_ASSIGN structure. This structure specifies the options to use when assigning the virtual printer. It is also used to return the ID associated with the assigned printer. For a description of the VP_ASSIGN structure, see "VP_ASSIGN" on page 24-15.
lpszLocalName	Points to a null-terminated character string which identifies the local device to redirect. Valid local device names are LPT1 through LPT9.

Notes:

1. If szPrtFile is specified, the szPrtFileLib field identifies the library where the printer file exists. If the library is not specified, *USERLIBL is used.
2. If the System Name field of the DeviceData field is null, the default system is used.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

EHNVP_BuildList

Purpose

This function builds one of the following lists:

- A list of printers on an AS/400 system
- A list of all print files (or all print files matching a specified wildcard criteria) in the libraries of the library list of the caller
- A list of all print files (or all print files matching a specified wildcard criteria) in a specified library
- A list of all libraries in the user library list (or matching a specified wildcard criteria)

This function must be used before the EHNVP_QueryListHead and EHNVP_GetListItem functions can be used.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_BuildList(
                    HWND            hWnd,
                    WORD            wListType,
                    LPVP_BUILDLIST  lpListParms);
```

Parameters

hWnd — Identifies the current window of the application.

wListType — Specifies the type of list request. Possible values are:

- EHNVP_LTPRTLIST (X'0000') printer list
- EHNVP_LTLIBLIST (X'0001') library list
- EHNVP_LTPRTFLIST (X'0002') printer files list

lpListParms — Points to a VP_BUILDLIST structure. This structure specifies the options to use when building the list. It is also used to return the information about the generated list. For a description of the VP_BUILDLIST structure see "VP_BUILDLIST" on page 24-17.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

EHNVP_ChgTransTable

Purpose

This function changes the translation table used by a virtual printer.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"

word far fascal EHNVP_ChgTransTable(HWND          hWnd,
                                    unsigned short wVPid,
                                    char far      *lpcTransTable);
```

Parameters

hWnd — Identifies the current window of the application.

wVPid, — The ID for the virtual printer to be changed.

lpcTransTable — A far pointer to a 256 byte translation table used for converting ASCII to EBCDIC.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

EHNVP_CloseJob

Purpose

This function closes a print job on a virtual printer.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPOS2.H"

word far pascal EHNVP_CloseJob(
      HWND            hWnd,
      unsigned short  wVPid);
```

Parameters

hWnd Identifies the current window of the application.

wVPid The ID of the virtual printer.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

Additional Information

This call is used to close a print job on a virtual printer that has the timeout value set to zero.

EHNVP_GetErrorText

Purpose

This function retrieves any error text from virtual printer network errors and clears the error text buffer of the DLL. The error for which the text is retrieved is one of the following:

- The last virtual print API request that had a return code of EHNVP_NETERROR (X'0007')
- The last EHNVP_GetAsnStat request that returned a status with the EHNVP_ERRPENDING bit turned on.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_GetErrorText(
                  HWND              hWnd,
                  LPSTR             lpszErrBuffer);
```

Parameters

hWnd — Identifies the current window of the application.

lpszErrBuffer — Points to a character buffer that is used to store the retrieved error text. This buffer must be 514 bytes in length. The error text is stored in this buffer as a null-terminated character string.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

EHNVP_GetListItem

Purpose

This function retrieves the next item in a list built by the EHNVP_BuildList API.

Note: EHNVP_BUILDLIST must be called before calling this API.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_GetListItem(
                    HWND            hWnd,
                    WORD            wListID,
                    LPSTR           lpszItemBuffer);
```

Parameters

hWnd — Identifies the current window of the application.

wListID — Specifies the list ID returned in the wListID field of the VP_BUILDLIST structure from the call to EHNVP_BUILDLIST.

lpszItemBuffer — Points to a buffer where the list item is to be stored. The buffer must be at least usRecLength bytes in length (returned in the VP_BUILDLIST structure on the EHNVP_BuildList call). The list item is stored in the buffer as a null-terminated character string. The characters up to the first blank contain the name of the printer, print file, or printer file library. Any additional characters provide more information about the item.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21. If the return value is EHNVP_BADLISTID (X'0013'), the ID is not valid or the list has been corrupted.

EHNVP_GetRedirList

Purpose

This function, when called repeatedly, returns status information for a list of AS/400 network printers. The network printer is identified by the value specified for the wIndex parameter. If an entry in the current list of redirected printers is found, the current status of the AS/400 network printer and its assigned parameter values are returned.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_GetRedirList(
                    HWND              hWnd,
                    WORD              wIndex,
                    LPVP_REDIRINFO    lpRedirItem);
```

Parameters

hWnd	Identifies the current window of the application.
wIndex	Specifies the list position of the redirected printer device for which status information is needed. A value of 0 indicates the first device, a value of 1 indicates the second device, and so on.
lpRedirItem	Points to a VP_REDIRINFO structure. This structure is used to return information about the network printer. For more information about the VP_REDIRINFO structure, see "VP_REDIRINFO" on page 24-18.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21. This function returns EHNVP_ENDOFLIST when there are no more redirected devices.

EHNVP_QueryAsnParms

Purpose

This function returns detailed status information regarding the network printer assignment for the specified printer ID.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_QueryAsnParms(
                    HWND              hWnd,
                    LPWORD            lpwVPid,
                    LPVP_ASNSTATUS    lpStatusData);
```

Parameters

hWnd	Identifies the current window of the application.
lpwVPID	Specifies the ID of the virtual printer to query. This is the ID returned when the printer was assigned. This value can also be obtained by calling EHNVP_QryAsnStatus or EHNVP_GetRedirList.
lpStatusData	Points to a VP_ASNSTATUS structure. This structure is used to return the status of the network printer. For a description of the VP_ASNSTATUS structure, see "VP_ASNSTATUS" on page 24-15.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

EHNVP_QueryAsnStatus

Purpose

This function returns the ID and current status of the specified printer (if a device is assigned as a virtual printer).

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_QueryAsnStatus(
                    HWND            hWnd,
                    LPSTR           lpszLocalDevice,
                    LPWORD          lpwVPid,
                    LPWORD          lpwVPStatus);
```

Parameters

hWnd	Identifies the current window of the application.
lpszLocalDevice	Points to a null-terminated character string that identifies the local device being queried. Valid local device names are LPT1 through LPT9.
lpwVPID	Points to an integer variable which is used to return the virtual printer ID. This ID can be used as input to the EHNVP_QueryAsnParms call to get the full assignment information.
lpwVPStatus	Points to an integer variable where the status of the local device is to be stored. The following bits may be turned on: EHNVP_ASSIGNED (X'0001') EHNVP_ERRPENDING (X'0004') EHNVP_SUSPENDED (X'0008') EHNVP_DATAPASS (X'0010')

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

EHNVP_QueryCapability

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_QueryCapability(
                    HWND          hWnd);
```

Parameters

hWnd — Identifies the current window of the application.

Return Codes

The return value is the functional level of the virtual printer Windows API. EHNVP_CAPLVL1 is defined in EHNVPWIN.H. The original version of this DLL always returns a value of EHNVP_CAPLVL1 (X'0000') on this call.

EHNVP_QueryErrorText

Purpose

This function retrieves error text for virtual printer network errors. The error for which the text is retrieved is one of the following:

- The last virtual printer API request had a return code of EHNVP_NETERROR (X'0007')
- The last EHNVP_GetAsnStat request that returned a status with the EHNVP_ERRPENDING bit turned on.

The error message buffer is not cleared by this call; if you call this function twice, the same message text is returned both times.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_QueryErrorText(
                    HWND              hWnd,
                    LPSTR             lpszErrBuffer);
```

Parameters

hWnd — Identifies the current window of the application.

lpszErrBuffer — Points to a character buffer which is used to store the retrieved error text. This buffer must be 514 bytes in length. The error text is stored in this buffer as a null-terminated character string.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

EHNVP_QueryListHead

Purpose

This function retrieves the heading of a list built by the EHNVP_BuildList function.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_QueryListHead(
                    HWND            hWnd,
                    WORD            wListID,
                    LPSTR           lpszHeadBuffer);
```

Parameters

hWnd	Identifies the current window of the application.
wListID	Specifies the list ID returned in the wListID field of the VP_BUILDLIST structure from the call to EHNVP_BUILDLIST.
lpszHeadBuffer	Points to a buffer where the heading of the list is to be stored. This buffer must be at least usRecLength bytes in length (returned by EHNVP_BUILDLIST structure on the EHNVP_BuildList call). The heading is stored in the buffer as a null-terminated character string. This heading contains a text description of the content of each item in the list. A programmer may choose to use this for display purposes.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21. See "Return Codes for the Virtual Printer Windows API" on page 24-21. If the return value is EHNVP_BADLISTID, the ID is not valid or the list has been corrupted.

EHNVP_QueryVPStatus

Purpose

This function returns the status of the virtual print program.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_QueryVPStatus(
                    HWND            hWnd,
                    LPVP_VPSTATUS   lpVPStatus);
```

Parameters

hWnd	Identifies the current window of the application.
lpVPStatus	Points to a VP_VPSTATUS structure. This structure is used to store the status of the virtual print program. For a description of the VP_VPSTATUS structure, see "VP_VPSTATUS" on page 24-18.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

EHNVP_ReleaseVP

Purpose

This function releases an assigned AS/400 network printer.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_ReleaseVP(
                    HWND            hWnd,
                    LPWORD          wVPid);
```

Parameters

hWnd	Identifies the current window of the application.
wVPID	Specifies the ID of the virtual printer to be released. This is the ID returned when the printer was assigned. This value can also be obtained by calling EHNVP_QueryAsnStatus or EHNVP_GetRedirList.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

EHNVP_ResetParms

Purpose

This function resets some of the parameters for a currently assigned printer.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_ResetParms(
                    HWND            hWnd,
                    WORD            wVPid,
                    LPVP_ASNPARMS   lpNewParms);
```

Parameters

hWnd — Identifies the current window of the application.

wVPID — Specifies the ID of the virtual printer for which the parameters are to be reset. This is the ID returned when the printer was assigned. This value can also be obtained by calling EHNVP_GetRedirList.

lpNewParms — Points to a VP_ASNPARMS structure. This structure is used to specify the new parameter values. For a description of the VP_ASNPARMS structure, see "VP_ASNPARMS" on page 24-12.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

Additional Information

The following fields may be changed by this API:

- usCopies
- usTimeout
- fDeferPrint
- fAppFormData
- usACharSet
- cUntransChar
- usTOS (DBCS environment only)
- usFormFeed

EHNVP_VerifyAsnDevice

Purpose

This function verifies that a printer, library, and printer file are valid. If the objects are valid, the function returns information about them.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNVPWIN.H"
word far pascal EHNVP_VerifyAsnDevice(
                    HWND                hWnd,
                    LPVP_VERIFYREQ      lpDeviceData,
                    LPVP_VERIFYOUT      lpVerifyOutput);
```

Parameters

hWnd — Identifies the current window of the application.

lpDeviceData — Points to a VP_VERIFYREQ structure. This structure is used to identify the AS/400 system name, printer, library and printer file to verify. For a description of the VP_VERIFYREQ structure, see "VP_VERIFYREQ" on page 24-19.

Notes:

1. If szPrtFile is specified, the szPrtFileLib field identifies the library where the printer file exists. If the library is not specified, *USERLIBL is used.
2. If the szSystem field is null, the default system is used.
3. On input, the szOutQue and szOutQueLib parameters are reserved and should be set to NULL.

lpVerifyOutput Points to a VP_VERIFYOUT structure. This structure is to be used to store the printer type default, and possible values for assign options, if the specified objects are valid. For a description of the VP_VERIFYOUT structure, see "VP_VERIFYOUT" on page 24-20.

Note: Both the szOutQue and szOutQueLib fields are updated with the output queue and output queue library that the data will be spooled to on the AS/400 system. These values should be passed back on an Assign request. The output queue name and the output queue library name are not provided if you are attempting to assign a virtual printer to an AS/400 system running in a version or release of the OS/400 program before Version 2 Release 2.

Return Codes

For return codes, see "Return Codes for the Virtual Printer Windows API" on page 24-21.

Virtual Printer Windows API Structures

VP_ASNPARMS

Description

This structure is used by the EHNVP_ResetParms and EHNVP_AssignVP API routines. The structure is defined as follows:

```
typedef struct _VP_ASNPARMS {
   unsigned short  usSize;        //(in)  - length in bytes
   char szPrinterType[EHNVP_PRTTYPELEN + 1];  // Printer type
   unsigned short  usDataType;    //Data type
   unsigned short  usCopies;      //number of Copies
   unsigned short  usTimeout;     //printer time out in seconds
   BOOL            fDeferPrint;   //defer printing
   unsigned short  usCPI;         //default CPI*10
   unsigned short  usCPL;         //default Chars Per Line
   unsigned short  usLPI;         //default LPI*10
   unsigned short  usPageLen;     //default Page Length
   unsigned short  usLPP;         //default LinesPerPage
   BOOL            fAppFormData;  //Application formatted data
   unsigned short  usACharSet;    //ASCII Character Set
   BYTE            bUntransChar;  //Untranslatable Character
   char            achReserved[4];
   unsigned short  usTOS;         //Target Operating System
   unsigned short  usFormFeed;    //Append Form Feed
   } VP_ASNPARMS;
```

Field	Description
usSize	Identifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
szPrinterType	Specifies the printer type (for example, 3812). This is a null-terminated character string.
usDataType	Identifies the data type the virtual printer will be handling. Possible values are: EHNVP_DTSCS (X'0001') EHNVP_DTCONVRT (X'0002') EHNVP_DTFFT (X'0003') EHNVP_DTASCII (X'0004') EHNVP_DTAFPDS (X'0005')
usCopies	Identifies the number of copies requested. Possible values are 1 to 255. If the usDataType field is EHNVP_DFTFT, the allowed values are 1 to 99.
usTimeOut	Identifies the printer time-out value associated with this printer. The time-out value is the number of seconds the virtual printer is to wait after it stops receiving data before it closes the output file. A value of zero (0) indicates no time-out. Any other value represents the value in seconds. Possible values are 0 to 255.
fDeferPrint	Indicates whether the output file on the AS/400 system must be closed before printing can start. Possible values are True or False.
usCPI	Identifies the number of characters per inch multiplied by 10 (for example, 16.7 would be 167). The possible values are returned on the EHNVP_VerifyAsnDevice API routine.
usCPL	Identifies the number of characters per line. Possible values are 1 to 378.
usLPI	Identifies the number of lines per inch to print. Possible values are returned by the EHNVP_VerifyAsnDevice API routine.

usPageLen — Identifies the length of the paper in lines. Possible values are 1 to 255.

usLPP — Identifies the number of lines printed per page. Possible values are 1 to the value specified for the usPageLen parameter.

fAppFormData — Identifies whether the data was formatted by an application. Possible values are True or False. If True, virtual print will assume the formatting is correct and will allow the values of characters per inch (CPI), lines per inch (LPI), lines per page (LPP), and Page Length to be changed by the data stream. If False, virtual print will ignore any attempts to change these values.

usACharSet — Identifies the ASCII character set to use. Character set 2 has printable characters at code points from X'80' to X'9F'. Character set 1 has printer controls in this range. Possible values are:

EHNVP_CHARSET1 (X'0001')
EHNVP_CHARSET2 (X'0002')

bUntransChar — Identifies the untranslatable character substitute character. This character is substituted for characters that are not found in the ASCII-to-EBCDIC table.

usTOS — Indicates the target operating system on which the print emulator is running. The possible values are:

EHNVP_TOSDOS (X'0001')
EHNVP_TOSOS2 (X'0002')
EHNVP_TOSDOSV (X'0003')

Note: This field is ignored in single-byte character set data.

usFormFeed — Identifies whether the virtual printer should add a form feed control character to the end of each job. The possible values are:

EHNVP_FFYES (X'0001')
EHNVP_FFNO (X'0002')
EHNVP_FFCON (X'0003')

Note: If the value is ENHVP_FFCON, the virtual printer adds a form feed to the end of a job only if the last byte of the job is not a form feed control character.

Notes:

1. For the EHNVP_AssignVP API, all of these fields should be filled in by the caller.
2. For the EHNVP_ResetParms API, the usSize, fDeferPrint, usTimeOut, and usCopies fields should be filled in by the caller. All other fields are ignored.

VP_ASNSTATUS

Description

This structure is used by the EHNVP_QueryAsnParms API routine. The structure is defined as follows:

```
typedef struct _VP_ASNSTATUS {
   unsigned short usSize;
   WORD           wAnsStatus;
   VP_ASSIGN    AssignInfo;
   VP_PRTSTAT   PrintStats;
} VP_ASNSTATUS;
```

Field	Description
usSize	Identifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
wAnsStatus	Is used to return flags that indicate the assign status for this virtual printer. The following bits may be turned on: EHNVP_ASSIGNED (X'0001') EHNVP_ERRPENDING (X'0004') EHNVP_SUSPENDED (X'0008') EHNVP_DATAPASS (X'0010') If the EHNVP_ASSIGNED bit is off, none of the other bits are valid.
AssignInfo	Is used to return the assignment information.
PrintStats	Is used to return the printer statistics.

VP_ASSIGN

Description

This structure is used by the EHNVP_AssignVP and EHNVP_QueryAsnParms API routines.

```
typedef struct _VP_ASSIGN {
   unsigned short      usSize;
   WORD                wVPID;
   VP_VERIFYREQ        DeviceData;
   VP_ASNPARMS         AssignParms;
} VP_ASSIGN;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
wVPID	Specifies the ID of the virtual printer. This field is filled in by the virtual print program when the device is successfully assigned.
DeviceData	Contains the information on where the local port is to be redirected.

AssignParms This structure contains the assign parameters. Some of the parameters are ignored based on the value given for the usDataType field.

The following table shows which parameters must be specified when assigning a virtual printer for each of the possible data types (SCS, convert ASCII to SCS, final form text, ASCII, and advanced function printing data stream).

Table 24-1. Assign Parameters Used with Data Types

Parameter	Data Type 1	Data Type 2	Data Type 3	Data Type 4	Data Type 5
usTimeOut	X	X	X	X	X
usCopies	X	X	X	X	X
fDeferprint	X	X		X	X
usCPI	X	X			
usCPL	X	X			
usLPI	X	X			
usPageLen	X	X			
usLPP	X	X			
fAppFormData		X			
usACharSet		X			
cUntransChar		X			
usTOS				X	
usFormFeed				X	

Notes:

1. The usSize field for the AssignParms parameter must be filled in by the caller with the size of the entire VP_ASNPARMS structure in bytes, regardless of the data type.
2. For the EHNVP_AssignVP API, the wVPID field is filled in as a result of the API call. All other fields should be filled in by the caller.
3. For the EHNVP_QueryAsnParms API, the usSize field should be filled in by the caller. All other fields are filled in as a result of the API call.

VP_BUILDLIST

Description

This structure is used by the EHNVP_BuildList API routine. The structure is defined as follows:

```
typedef struct _VP_BUILDLIST  {
   unsigned short   usSize;
   WORD             wListID;
   unsigned short   usRecLength;
   char szSystem[EHNCL_L_SYSNAME + 1];
   char szPrtFileLib[EHNCL_L_LIBNAME + 1];
   char szPrtFile[EHNCL_L_FILENAME + 1];
} VP_BUILDLIST;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
wListID	Is used to return an ID of the list that is needed for the EHNVP_QueryListHead and EHNVP_GetListItem APIs.
usRecLength	Is used to return the maximum length of records in the list. This is the size of the buffer needed when calling EHNVP_QueryListHead and EHNVP_GetListItem.
szSystem	Specifies a null-terminated AS/400 system name. If the name has a length of zero, the default system is used.
szPrtFileLib	Specifies the name of the library to search for printer files. This field is ignored if the list type is not EHNVP_LTPRTFLIST or EHNVP_LTLIBLIST.
szPrtFile	Specifies a generic printer file name used to filter which printer files should be put on the list. This field is ignored if the list type is not EHNVP_LTPRTFLIST.

VP_PRTSTAT

Description

This structure is used by the EHNVP_QueryAsnParms API routine. The structure is defined as follows:

```
typedef struct _VP_PRTSTAT  {
   unsigned short   usSize;
   unsigned short   usPrintedFiles;
   unsigned short   usPages;
   unsigned short   usUntransChars;
} VP_PRTSTAT;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.

usPrintedFiles	Is used to return the number of files that have been printed through this virtual printer since it was assigned.
usPages	Is used to return the number of pages in the last job.
usUntransChars	Is used to return the number of untranslatable characters in the last job.

VP_REDIRINFO

Description

This structure is used by the EHNVP_GetRedirList API routine. The structure is defined as follows:

```
typedef struct _VP_REDIRINFO  {
   unsigned short usSize;
   WORD           wVPid;
   WORD           wAnsStatus;
   char szLocalName[EHNVP_MAXLOCNAME + 1];
   VP_VERIFYREQ DeviceData;
} VP_REDIRINFO;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
wVPID	Is used to return the ID of the local printer to be used with other API calls.
wAnsStatus	Is used to return flags indicating the current status of the device. Possible values can be one or more of the following: • EHNVP_ASSIGNED (X'0001') • EHNVP_ERRPENDING (X'0004') • EHNVP_SUSPENDED (X'0008') • EHNVP_DATAPASS (X'0010')
szLocalName	Is used to return the local device that is redirected. The name is returned as a null-terminated character string.
DeviceData	Is used to return the remote device that the local device is redirected to.

VP_VPSTATUS

Description

This structure is used by the EHNVP_QueryVPStatus API routine. The structure is defined as follows:

```
typedef struct _VP_VPSTATUS {
   unsigned short usSize;
   unsigned short usNumVPs;
   unsigned short usAPILevel;
   char           achVerRelLev[EHNVP_VRLSIZE];
}  VP_VPSTATUS;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
usNumVPs	Is used to return the number of assigned virtual printers.
usAPILevel	Is used to return the level of support for the API. For Version 2 Release 2, this is 1.
achVerRelLev	Is used to return the 6 ASCII characters that identify the version, release, and level of the virtual print Windows API program.

VP_VERIFYREQ

Description

This structure is used by the EHNVP_GetRedirList, EHNVP_AssignVP, and EHNVP_VerifyAsnDevice API routines. The structure is defined as follows:

```
typedef struct _VP_VERIFYREQ {
   unsigned short usSize;
   char szSystem[EHNCL_L_SYSNAME + 1];
   char szPrinter[EHNCL_L_PRTNAME + 1];
   char szPrtFileLib[EHNCL_L_LIBNAME + 1];
   char szPrtFile[EHNCL_L_FILENAME + 1];
   char szOutQueLib[EHNCL_L_LIBNAME + 1];
   char szOutQue[EHNCL_L_QUENAME + 1];
} VP_VERIFYREQ;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
szSystem	Is the name of the AS/400 system. This name is a null-terminated character string.
szPrinter	Is the AS/400 printer device name. This name is a null-terminated character string.
szPrtFileLib	Is the name of the library for the szPrtFile field. This name is a null-terminated character string.
szPrtFile	Is the name of the printer file to use for printing. This name is a null-terminated character string.
szOutQuelib	Identifies the name of the library containing the output queue identified in the szOutQue field. This name is a null-terminated character string.
szOutQue	Is the name of the output queue on the AS/400 system where data is being sent. This name is a null-terminated character string.

Notes:

1. For the EHNVP_AssignVP API, all of these fields should be filled in by the caller.
2. For the EHNVP_GetRedirList API, the usSize field should be filled in by the caller. All other fields are filled in as a result of the API call.

VP_VERIFYOUT

Description

This structure is used by the EHNVP_VerifyAsnDevice API routine. The structure is defined as follows:

```
typedef struct _VP_VERIFYOUT {
   unsigned short  usSize;              //(in)  - length in bytes
   VP_VERIFYREQ    DeviceData;          //actual data
   unsigned short  usDefCPI;            //default CPI
   unsigned short  usDefCPL;            //default page width
   unsigned short  usDefLPI;            //default LPI
   unsigned short  usDefPageLen;        //default paper length
   unsigned short  usDefLPP;            //default lines per page
   unsigned short  usDefCopies;         //default copies
   BOOL            fDefDefer;           //default defer value
   unsigned short  usDefTimeOut;        //default time out
   BOOL            fDefAppFormData;     //default application data
   unsigned short  usDefACharSet;       //default ASCII Character Set
   unsigned short  ausValidCPIs-EHNVP_MAXCPIVAL-;
   unsigned short  ausValidLPIs-EHNVP_MAXLPIVAL-;
   unsigned short  ausValidDataTypes-EHNVP_MAXDATATYPE-;
   char szPrinterType-EHNVP_PRTTYPELEN + 1-;  // Printer type
   BYTE            bDefUntransChar;     //default untranslatable character
   unsigned short  usDefTOS;            //default Target Operating System
   unsigned short  usDefFormFeed;       //default Append Form Feed
} VP_VERIFYOUT;
```

Field	Description
usSize	Specifies the length of this structure in bytes. This field must be filled in by the caller prior to the call.
DeviceData	Returns the actual names of the system, printer file library, and printer file.
usDefCPI	Returns the default characters per inch multiplied by 10.
usDefCPL	Returns the default for the number of characters that are printed on a line.
usDefLPI	Returns the default for the number of lines per inch multiplied by 10.
usDefPageLen	Returns the default for the paper length in lines.
usDefLPP	Returns the default for the number of lines that are printed on a page.

usDefCopies	Returns the default for the number of copies to be made for each job.
fDefDefer	Returns the default for whether the job should not be printed on the AS/400 system until the output file is closed. A nonzero value indicates that the job should be postponed.
usDefTimeOut	Contains the default timeout value.
fDefAppFormData	Returns the default value for the fDefAppFormData field.
usDefACharSet	Returns the default value for the usDefACharSet field.
ausValidCPIs	Returns the valid characters per inch (CPI) values multiplied by 10. If less valid CPI values exist than the array has room for, the extra elements of the array will contain 0s.
ausValidLPIs	Returns the valid lines per inch (LPI) values multiplied by 10. If less valid LPI values exist than the array has room for, the extra elements of the array will contain 0s.
ausValidDataTypes	Returns the valid data type values. Each element in the array may be: • 0 (indicating an unused element) • EHNVP_DTSCS (X'0001') SCS • EHNVP_DTCONVRT X'0002') ASCII to SCS • EHNVP_DTFFT (X'0003') final form text • EHNVP_DTASCII (X'0004') ASCII • EHNVP_DTAFPDS (X'0005') advanced function printing data stream
szPrinterType	Returns the printer type (for example, 3812). This is returned as a null-terminated character string.
bDefUntransChar	Returns the default value for the bUntransChar field.
usDefTOS	Returns the default value for the usTOS field.
usDefFormFeed	Returns the default value for the usFormFeed field.

Return Codes for the Virtual Printer Windows API

Functions in the virtual printer Windows API use the following return codes. These are constants defined in EHNVPWIN.H.

Table 24-2. Return Codes for the Virtual Printer Windows API

Name	Hex Value	Description
EHNVP_SUCCESS	0000	Command completed successfully.
EHNVP_INVALIDPARM	0001	Invalid parameter.
EHNVP_VPNOTSTARTED	0002	Virtual print not started.
EHNVP_VPBACKLEVEL	0003	API not available
EHNVP_NOERRORTEXT	0004	No error text available.
EHNVP_NOTREDIR	0005	Device not redirected.
EHNVP_ALREADYREDIR	0006	Device already redirected.
EHNVP_NETERROR	0007	Error from virtual print.
EHNVP_ROUTERERROR	0008	Error with router DLL.
EHNVP_FUNCTIONBUSY	0009	Function busy.
EHNVP_ENDOFLIST	0012	End of the list.
EHNVP_BADLISTID	0013	Corrupted list.
EHNVP_BADVPID	0014	Virtual printer ID is not valid.
EHNVP_BADLOCALNAME	0015	Local name is not valid.
EHNVP_BADUSSIZE	0016	usSize field is too small.
EHNVP_WINDOWSERROR	0100	Windows error.
EHNVP_MEMERROR	0101	General memory error.
EHNVP_DOSMEMERROR	0102	Memory error below 640K.
EHNVP_BADENVIRONMENT	0103	Invalid Windows environment.
EHNVP_UNEXPECTEDNETERROR	FFFF	Internal error.

Chapter 25. Remote SQL Windows Application Program Interface

The remote SQL API provides access to AS/400 SQL database files. It provides full AS/400 SQL capability. Applications that need to update records individually should use this API. The remote SQL API also provides a communications mechanism between PC and AS/400 applications.

Note: Before using the APIs in this chapter, review "Remote SQL (RMTSQL) Function Overview" on page 15-1 and "Application Development Using the Remote SQL API" on page 15-3.

For a more detailed discussion on SQL, refer to the *Systems Application Architecture* Structured Query Language/400 Programmer's Guide*, SC41-9609 or to the *SQL/400* Reference*.

This chapter describes the individual routines, data structures, and return codes which make up the remote SQL Windows API. The descriptions make reference to predefined constants, data structures, and function prototypes which can be found in the file EHNRQW.H in the PC Support tools folder (QIWSTOOL). The PC Support tools folder also contains a sample program which demonstrates how to use the remote SQL Windows API. For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

For information on the requirements for developing and running applications that use the PC Support Windows APIs, see Chapter 19, "PC Support/400 Windows Application Program Interface Overview."

Optimal Communication Buffer Size

Remote SQL performance in the Windows environment can be improved by setting the proper communication buffer size. The optimal buffer size for your communication environment can be calculated using the following formula:

```
Optimal_size = comm_buffer_size * (max_buffer_size / comm_buffer_size)
```

- The comm_buffer_size is obtained by calling the EHNAPPC_GetCapabilities API.
- The max_buffer_size is 2K.
- The minimum comm_buffer_size is 276 bytes.
- Optimal_size is an INTEGER value with the maximum value of 2048.

Remote SQL Windows API Routines

The following discussions of each remote SQL Windows API routine are described in detail:

- Purpose
- Procedure declaration
- Parameters
- Return codes

EHNRQ_ATTR

Purpose

This routine returns an array of entries that describe the attributes of the columns that are returned by the currently active SELECT statement. This function can be called after a successful call to EHNRQ_SELECT or EHNRQ_SELECTVAL to determine the attributes of the columns that can be retrieved by calls to EHNRQ_FETCH. There is one entry in the column attributes table for each table column. The information provided is essentially the same as that contained in the SQLDA control block on an EHNRQ_DESC call. However, it is in a simpler form. For example, the SQLLEN field of the SQLDA control block is sometimes the length of the corresponding column data, or is sometimes a combination of the precision and scale attributes. In the column attributes table returned by this entry point, there are separate fields for precision, scale, and data width. Additionally, there are offsets to the beginning of the fields within the result area returned by EHNRQ_FETCH.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_ATTR(
                  HWND              hWnd,
                  hcursor           acursor,
                  Col_Attributes    atable,
                  short _far *      alen);
```

Parameters

hWnd — Identifies the current window of the application.

acursor — Identifies a cursor handle corresponding to an active SELECT cursor (returned by the EHNRQ_SELECT or EHNRQ_SELECTPM call).

atable — Points to a buffer area that is used to store the attribute table. The attribute table is stored in this buffer in the format defined by the Col_Attributes structure. For a description of the Col_Attributes structure, see "Col_Attributes" on page 25-33.

alen — Points to a short integer variable. On input, it specifies the size in bytes of the area pointed to by the atable parameter. If the area is not large enough, this variable is used to store the minimum buffer size needed for this call.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_CLOSE

Purpose

This routine ends a SELECT operation and closes the associated cursor. EHNRQ_CLOSE frees resources (and locks on table rows) in the host system. This function should be called when all of the selected rows have been retrieved through EHNRQ_FETCH or EHNRQ_GETF.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"

extern int pascal far EHNRQ_CLOSE(
                      HWND          hWnd,
                      hcursor       cur,
                      FARPROC       lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

cur — Identifies a cursor handle corresponding to an active SELECT cursor (returned by the EHNRQ_SELECT or EHNRQ_SELECTPM call).

lpProcCallBack — Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_CONNECT

Purpose

Explicitly connects to a remote SQL database and allows remote SQL to use Distributed Relational Data Access (DRDA) support available on the AS/400 system. More setup and usage information on DRDA is available in the *Distributed Relational Database Guide*.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_CONNECT(
                  HWND                          hWnd,
                  char _far *                   connect_stmt,
                  struct Options_Struct _far *  options,
                  short _far *                  opt_len,
                  FARPROC                       lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

stmt — A null-terminated character string that defines the SQL CONNECT statement to be run.

options — Points to the options structure allocated by and filled in by the calling program.

Notes:

1. To have this structure filled in with the current option values, run an EHNRQOPTIONS_QUERY before running the EHNRQCONNECT function.
2. If options are not required, specify a null pointer for the options structure.

See "Options_Struct" on page 25-35 for more information on the options structure.

opt_len — The size in bytes of the area pointed to by the options parameter.

Returned values are:

options — The option structure (if not null) contains the current settings of the options. Refer to "Options_Struct" on page 25-35 for more information on the options structure.

opt_len — If the size of the options structure is too small, this value is set to the minimum size needed in bytes. If the size of the options structure is above the minimum size, this value is set to the size used.

lpProcCallBack — Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be

exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_DELETE

Purpose

This routine deletes the row at the current cursor position. EHNRQ_DELETE must be preceded by a successful EHNRQ_FETCH or EHNRQ_GETF call. In addition, the EHNRQ_SELECT or EHNRQ_SELECTVAL call that returned the specified cursor handle must have specified update mode.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_DELETE(
                  HWND          hWnd,
                  hcursor       cur);
```

Parameters

hWnd — Identifies the current window of the application.

cur — Identifies a cursor handle corresponding to an active SELECT cursor (returned by the EHNRQ_SELECT or EHNRQ_SELECTPM call).

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_DESC

Purpose

This routine obtains the description of the columns identified by a SELECT statement. This routine may be called after a successful call to EHNRQ_SELECT or EHNRQ_SELECTVAL to determine the attributes of the columns, which can be retrieved by calls to EHNRQ_FETCH.

Note: The EHNRQ_ATTR function provides an alternative method of obtaining this same information.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_DESC(
                  HWND                  hWnd,
                  hcursor               cur,
                  struct sqlda _far *   desc_sqlda,
                  short _far *          desc_len);
```

Parameters

hWnd	Identifies the current window of the application.
cur	Identifies a cursor handle corresponding to an active or prepared SELECT cursor (returned by the EHNRQ_SELECT or EHNRQ_SELECTPM call).
desc_sqlda	Points to a buffer area allocated by the application that is used to store the column descriptions. The column descriptions are stored in this buffer in the format defined by the SQLDA structure. For a description of the SQLDA structure, see "sqlda" on page 25-39.
desc_len	Points to a short integer variable. On input, it specifies the size in bytes of the buffer pointed to by the desc_sqlda parameter. If the buffer is not large enough, this variable is used to store the minimum buffer size needed for this call.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_END

Purpose

This function ends the remote SQL environment. EHNRQ_END should be called before ending an application that uses remote SQL or before restarting the remote SQL environment with another EHNRQ_START or EHNRQ_StartSec call.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"

extern pascal far EHNRQ_END(
                  HWND          hWnd,
                  FARPROC       lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

lpProcCallBack — Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40. If the return code is EHNRQ_NOT_INIT (0x0014), remote SQL is not active and cannot be ended.

EHNRQ_ERROR

Purpose

This routine returns the APPC major and minor error return codes associated with the last communication attempt with the host system. Normally, EHNRQ_ERROR is issued to obtain the return codes for logging, or for use in an error message to indicate that an error had occurred during communications with the host system. For example, it may be called after receiving an EHNRQ_COMM_ERROR (0x0020) return code from a previous remote SQL request.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern void extentry EHNRQ_ERROR(
                    HWND            hWnd);
                    char _far *     primary,
                    char _far *     secondary);
```

Parameters

hWnd — Identifies the current window of the application.

primary — Points to a character buffer that is used to store a 4-byte string containing the APPC major return code from the last communications operation. The return code is not ended with a null character. This buffer must be at least 4 bytes in length.

secondary — Points to a character buffer that is used to store an 8-byte string containing the APPC minor return code from the last communications operation. The return code is not terminated with a null character. This buffer must be at least 8 bytes in length.

EHNRQ_EXEC

Purpose

This routine sends an SQL statement (other than a SELECT) to the host system to be prepared and run.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_EXEC(
                  HWND            hWnd,
                  char _far *     stmnt,
                  FARPROC         lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

stmnt — Points to a null-terminated character string that contains the SQL statement to be run. The maximum length of the string is limited to the buffer size specified on the EHNRQ_START or EHNRQ_STARTSEC call.

Note: On DBCS systems, this limit applies to the length of the statement after it has been translated to EBCDIC.

lpProcCallBack — Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_EXECPM

Purpose

This function is similar to the EHNRQ_EXEC function, however, this function supports the use of parameter markers. A parameter marker is a question mark (?) that appears in a statement string of dynamic SQL statements. The question mark can appear any place a host variable can appear for static SQL statements. Using no parameter markers is also supported. Using the EHNRQ_EXEC function, this routine prepares only the statement. To run the statement, the EHNRQ_EXECVAL function must be called. The EHNRQ_EXECVAL function specifies values for the parameter markers.

Using statements with parameter markers enhances performance allows an application programmer to prepare a statement once and then run it many times using different sets of values for the parameter markers.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_EXECPM (
                  HWND              hWnd,
                  char _far *       stmnt,
                  hcursor _far *    cur,
                  FARPROC           lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

stmnt — Points to a null-terminated character string that contains the SQL statement (with parameter markers) to be run. The maximum length of the string is limited to the buffer size specified on the EHNRQ_Start or EHNRQ_STARTSEC call.

Note: On DBCS systems, this limit applies to the length of the statement after it has been translated to EBCDIC.

cur — Points to a hcursor variable that is used to return a handle identifying this statement. This handle must be passed on subsequent EHNRQ_EXECVAL calls that reference the parameter markers for this statement.

lpProcCallBack — Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information

on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should be used only by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not suspended.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_EXECST

Purpose

This API executes one or more previously stored SQL statements in an SQL package on the host AS/400 system.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern int  extentry EHNRQ_EXECST(
                     HWND           hWnd,
                     struct execst_parms far *execst,
                     FARPROC lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

execst — Points to the execst_pars data structure. See "execst_parms" on page 25-34 for additional information.

lpProcCallBack — Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used, and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should be used only by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not suspended.

Additional Information

Stored SQL statements can be executed in any order specified by the parameters passed to this API. All the statements executed in a single call must exist in the same SQL package.

If the quit_on_warning option is set to TRUE, the host system treats warnings the same as errors. If a warning occurs, the host system will stop processing the EHNRQ_EXECST request. To process errors promptly, you should try to limit the number of stored statements executed by an EHNRQ_EXECST request in your application. If the quit_on_warning option is set to FALSE, the host system ignores warnings and continues processing until an error occurs or until it has finished processing the EHNRQ_EXECST request.

Also, EHNRQ_EXECST does not support remote databases. In this situation, a remote database is one that is accessed by a call to EHNRQ_CONNECT.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

After a return from EHNRQ_EXECST with a nonzero return code, verify that any cursor handles opened by the call contain nonzero values. If the cursor handle is zero, the cursor is closed.

EHNRQ_EXECVAL

Purpose

This function sends parameter values for a prepared EHNRQ_EXECPM statement and then runs the statement.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_EXECVAL(
                  HWND                hWnd,
                  hcursor             cur,
                  char _far *         fmt,
                  void _far * _far *  vap,
                  FARPROC             lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

cur — Identifies the handle corresponding to an prepared execute with parameter marker statement. This must be the value returned on the EHNRQ_EXECPM call for this statement.

fmt — Points to a null-terminated character string that contains format items. The format items define the data types used for each of the parameter markers specified in the EHNRQ_EXECPM statement. Each format item must be separated by a blank character. The number of format items in the string defines the number of ele-

ments required in the array of pointers passed in the vap parameter.

One format item must be provided for each parameter marker that is specified in the EHNRQ_EXECPM call. The allowable format items are:

A The corresponding variable is a far, null-terminated character string. The corresponding SQL column must be a CHAR, VAR CHAR, or LONG VAR CHAR type.

F The corresponding variable is a FLOAT type. The corresponding SQL column may be any of the SQL numeric data types (SMALLINT, INTEGER, NUMERIC, DECIMAL, single-precision or double-precision FLOAT).

D The corresponding variable is a DOUBLE type. The corresponding SQL column may be any of the SQL numeric data types.

L The corresponding variable is a LONG integer type. The corresponding SQL column may be any of the SQL numeric data types.

S The corresponding variable is a SHORT integer type. The corresponding SQL column may be any of the SQL numeric data types.

vap Points to an array of far pointers to variables of the type indicated by the format string items. The values of these variables is inserted into the SQL statement specified in the EHNRQ_EXECPM in place of the parameter markers. If a value of NULL is to be used, a NULL pointer must appear in the array of pointers.

lpProcCallBack Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40. If the return code is negative, a format item that is not valid was found. The absolute value of the return code indicates which format item was being processed when the error was discovered.

EHNRQ_FETCH

Purpose

This routine retrieves a row of data satisfying a previously prepared and executed SELECT statement. It differs from the EHNRQ_GETF operation described on page 15-28 in that it retrieves the raw data as returned by SQL on the remote system. (Raw data does not have any conversion of zoned or decimal data, and the block of data returned is not parsed into individual fields or variables.) The benefits of using EHNRQ_FETCH are as follows:

- The number and type of columns being returned need not be known in advance.
- The access to the raw data may enable its use without the loss of precision that could occur during data conversions performed automatically by EHNRQ_GETF (for example, when converting FLOAT or scaled DECIMAL data to INT).

A disadvantage of using EHNRQ_FETCH is that more detailed knowledge of SQL data formats and control blocks is required, as well as the need to provide explicit data conversions.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_FETCH(
                  HWND            hWnd
                  hcursor         cur,
                  char _far *     rcvr,
                  short _far *    rcvr_len,
                  FARPROC         lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

cur — Identifies a cursor handle corresponding to an active SELECT cursor (returned by the EHNRQ_SELECT or EHNRQ_SELECTPM call).

rcvr — Points to a buffer area that is used to store the retrieved column values and any associated indicators.

The size and layout of the result area are as follows:

1. Result columns are returned in the order in which they are described in the SQLDA structure returned by EHNRQ_DESC or in the attribute table returned by EHNRQ_ATTR.
2. Column values that are an odd number of bytes in length are followed by a single pad byte (each column starts on an even byte boundary).
3. If a column has an associated indicator, the indicator is stored immediately after the column value data (and its trailing pad byte, if present). If a column has no associated indicator, no space is reserved. See Figure 15-1 on page 15-27 for an

example of a SELECT statement and how the corresponding result area is laid out.

For example:

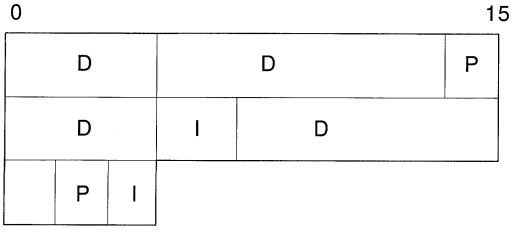
Suppose a SELECT statement similar to the following is run:

```
SELECT A,B,C,D FROM SAMPLE_TABLE
```

Where:

A Is an integer column with no associated indicator
B Is a fixed-character column of width 13 with no associated indicator
C Is an integer column with an associated indicator
D Is a fixed-character column of width 13 with an associated indicator

A result area of 36 bytes would be returned, laid out as follows:



D = Column data
I = Indicator
P = Pad byte

RV2F956-0

Note: EHNRQ_ATTR can be used to get a table that contains offsets to the start of the column data in the result area. In addition, the Col_Ind_Off record of the structure returned by EHNRQ_ATTR indicates whether or not an indicator is in the column.

rcvr_len Points to a short integer variable. On input, it specifies the size in bytes of the area pointed to by the rcvr parameter. If the area is not large enough, this variable is used to store the minimum buffer size needed for this call.

lpProcCallBack Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_FREEPM

Purpose

This routine closes or frees a prepared statement with parameter markers. This function is valid for both SELECT statements and statements other than SELECT. This is mainly for internal remote SQL tracking. Nothing is closed or freed by SQL for the existing prepared statement. This allows remote SQL to reuse handles.

If the handle is for a prepared SELECT statement and the cursor is currently open for that statement, the cursor is closed and the handle is freed.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern int ententry EHNRQ_FREEPM(
                    HWND          hWnd
                    hcursor       cur,
                    FARPROC       lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

cur — Identifies a handle corresponding to a previously prepared statement with parameter markers. This must be the value returned on the EHNRQ_SELECTPM or EHNRQ_EXECPM call for this statement.

lpProcCallBack — Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_GETF

Purpose

This routine provides an alternative to EHNRQ_FETCH, described on "EHNRQ_FETCH" on page 25-13. Rather than yielding a selected SQL table row in raw data form, the selected column values are converted to the appropriate data types, if necessary, and are stored directly into specified variables. If EHNRQ_GETF is used, the application programmer is not required to do EHNRQ_DESC or EHNRQ_ATTR calls, or to be aware of the precise format of the data as provided by SQL. Knowledge of the number of columns and their general type (numeric or character) is still required.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_GETF(
                  HWND                  hWnd,
                  hcursor               cur,
                  char _far *           fmt,
                  void _far * _far *    vap,
                  FARPROC               lpProcCallBack);
```

Parameters

hWnd Identifies the current window of the application.

cur Identifies a cursor handle corresponding to an active SELECT cursor (returned by the EHNRQ_SELECT or EHNRQ_SELECTPM call).

fmt Points to a null-terminated character string that contains format items. Format items define the number of columns to be retrieved and the type of each column. Each format item must be separated by a blank character. The number of format items also defines the number of elements required in the array of pointers passed in the vap parameter.

The allowable format items are:

Annn The corresponding variable is a far pointer to an area to receive a null-terminated character string with a maximum length of nnn (including the ending null). The corresponding SQL data type must be a CHAR, VAR CHAR, or LONG VAR CHAR type. If the column data is longer than the output string, it is truncated without warning.

F The corresponding variable is a FLOAT type. The corresponding SQL data type may be any of the SQL numeric data types (SMALLINT, INTEGER, NUMERIC, DECIMAL, single-precision or double-precision FLOAT). Loss of precision may result if the precision or scale of the column value is greater than can be accommodated.

Note: Floating point numbers are stored in the IEEE format.

D — The corresponding variable is a DOUBLE (8-byte float) type. The corresponding SQL data type may be any of the SQL numeric data types. Loss of precision may result if the precision or scale of the column value is greater than can be accommodated.

Note: Floating point numbers are stored in the IEEE format.

L — The corresponding variable is a LONG (4-byte) integer type. The corresponding SQL data type may be any of the SQL numeric data types. Loss of precision will result if the column value has a nonzero scale or a fractional part.

S — The corresponding variable is a SHORT (2-byte) integer type. The corresponding SQL data type may be any of the SQL numeric data types. Loss of precision results if the column value has a nonzero scale or a fractional part.

vap — Points to an array of far pointers to the variables into which the converted column values are to be stored. There must be a pointer entry in the array for each item in the format list.

lpProcCallBack

Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

If the return code is negative, a format item was found that was not valid, or the indicated data type was incompatible with the data type of the corresponding SQL column. The absolute value of the return code indicates which format item was being processed when the error was discovered.

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_INVOKE

Purpose

This routine causes the AS/400 server to start the specified AS/400 program. This specified program may then use the server's QRQRCVDT and QRQSNDT entry points to receive parameters from the PC application and to return results to it. See "Creating Applications Using Program-to-Program Communications" on page 15-4 for additional information about communicating with an AS/400 application beyond the base database server.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_INVOKE(
                  HWND            hWnd,
                  char _far *     pgm);
```

Parameters

hWnd Identifies the current window of the application.

pgm Points to a character string that specifies the name of the AS/400 program to run. This string must be in the form *library name/program*. This string should be either a null-terminated character string or padded with trailing blanks. If the string is null-terminated, it can be up to 21 characters in length (22 including the null character). If the string is padded with blanks, the string must be 21 characters long.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_OPTIONS

Purpose

Allows applications to query and set SQL parser options. The parser options currently supported are:

- Commitment control level
- Date format
- Date separator
- Decimal separator
- Naming convention
- Time format
- Time separator

Note: Options can be queried any time there is not an active block mode select or a pending send or receive. Options can be set only before any SQL statements are processed.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern int  extentry EHNRQ_OPTIONS(
                     HWND           hWnd,
                     int            option_ind,
                     struct         options_struct,
                     int            opt_len);
```

Parameters

hWnd Identifies the current window of the application.

option_ind
Indicates whether a SET or QUERY option is performed.

0 RQ_OPTION_QUERY
1 RQ_OPTION_SET

opt_struct Identifies the size in bytes of the area pointed to by the opt_struct parameter. This is the size of Options_Struct.

For a SET operation, the indicator field for each indicator value pair can have one of the following input values:

indicator = 0 The current value of this option is not changed.

indicator = 1 The current value of this option is changed.

indicator = n (where n is any other value) Is interpreted as if it were a zero (0) and, therefore, the current value is not changed.

For the possible values for each parser option, see "Remote SQL Parser Options" on page 15-52.

For a QUERY operation, the indicator field for each pair is ignored.

opt_len Identifies the size of the opt_struct structure. If the size is too small, this value is set to the minimum size required in bytes. If the size is more than the minimum size required, this value is not changed.

The returned values are:

opt_struct For a SET operation, the indicator field for each pair can have one of the following output values:

indicator = 0 The value specified for this option is valid.

indicator = E The value specified for this option is not valid.

For a QUERY operation, the indicator field for each pair has the following output value:

indicator = 0 The value specified for this option has been returned successfully.

For a list of the option values, see Table 15-3 on page 15-52.

opt_len Specifies the size of the returned structure.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_PREPST

Purpose

Prepares and stores SQL statements into an SQL package on the host AS/400 system for later execution.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern int  extentry EHNRQ_PREPST(
                     HWND           hWnd,
                     struct prepst_parms far *prepst,
                     FARPROC lpProcCallBack);
```

Parameters

hWnd Identifies the current window of the application.

prepst Points to the buffer that holds the description of the SQL package to be prepared and stored. For additional information, see "prepst_parms" on page 25-37.

lpProcCallBack
Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used, and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should be used only by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not suspended.

Additional Information

A new package is created, or an existing package accessed, on the host AS/400 system specified on the call to the EHNRQ_START API.

If the quit_on_warning option is set to TRUE, the host system treats warnings the same as errors. If a warning occurs, the host system will stop processing the EHNRQ_PREPST request. To process errors promptly, you should try to limit the number of stored statements executed by an EHNRQ_PREPST request in your application. If the quit_on_warning option is set to FALSE, the host system ignores warnings and continues processing until an error occurs or until it has finished processing the EHNRQ_PREPST request.

Also, EHNRQ_PREPST does not support remote databases. In this situation, a remote database is one that is accessed by a call to EHNRQ_CONNECT.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_RECV

Purpose

This routine receives a block of data from the host partner program. A call to this routine is valid only after a call to EHNRQ_INVOKE has been made. This routine must be called repeatedly until an EHNRQ_EOD (0x001D) return code is received indicating that the host system is finished sending. At that time, EHNRQ_SEND or any of the SQL database access calls may be issued.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_RECV(
                  HWND            hWnd,
                  int             buflen,
                  char _far *     buffer,
                  int _far *      rcvlen,
                  FARPROC         lpProcCallBack);
```

Parameters

hWnd Identifies the current window of the application.

buflen Identifies the length of the buffer that is used to return the received data. This length is limited by the buffer size specified on the EHNRQ_START or EHNRQ_STARTSEC command.

buffer Points to a character buffer that is used to return the received data.

rcvlen Points to an integer variable that is used to return the actual length of the data received from the AS/400 program.

lpProcCallBack
Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40. If the return code is EHNRQ_NVLD_RW (X'001A'), the partner program is receiving. Therefore, a receive request is not valid now.

EHNRQ_RTVMSG

Purpose

Retrieves the message text associated with the status of the last SQL operation performed. The message text is placed in the user's buffer and is null-terminated.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern int  extentry EHNRQRTVMSG(
                     HWND            hWnd,
                     char far *      buffer,
                     short far *     buf_len);
```

Parameters

User-supplied values:

hWnd Identifies the current window of the application.

buffer Points to the buffer where the message text is placed.

buf_len The size in bytes of the buffer that holds the message text.

Returned values:

buf_len If the size of the buffer is too small, this value is set to the minimum size needed in bytes to hold the complete message text.

If the size of the buffer is larger than the minimum size, this value is the size in bytes of the message.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_SELECT

Purpose

Sends an SQL SELECT statement to the host system to be prepared and executed. Up to 10 select statements may be prepared or active at one time.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_SELECT(
                  HWND              hWnd,
                  char _far *       stmnt,
                  int               update,
                  hcursor _far *    cur,
                  FARPROC           lpProcCallBack);
```

Parameters

hWnd Identifies the current window of the application.

stmnt Points to a null-terminated character string that identifies the SQL SELECT statement to be run. The maximum length of this string is limited to the buffer size specified on the EHNRQ_START or EHNRQ_STARTSEC call.

Note: On DBCS systems, this limit applies to the length of the statement after it has been translated to EBCDIC.

update Specifies whether updates and deletes are allowed. If a nonzero value is specified, selected rows are returned one at a time from the host system, and EHNRQ_UPCUR or EHNRQ_DELETE calls can optionally be used to update or delete values in selected rows. If a zero value is provided, the physical row data is returned from the host in blocks while selected rows are logically returned to the application one at a time. This provides improved performance, but precludes updates or deletions based on cursor position.

You can use the EHNRQ_SETROWS function to specify the number of rows you want this function to block together. If you do not use EHNRQ_SETROWS, all rows are blocked together.

Note: A cursor is a logical pointer associated with the processing of a SELECT statement that is used to keep track of which selected rows have been returned. EHNRQ_FETCH and EHNRQ_GETF cause the cursor to be moved to the next available selected row, if any. EHNRQ_UPCUR and EHNRQ_DELETE act upon the most recently retrieved row as indicated by the current cursor position. See the *SQL/400* Reference* for more information.

cur Points to an hcursor variable that is used to return a handle identifying this SELECT statement. This handle must be passed on subsequent calls that reference this statement as the active SELECT statement (for example, for EHNRQ_GETF and EHNRQ_UPCUR calls).

lpProcCallBack
Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null

pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_SELECTPM

Purpose

This function is similar to the EHNRQ_SELECT function. However, this function supports the use of parameter markers. A parameter marker is a question mark (?) that appears in a statement string of dynamic SQL statements. The question mark can appear any place a host variable can appear for static SQL statements. Unlike the EHNRQ_SELECT function, this routine only prepares the statement. To run the statement, the EHNRQ_SELECTFAL function must be called. The EHNRQ_SELECTVAL function specifies values for the parameter markers.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_SELECTPM(
                  HWND              hWnd,
                  char _far *       stmt,
                  hcursor _far *    cur,
                  FARPROC           lpProcCallBack);
```

Parameters

hWnd Identifies the current window of the application.

stmt Points to a null-terminated character string that contains the SQL statement, with markers, to be run. The maximum length of the string is limited to the buffer size specified on the EHNRQ_START or EHNRQ_STARTSEC call.

Note: On DBCS systems, this limit applies to the length of the statement after it has been translated to EBCDIC.

cur Points to an hcursor variable that is used to return a handle. This handle must be passed on subsequent EHNRQ_SELECTVAL calls that reference the parameter markers for this statement.

lpProcCallBack

Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_SELECTVAL

Purpose

This function sends parameter values for a prepared EHNRQ_SELECTPM statement and then runs the statement.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQAPI.H"
extern int ententry EHNRQ_SELECTVAL(
                    HWND                    hWnd,
                    hcursor _far *          cur,
                    char _far *             fmt,
                    void _far * _far *      vap,
                    int                     update,
                    FARPROC                 lpProcCallBack);
```

Parameters

hWnd Identifies the current window of the application.

cur Identifies the handle corresponding to a prepared select with parameter marker statement. This must be the value returned on the EHNRQ_SELECTPM call for this statement.

fmt Points to a null-terminated character string of format items. The format items define the data types to be used for each of the parameter markers specified in the EHNRQ_SELECTPM statement. Each format item must be separated by a blank character. The number of format items on the string defines the number of elements required in the array of pointers passed on the vap parameter.

One format item must be provided for each parameter marker that is specified in the EHNRQ_SELECTPM call. The allowable format items are:

A The corresponding variable is a far, null-terminated character string. The corresponding SQL column must be a CHAR, VAR CHAR, or LONG VAR CHAR type.

F The corresponding variable is a FLOAT type. The corresponding SQL column may be any of the SQL numeric data types (SMALLINT, INTEGER, NUMERIC, DECIMAL, single-precision or double-precision FLOAT).

D The corresponding variable is a DOUBLE type. The corresponding SQL column may be any of the SQL numeric data types.

L The corresponding variable is a LONG integer type. The corresponding SQL column may be any of the SQL numeric data types.

S The corresponding variable is a SHORT integer type. The corresponding SQL column may be any of the SQL numeric data types.

Note: Floating point numbers are assumed to be in the IEEE format.

vap Points to an array of far pointers to variables of the type indicated by the format string items. The values of these variables are inserted into the SQL statement specified in the EHNRQ_SELECTPM in place of the parameter markers. If a value of NULL is to be used, a NULL pointer must appear in the array of pointers.

update Specifies whether updating and deleting are allowed. If a nonzero value is specified, selected rows are returned one at a time from the host system, and EHNRQ_UPCUR or EHNRQ_DELETE calls can optionally be used to update or delete values in selected rows. If a zero value is provided, the physical row data is returned from the host in blocks while selected rows are logically returned to the application one at a time. This provides improved performance, but precludes updates or deletions based on cursor position.

You can use the EHNRQ_SETROWS function to specify the number of rows you want this function to block together. If you do not use EHNRQ_SETROWS, all rows are blocked together.

Note: A cursor is a logical pointer associated with the processing of a SELECT statement that is used to keep track of which selected rows have been returned. EHNRQ_FETCH and EHNRQ_GETF cause the cursor to be moved to the next available selected row, if any. EHNRQ_UPCUR and EHNRQ_DELETE act upon the most recently retrieved row as indicated by the current cursor position. See the *SQL/400* Reference* for more information.

lpProcCallBack
Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

If the return code is negative, a format item that is not valid was found. The absolute value of the return code indicates which format item was being processed when the error was discovered.

EHNRQ_SEND

Purpose

This routine sends a block of data to the partner program on the remote system. This call is only valid after an EHNRQ_INVOKE call has been issued (for example, when an AS/400 application is talking to the remote SQL server on the host side).

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_SEND(
                  HWND          hWnd,
                  int           send_len,
                  char _far *   buffer);
```

Parameters

hWnd	Identifies the current window of the application.
send_len	Identifies the number of bytes to send. The maximum value allowed is limited to the buffer size specified on the EHNRQ_START or EHNRQ_STARTSEC call.
buffer	Points to a character buffer containing the data to be sent.

Return Codes

If the return code is EHNRQ_NVLD_RW (X'001A'), the partner program is sending. Therefore, a send request is not valid now.

EHNRQ_SETROWS

Purpose

This routine allows an application programmer to specify the maximum number of rows to be blocked together when an EHNRQ_SELECT or EHNRQ_SELECTVAL is done in a mode other than update. If this function is not called and the specified mode is not update, EHNRQ_SELECT and EHNRQ_SELECTVAL block all rows together.

This function allows PC applications to issue other API calls between requests to retrieve the data from the host system.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_SETROWS(
                  HWND              hWnd,
                  unsigned short    max_rows);
```

Parameters

hWnd	Identifies the current window of the application.
max_rows	Identifies the maximum number of rows to be blocked together when a SELECT is done in a mode other than update mode.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_SQLCA

Purpose

This routine returns information describing the ending conditions associated with the last SQL operation performed.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_SQLCA(
                  HWND                  hWnd,
                  struct sqlca _far *   lastca,
                  short _far *          calen);
```

Parameters

hWnd	Identifies the current window of the application.
lastca	Points to a buffer that is used to store the information describing the ending conditions of the last SQL operation. This information is stored in this buffer in the format defined by the sqlca structure. For a description of the sqlca structure, see "sqlda" on page 25-39.
calen	Points to a short integer variable. On input, it specifies the size in bytes of the buffer pointed to by the lastca parameter. This value should be at least 136 bytes. If the buffer is not large enough, this variable is used to store the minimum buffer size needed for this call.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_START

Purpose

This routine performs the initialization necessary for communication with the remote SQL server on the host system. This function or the EHNRQ_STARTSEC function must be the first remote SQL function called by an application.

This routine uses the user ID and password specified when the PC Support router was started. To specify a different user ID or password, the EHNRQ_STARTSEC function should be used.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_START(
                  HWND              hWnd,
                  char _far *       sys,
                  short             commit,
                  unsigned short    buffer);
                  FARPROC           lpProcCallBack);
```

Parameters

hWnd — Identifies the current window of the application.

sys — Points to a null-terminated character string that identifies the host system name.

commit — Indicates the level of COMMIT-type operations that are to be supported. For the possible values using native AS/400 naming conventions or SQL naming conventions, see "Remote SQL Start Options" on page 15-51.

buffer — Identifies the size of the buffer to be used for data transfers to and from the host system. This value represents the maximum length (after translation) of a SQL statement provided on a EHNRQ_EXEC, EHNRQ_EXECPM, EHNRQ_SELECT, or EHNRQ_SELECTPM call, or constructed as a result of an EHNRQ_UPCUR, EHNRQ_EXECVAL, or EHNRQ_SELECTVAL call.

This value also represents the maximum amount of data that can be sent (after translation) by a single EHNRQ_SEND call or received by an EHNRQ_RECV call. The value provided is adjusted if necessary to force it to fall into the range from 1024 to 31 744 bytes. (For example, a value of zero would result in a maximum buffer size of 1024 bytes). This parameter does not limit the amount of data that can be received by the application on a single EHNRQ_FETCH or EHNRQ_GETF call.

lpProcCallBack Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_STARTSEC

Purpose

This routine performs the initialization necessary for communication with the remote SQL server on the host system using the specified user ID and password. This function or EHNRQ_START must be the first remote SQL function called by an application. The override of the user ID and password is for this conversation only. The user ID and password revert back to the values specified when the PC Support router was started after this conversation ends.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern int ententry EHNRQ_STARTSEC(
                    HWND            hWnd,
                    char _far *     sys,
                    short           commit,
                    unsigned short  buffer,
                    char _far *     userid,
                    char _far *     password,
                    char _far *     reserved,
                    FARPROC         lpProcCallBack);
```

Parameters

hWnd Identifies the current window of the application.

sys Points to a null-terminated character string that identifies the host system name.

commit — Indicates the level of COMMIT-type operations that are to be supported. For the possible values using native AS/400 naming conventions or SQL naming conventions, see "Remote SQL Start Options" on page 15-51.

buffer — Identifies the size of the buffer to be used for data transfers to and from the host system. This value represents the maximum length (after translation) of a SQL statement provided on a EHNRQ_EXEC, EHNRQ_EXECPM, EHNRQ_SELECT, or EHNRQ_SELECTPM call, or constructed as a result of an EHNRQ_UPCUR, EHNRQ_EXECVAL, or EHNRQ_SELECTVAL call.

This value also represents the maximum amount of data that can be sent (after translation) by a single EHNRQ_SEND call or received by an EHNRQ_RECV call. The value provided is adjusted if necessary to force it to fall into the range from 1024 to 31 744 bytes. (For example, a value of zero would result in a maximum buffer size of 1024 bytes). This parameter does not limit the amount of data that can be received by the application on a single EHNRQ_FETCH or EHNRQ_GETF call.

userid — Identifies the user ID to be associated with this session. This user ID determines the authority levels for working with databases. The maximum length of the user ID is 10 characters.

password — Identifies the password for the user ID specified by the userid parameter. The maximum length of the password is 1 characters.

reserved — Is a reserved parameter and must be null.

lpProcCallBack — Identifies the address of a routine within the calling application that the remote SQL DLL can call. The purpose of this parameter is to allow the application to give control back to Windows during a lengthy remote SQL request so other applications are allowed to run. This address must be the value returned from the MakeProcInstance call for the routine. The routine also must be exported in the .DEF file of the application. For more information on how this parameter is used and what function the routine should perform, see "Multitasking Issues" on page 19-5.

Note: The parameter should only be used by experienced Windows programmers. For inexperienced Windows programmers, a null pointer should be passed for this parameter. If the transfer request is long, a message box should be displayed before calling this routine to indicate the personal computer is not hung.

Return Codes

For return codes, see "Return Codes for the Remote SQL Windows DLL" on page 25-40.

EHNRQ_UPCUR

Purpose

This function performs an update of one or more columns in the row at the current cursor position. EHNRQ_UPCUR must be preceded by a successful EHNRQ_FETCH or EHNRQ_GETF call. In addition, the EHNRQ_SELECT or EHNRQ_SELECTVAL call that returned the specified cursor handle must have specified update mode. Only columns that are being retrieved by the active SELECT statement can be updated by this call.

Note: Some SELECT statement options preclude updates based on cursor position. For example, ORDER BY, FOR UPDATE OF, and GROUP BY can all affect whether updates may be done (see "Read Only Tables" in the *SQL/400* Reference*).

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNRQW.H"
extern pascal far EHNRQ_UPCUR(
                  HWND                  hWnd,
                  hcursor               cur,
                  char _far *           fmt,
                  void _far * _far *    vap);
```

Parameters

hWnd Identifies the current window of the application.

cur Identifies a cursor handle corresponding to an active SELECT cursor (returned by the EHNRQ_SELECT or EHNRQ_SELECTPM call).

fmt Points to a null-terminated character string of format items. Format items define the columns in the row to be updated and the data types of the columns. Each format item must be separated by a blank character. The number of format items defines the number of elements required in the array of pointers passed in the vap.

One format item must be provided for each column being returned by the active SELECT statement. However, if all of the columns are not being updated, format items only need to be provided for the columns up to and including the last one to be updated. For example, if six columns are being returned, but only the first and third are being updated, the format list might look like "A * D." The allowable format items are:

A The corresponding variable is a far, ASCIIZ string. The corresponding SQL column must be a CHAR, VAR CHAR, or LONG VAR CHAR type. If the update string is longer than the SQL column, the update operation is suppressed.

F The corresponding variable is a FLOAT type. The corresponding SQL column may be any of the SQL numeric data types (SMALLINT, INTEGER, NUMERIC, DECIMAL, single-precision or double-precision FLOAT).

D The corresponding variable is a DOUBLE type. The corresponding SQL data type may be any of the SQL numeric data types.

L The corresponding variable is a LONG integer type. The corresponding SQL data type may be any of the SQL numeric data types.

S The corresponding variable is a SHORT integer type. The corresponding SQL data type may be any of the SQL numeric data types.

* No corresponding variable is required, and one must not be provided. This format item serves as a placeholder to skip over a column that is not being updated.

Notes:

1. Floating point numbers are assumed to be in the IEEE format.
2. Loss of precision may result if the precision or scale of the update value is greater than that of the SQL column.

vap Points to an array of far pointers to variables of the type indicated by the corresponding format string items. The values of these variables are used to update the corresponding selected columns of the current table row. There must be an entry in the array for each item in the format list, excluding asterisks (*). If a value of NULL is to be used, a NULL pointer must appear in the array of pointers.

Additional Information

Performance can be improved by providing values for all the columns specified in the SELECT statement. This applies even when all the columns are not to be updated.

Return Codes

If the return code is negative, then a format item that is not valid was found. The absolute value of the return code indicates which format item was being processed when the error was discovered.

If the return code is zero or positive, see "Return Codes for the Remote SQL Windows DLL" on page 25-40 for the return codes.

Remote SQL Windows API Structures

Col_Attributes

Description

This structure is used with the EHNRQ_ATTR API to return descriptions of the columns identified by a SELECT statement. This structure defined the format of an entry in the column attributes table.

```
struct Col_Attributes
{
   short Col_Type;
   unsigned char Col_Prec;
   unsigned char char Col_Scale;
   char  CCSID[2];
   char rsvd2[4];
   short Col_Width;
   short Col_Data_Off;
   short Col_Ind_Off;
   char Col_Name[32];
};
```

Field	Description
Col_Type	Identifies the SQL column type.
Col_Prec	Identifies the column precision. This field is only used for columns with an arithmetic type.
Col_Scale	Identifies the column scale.
CCSID	Is a reserved field.
rsvd2	Is a reserved field.
Col_Width	Identifies the source data width of the column.
Col_Data_Off	Identifies the offset to data in the result area.
Col_Ind_Off	Identifies the offset of the indicator, for the column, in the result area. If this value is zero, no indicator exists for this column.
Col_Name	Identifies the null-terminated column name.

---

execst_parms

Description

```
struct execst_parms
{
   char               *lib_name;
   char               *pkg_name;
   short              number_of_statements;
   unsigned short     quit_on_warning;
   struct execst_data prepst_data;
   {
      short           stmt_number;
      hcursor         *c_hndl;
      unsigned short  update;
      char            *format;
      void            *v_array;
   } execst_data[1];
};
```

Field	Description
lib_name	Points to a null-terminated string containing the library name. This name must be between one and ten characters long and have no blanks or wildcard characters. The library name may be QTEMP if the package exists in that library.
pkg_name	Points to a null-terminated string containing the name of the SQL package. This name must be between one and ten characters long and have no blanks or wildcard characters.
number_of_statements	The number of statements to be processed (must be greater than zero). This field is also used to return the number of statements that are successfully executed.
quit_on_warning	Indicates how SQL warning messages are handled. TRUE Quit processing when an SQL warning is encountered. FALSE Ignore SQL warnings.
stmt_number	Statement number within the SQL package.
c_handle	Points to the cursor handle for a SELECT statement. This must be NULL for any non-SELECT statement.
update	Set to 0 for no update intent. This field is checked only if c_hndl is not NULL.
format	Points to a format string describing data for parameter markers in the statements that are to be executed.
v_array	Array of pointers to data for parameter markers in statements to be executed.
execst_data	Points to the execst_data structure.

Options_Struct

Description

This structure is used with the EHNRQ_OPTIONS API for setting and querying SQL parser options.

On input for a set request, an indicator value of 1 means the option should be updated. Any other value means the indicator should not be updated. On output from a set request, an indicator value of 0 means the indicator value was valid. A value of -1 means the indicator value was not valid.

On input for a query request, the indicator fields of this structure are ignored. However, it is recommended that they are set to zero. On output from a query request, all indicator fields are set to 0.

```
struct Options_Struct
  {
   char name_conven_indicator;
   char name_conven_value;
   char comm_cntrl_indicator;
   char comm_cntrl_value;
   char date_format_indicator;
   char date_format_value;
   char date_separt_indicator;
   char date_separt_value;
   char time_format_indicator;
   char time_format_value;
   char time_separt_indicator;
   char time_separt_value;
   char decimal_point_indicator;
   char decimal_point_value;
  };
```

Table 25-1. Naming Convention Indicator

Constant Name	Decimal Value	SQL Parser Value
EHNRQ_NAME_SYS	X'00'	*SYS (AS/400 names)
EHNRQ_NAME_SQL	X'01'	*SQL (SQL naming convention)

Table 25-2. Commitment Control Indicator

Constant Name	Decimal Value	SQL Parser Value
EHNRQ_COMMIT_NONE	X'00'	*NONE (none)
EHNRQ_COMMIT_ALL	X'01'	*ALL (all)
EHNRQ_COMMIT_CHG	X'02'	*CHG (change)
EHNRQ_COMMIT_CS	X'03'	*CS (cursor stability)

Table 25-3. Date Format Indicator

Constant Name	Decimal Value	SQL Parser Value
EHNRQ_DATEFMT_USA	X'00'	*USA (US format)
EHNRQ_DATEFMT_ISO	X'01'	*ISO
EHNRQ_DATEFMT_EUR	X'02'	*EUR
EHNRQ_DATEFMT_JIS	X'03'	*JIS
EHNRQ_DATEFMT_MDY	X'04'	*MDY (month/day/year)
EHNRQ_DATEFMT_DMY	X'05'	*DMY (day/month/year)
EHNRQ_DATEFMT_YMD	X'06'	*YMD (year/month/day)
EHNRQ_DATEFMT_JUL	X'07'	*JUL (Julian)

Table 25-4. Date Separator Indicator

Constant Name	Decimal Value	SQL Parser Value
EHNRQ_DATESEP_PERIOD	X'00'	period (.)
EHNRQ_DATESEP_COMMA	X'01'	comma (,)
EHNRQ_DATESEP_BLANK	X'02'	blank ( )
EHNRQ_DATESEP_SLASH	X'03'	slash (/)
EHNRQ_DATESEP_DASH	X'04'	dash (–)

Table 25-5. Time Format Indicator

Constant Name	Decimal Value	SQL Parser Value
EHNRQ_TIMEFMT_USA	X'00'	*USA
EHNRQ_TIMEFMT_ISO	X'01'	*ISO
EHNRQ_TIMEFMT_EUR	X'02'	*EUR
EHNRQ_TIMEFMT_JIS	X'03'	*JIS
EHNRQ_TIMEFMT_HMS	X'04'	*HMS

Table 25-6. Time Separator Indicator

Constant Name	Decimal Value	SQL Parser Value
EHNRQ_TIMESEP_PERIOD	X'00'	period (.)
EHNRQ_TIMESEP_COMMA	X'01'	comma (,)
EHNRQ_TIMESEP_BLANK	X'02'	blank ( )
EHNRQ_TIMESEP_COLON	X'03'	colon (:)

Table 25-7. Decimal Point Indicator

Constant Name	Decimal Value	SQL Parser Value
EHNRQ_DECIMAL_PERIOD	X'00'	period (.)
EHNRQ_DECIMAL_COMMA	X'01'	comma (,)

prepst_parms

Description

```
struct prepst_parms
{
   char               *lib_name;
   char               *pkg_name;
   short              number_of_statements;
   char               create_package;
   unsigned short     quit_on_warning;
   struct prepst_data prepst_data
   {
      short           stmt_number;
      char            *sql_stmt;
   } prepst_data[1];
};
```

Field	Description
lib_name	Points to a null-terminated string containing the library name. This name must be between one and ten characters long and have no blanks or wildcard characters.
pkg_name	Points to a null-terminated string containing the name of the SQL package. This name must be between one and ten characters long and have no blanks or wildcard characters.
number_of_stmts	The number of statements to be processed (must be greater than zero). This field is also used to return the number of statements that are successfully executed.
create package	Indicates whether or not a package will be created.
	Y Create a new package.
	N Store the statements in an existing package.
	A If the package already exists, store the statements in the existing package.
quite_on_warning	Indicates how SQL warning message will be handled.
	TRUE Quit processing when an SQL warning is encountered.
	FALSE Ignore SQL warnings.
stmt_number	The statement number within the SQL package.
sql_stmt	Points to a null-terminated string containing the SQL statement to prepare and store.
prepst_data	Points to a prepst_data structure.

sqlca

Description

This structure is used by the EHNRQ_SQLCA API to return information about the ending conditions of the last SQL operation performed.

```
struct sqlca
{
   unsigned char  sqlcaid[8];
   long           sqlcabc;
   long           sqlcode;
   short          sqlerrml;
   unsigned char  sqlerrmc[70];
   unsigned char  sqlerrp[8];
   long           sqlerrd[6];
   unsigned char  sqlwarn[11];
   unsigned char  sqlext[5];
};
```

Field	Description
sqlcaid	Contains the characters *SQLCA*.
sqlcabc	Specifies the size of this structure in bytes. This size is 136.
sqlcode	Is used to store the SQL return code.
sqlerrml	Identifies the length of the error message tokens in SQLERRMC.
sqlerrmc	Is used to return error message tokens.
sqlerrp	Is used to return diagnostic information.
sqlerrd	Is used to return diagnostic information.
sqlwarn	Contains warning flags.
sqlext	Is a reserved field.

---

sqlda

Description

This structure is used by the EHNRQ_DESC API to return descriptions of the columns identified by a SELECT statement. The information is returned in a list. The first four fields of the sqlda structure make up the list header. This header is followed by one or more sqlvar structures. One sqlvar structure is returned for each column to be described.

```
struct sqlda
{
   unsigned char  sqldaid[8];
   long           sqldabc;
   short          sqln;
   short          sqld;
   struct sqlvar
   {
      short          sqltype;
      short          sqllen;
      unsigned char *sqldata;
      short         *sqlind;
      struct sqlname
      {
         short          length;
         unsigned char  data[30];
      }  sqlname;
   } sqlvar[1];
};
```

Field	Description
sqldaid	Contains the characters *SQLDA*.
sqldabc	Specifies the size in bytes of this entire structure (includes the length of all sqlvar elements). This value is 16 + (44 * sqln).
sqln	Specifies the number of sqlvar elements.
sqld	Specifies the number of used sqlvar elements.
sqltype	Specifies the column data type.
sqllen	Specifies the length of the column data.
sqldata	Points to the column data value.
sqlind	Points to a null indicator.
length	Length of the column name. This value must be between 1 and 30.
data	Specifies the column name.

Return Codes for the Remote SQL Windows DLL

Table 25-8 (Page 1 of 4). Return Codes for the Remote SQL Windows DLL

Return Code	Hex Value	Description
EHNRQ_OK	X'0000'	Operating completed successfully.
EHNRQ_MSG_NOTF	X'0003'	No message text available to retrieve.
EHNRQ_PREP_ERROR	X'0004'	An error occurred while preparing the statement. Use EHNRQSQLCA to obtain more information.
EHNRQ_NON_SELECT	X'0006'	An attempt was made to run a SELECT statement with EHNRQEXEC, or the statement specified on the EHNRQSELECT or EHNRQSELECTPM entry point is not a valid select statement.

Table 25-8 (Page 2 of 4). Return Codes for the Remote SQL Windows DLL

Return Code	Hex Value	Description
EHNRQ_NON_CONNECT	X'0006'	Invalid connect statement.
EHNRQ_LONG_STMNT	X'0007'	The SQL statement provided is longer than the buffer size specified or defaulted on the EHNRQACCEPT, EHNRQSTART, or EHNRQSTARTSEC call. The statement was not sent to the host system for processing.
EHNRQ_EXEC_ERROR	X'0008'	An error occurred while executing the statement. Use EHNRQSQLCA to obtain more information.
EHNRQ_NVLD_OPTION	X'0009'	At least one of the options that you tried to change had a value that was not valid.
EHNRQ_NO_FETCH	X'000A'	A successful EHNRQFETCH or EHNRQGETF was not the last operation run for the specified cursor.
EHNRQ_NOT_UPDATE	X'000B'	The cursor specified was not in update mode. Update mode is required for EHNRQDELETE or EHNRQUFCUR to be used.
EHNRQ_FETCH_ERR	X'000C'	An error occurred on the FETCH. No result data was returned. Use EHNRQSQLCA to obtain more information about the error.
EHNRQ_NOT_ACT	X'000D'	Cursor handle does not correspond to an active cursor.
EHNRQ_OPEN_ERROR	X'000E'	An error occurred opening the cursor.
EHNRQ_LASTROW	X'000F'	No more rows are available that satisfy the selection criteria. No result data was returned.
EHNRQ_MAX_HANDLE	X'0010'	The maximum number of handles (10) is already active. Each active SELECT statement and each active statement that uses a parameter marker has an associated handle.
EHNRQ_NVLD_STATE	X'0011'	Operation not valid now. An unrelated SELECT statement is active and is not in update mode, or the function is not valid because of a previous SQL statement that has been performed.
EHNRQ_XlateBufSz	X'0013'	Translation buffer size too small.
EHNRQ_ALRDY_INIT	X'0014'	Initialization has already been done through EHNRQSTART, EHNRQSTARTSEC, or EHNRQACCEPT.
EHNRQ_NOT_INIT	X'0014'	Remote SQL is not active.
EHNRQ_NVLD_COMMIT	X'0015'	The commit parameter is not 0, 1, 2, 3, 256, 257, 258, or 259.
EHNRQ_RMT_ACTIVE	X'0016'	Communications with a host application is already active through either EHNRQACCEPT or EHNRQINVOKE.
EHNRQ_NO_RMT_PGM	X'0017'	Send is not valid because there is no active partner application program on the host system.
EHNRQ_RMT_TERM	X'0018'	The partner program that this program initiated through an EHNRQINVOKE has ended. Database operations are still valid, but SEND and RECV are not.
EHNRQ_FORCE_TERM	X'0019'	The partner program that initiated this program has ended communications. An EHNRQEND should be issued.
EHNRQ_NVLD_RW	X'001A'	SEND/RECV exchange pending.

Table 25-8 (Page 3 of 4). Return Codes for the Remote SQL Windows DLL

Return Code	Hex Value	Description
EHNRQ_NVLD_LEN	X'001B'	The buffer provided was too small. No data was returned. Allocate a buffer at least the size of the data returned.
EHNRQ_DATA_TRUNC	X'001C'	Received data truncated. The buffer provided was not large enough to contain all of the data sent.
EHNRQ_EOD	X'001D'	End of data. The partner program has stopped sending. Database operations and EHNRQSEND are now valid.
EHNRQ_INVOC_FAIL	X'001E'	The host program requested on the previous EHNRQINVOKE could not be called. The most likely cause is that the library or program name provided is not correct.
EHNRQ_COMM_ERROR	X'0020'	Either there was no pending connection request from a host system, or an APPC error was detected while establishing the connection. EHNRQERROR may be used to obtain the APPC error return codes. An EHNRQEND should be done.
EHNRQ_LEVEL	X'0024'	The host server program and the local RMTSQL API version are at release levels that are not compatible.
EHNRQ_XI_UNAVAIL	X'0026'	PCSXI program has not been started (DOS only).
EHNRQ_LOAD_FAIL	X'0027'	Unable to load EHNRQAPI.OVL into protected memory (DOS only).
EHNRQ_CIM_UNAVAIL	X'0028'	The PC Support/400 router is not available. Processing cannot continue.
EHNRQ_DBCSXlateErr	X'002A'	DBCS translation error. 8580 Sys1IBM.table not found (DBCS only).
EHNRQ_XLATEOVERFL	X'002C'	Translation output buffer overflow (DBCS only). This is most likely due to an input string expanding during translation from ASCII to EBCDIC to a length greater than the buffer size specified on the EHNRQACCEPT, EHNRQSTART, or EHNRQSTARTSEC call.
EHNRQ_XLT_SETUP	X'002E'	Error initializing for DBCS translation. Unable to obtain ASCII DBCS range vector. PCSXLT may need to be loaded.
EHNRQ_NO_MEMORY	X'0032'	Insufficient memory to continue.
EHNRQ_MEMORY	X'0032'	Remote SQL has been ended. An error occurred when attempting to free memory (DOS only).
EHNRQ_2MANY_ACTIVE	X'0033'	Too many requester programs are using the remote SQL API. A limit of 10 active requester programs may use this API at one time.
EHNRQ_NO_USERID	X'0040'	No user ID specified, but a password was specified.
EHNRQ_USERID_INVLGTH	X'0041'	User ID exceeds maximum length of 10 characters.
EHNRQ_PWORD_INVLENGTH	X'0042'	Password exceeds maximum length of 10 characters.
EHNRQ_INV_SECURITY	X'0043'	User ID and password combination rejected by the host system.
EHNRQ_INV_CHAR	X'0044'	User ID or password contains invalid characters.
EHNRQ_PACKAGE_ERROR	X'0050'	Package error. The package does not exist, or is locked or is damaged.
EHNRQ_NVLD_PACKAGE	X'0051'	This is not a valid parameter for the Create Package API.

Table 25-8 (Page 4 of 4). Return Codes for the Remote SQL Windows DLL

Return Code	Hex Value	Description
EHNRQ_NO_STATEMENT	X'0052'	No statement is specified. The number of the statement is not greater than zero. .
EHNRQ_NVLD_NAME	X'0053'	The library or package name was not specified correctly.
EHNRQ_RMT_UPDT_NVLD	X'0054'	Cannot open a cursor for the stored SELECT statement while accessing a remote database.
EHNRQ_NVLD_STMT	X'0055'	Not a valid SQL statement.
EHNRQ_NVLD_STMTNUM	X'0056'	Not a valid SQL statement number
EHNRQ_CRT_RMTPKG_ERR	X'0057'	The create package field is set to 'Y' or 'A' while connected to a remote database.

Chapter 26. Data Queues Windows Application Program Interface and DDE Server

The data queues API for Windows provides two different interfaces for application programmers. Like the other PC Support functions, the data queues API provides a Windows DLL with entry points to access the data queues function. In addition to providing a DLL, data queues also supports the Windows DDE (dynamic data exchange) interface. This chapter explains both the DLL and the DDE interfaces.

For information on the requirements for developing and running applications that use the PC Support Windows APIs, see Chapter 19, "PC Support/400 Windows Application Program Interface Overview."

To use the data queues API, the PC Support data queues program (LOADDQ) must be loaded before starting Windows.

Data Queues Implementation of DDE

The Windows DDE protocol enables application programmers to communicate between applications. Data queues uses this function to provide communication between multiple personal computers and AS/400 applications by passing information through a shared data queue which resides on the AS/400 system.

For a more detailed description of the Windows DDE protocol, refer to the *Microsoft Windows Software Development Kit Reference*.

The following description of the data queues DDE interface refers to predefined constants found in the file DQSERVER.H in the PC Support tools folder (QIWSTOOL). The tools folder also contains a sample program which demonstrates how to use this interface. For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

Data queues provides a DDE server to process the DDE requests. This server must be started before issuing the first DDE message for data queues. Start the server by making a call to the Windows WinExec procedure from within an application program as follows:

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
WORD WinExec(
       LPSTR lpCmdLine,
       int   nCmdShow);
```

Parameters

lpCmdLine — Points to a null-terminated character string that identifies the name of the DDE server. The name of the server is DQSERVER. This name should contain a fully qualified path if the server cannot be found using the search criteria specified in the Windows definition of this procedure.

nCmdShow — Specifies how the DDE server window should initially be shown when the server is started. This parameter should have a value of SW_SHOWMINIMIZED to have the server window displayed as an icon.

The data queues DDE server can also be started by using an icon in the PC Support group or by using the run option of file pull-down on the Windows program manager.

DDE Messages

The Windows DDE interface is made up of several architected messages. The following message descriptions explain the data queues implementation of this protocol.

WM_DDE_ACK

Purpose

This message is used to acknowledge that a DDE message was received by either the data queues DDE server or by the client application program that issues the DDE requests. An acknowledgement message should be sent when the following DDE messages are received:

WM_DDE_ADVISE
WM_DDE_EXECUTE
WM_DDE_INITIATE
WM_DDE_POKE
WM_DDE_UNADVISE

Note: In addition, this message should sometimes be sent in response to a WM_DDE_DATA message.

When the DDE server sends the WM_DDE_ACK message in response to the WM_DDE_INITIATE message, the SendMessage function is used.

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
DWORD SendMessage(
          HWND    hWndRecipient,
          WORD    wMsg,
          WORD    hWndSender,
          DWORD   lParam);
```

Parameters

hWndRecipient — Identifies the window of the client application.

wMsg — Signals that the WM_DDE_INITIATE message was received. This value must be WM_DDE_ACK.

hWndSender — Identifies the window to the data queues DDE server.

lParam

Arguments

aApplication — Is the low-order word of the lParam parameter. It is an atom that identifies the name of the replying application. In this case, it is the name of the data queues DDE server (DQSERVER).

aTopic — Is the high-order word of the lParam parameter. It is an atom that identifies the topic the data queues DDE server is associated with. The value of this argument is DQSERVER.

For acknowledgement of all other DDE messages, the PostMessage function must be used to send the WM_DDE_ACK message.

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
BOOL PostMessage(
         HWND    hWndRecipient,
         WORD    wMsg,
         WORD    hWndSender,
         DWORD   lParam);
```

Parameters

hWndRecipient — Identifies the window of the program to receive this message. This is either the window of the data queues DDE server or the window of the client application.

wMsg — Signals a previous DDE message was received. This value must be WM_DDE_ACK.

hWndSender — Identifies the window of the program sending the message. This is either the window of the data queues DDE server or the window of the client application.

lParam — Varies depending on the DDE message being acknowledged. If the WM_DDE_ACK message is being sent in response to a WM_DDE_EXECUTE message:

Arguments

wStatus — Is the low-order word of the lParam parameter. It consists of a DDEACK data structure which gives the result of the WM_DDE_EXECUTE request. For a description of the DDEACK structure, see "DDEACK" on page 26-13.

hCommands — Is the high-order word of the lParam parameter. It is a handle to a global memory object containing the command string that was executed. This must be the same object passed for the WM_DDE_EXECUTE message.

If the WM_DDE_ACK message is being sent in response to any other message:

Arguments

wStatus	Is the low-order word of the lParam parameter. It consists of a DDEACK data structure which gives the result of the corresponding DDE request. For a description of the DDEACK structure, see "DDEACK" on page 26-13.
aItem	Is the high-order word of the lParam parameter. It is an atom that specifies the data item for which the response is sent. This may be the same atom used for the message being responded to or, the application sending the WM_DDE_ACK message may delete the received atom and create a new one.

WM_DDE_ADVISE

Purpose

This message is posted by the client application. It requests the data queues DDE server check for data on a specified queue. Whenever data is put on the queue, the server receives the data and sends it to the client using a WM_DDE_DATA message. This process continues until the client application sends a WM_DDE_UNADVISE or WM_DDE_TERMINATE message to the DDE server.

The client application should do the following when posting this message:

- Allocate the global memory object for the hOptions argument by calling GlobalAlloc with the GMEM_DDESHARE option
- Allocate the atom for the aItem argument by calling GlobalAddAtom
- Post the WM_DDE_ADVISE message
- Wait for a WM_DDE_ACK message from the DDE server
- Free the global memory object used for the hOPtions argument, if the DDE server responds with a negative acknowledgement
- Free the global memory object for the hData argument of the WM_DDE_DATA message, for all WM_DDE_DATA messages received

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
BOOL PostMessage(
        HWND    hWndRecipient,
        WORD    wMsg,
        WORD    hWndSender,
        DWORD   lParam);
```

Parameters

hWndRecipient	Identifies the window of the data queues DDE server.
wMsg	Signals the server to notify the client whenever data is put on a specified queue. This value must be WM_DDE_ADVISE.
hWndSender	Identifies the window of the client application.

lParam

Arguments

hOptions — Is the low-order word of the lParam parameter. It is a handle to a global memory data object that consists of a DDEADVISE data structure. This structure specifies how the data is to be sent. For a description of the DDEADVISE structure, see "DDEADVISE" on page 26-14. If the DDE server is able to handle the request successfully, it frees the global memory object.

aItem — Is the high-order word of the lParam parameter. It is an atom that identifies the data queue for which notification is requested. This data queue name must be in the following format:

```
//{system}/{library}/queue{/K=key}
```

Items contained within the braces ({ }) are optional. If the system name is not specified, the default system is used. If the library name is not specified, the library list is searched for the queue. If the key option is not specified, no key is used.

WM_DDE_DATA

Purpose

This message is posted by the data queues DDE server to inform the client of the availability of requested data. It also passes the client a handle to the global memory object where the data resides.

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
BOOL PostMessage(
         HWND    hWndRecipient,
         WORD    wMsg,
         WORD    hWndSender,
         DWORD   lParam);
```

Parameters

hWndRecipient — Identifies the window of the client application.

wMsg — Signals to the client that data is available. This value must be WM_DDE_DATA.

hWndSender — Identifies the window of the data queues DDE server.

lParam

Arguments

hData	Is the low-order word of the lParam parameter. It is a handle to a global memory data object that consists of a DDEDATA data structure. This structure contains the data available to the client application, as well as additional information for the client. For a description of the DDEDATA structure, see "DDEDATA" on page 26-14. The client application is responsible for freeing the global memory object.
aItem	Is the high-order word of the lParam parameter. It is an atom that identifies the data queue for which data is sent. This is the same atom that was used by the client application for the WM_DDE_ADVISE or the WM_DDE_REQUEST message.

WM_DDE_EXECUTE

Purpose

This message is posted by the client application. It requests the DDE server to run a data queues command(s).

The client application should do the following when posting this message:

- Allocate the global memory object for the hCommands argument by calling GlobalAlloc with the GMEM_DDESHARE option
- Post the WM_DDE_EXECUTE message
- Wait for a WM_DDE_ACK message from the DDE server
- Free the global memory object

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
BOOL PostMessage(
         HWND    hWndRecipient,
         WORD    wMsg,
         WORD    hWndSender,
         DWORD   lParam);
```

Parameters

hWndRecipient	Identifies the window of the data queues DDE server.
wMsg	Signals the server to run a data queues command. This value must be WM_DDE_EXECUTE.
hWndSender	Identifies the window of the client application.

lParam

Arguments

reserved — Is the low-order word of the lParam parameter. It is a reserved argument.

hCommands — Is the high-order word of the lParam parameter. It is a handle to a global memory object that contains a null-terminated character string identifying the commands to run. The command string may contain one or more of the following commands:

```
[DLTDTAQ(queue,{library},{system})]
[CLRDTAQ(queue,{library},{system})]
[CRTDTAQ(queue,{library},{system}
  max_len,{seq},{force},{auth}
  {sndr_id},{text},{key_len})]
```

Items contained in braces ({ }) are optional; remaining items must be included in the command. For a description of the parameters, see the corresponding DLL entry point descriptions (EHNDQ_Delete on 26-22, EHNDQ_Clear on 26-18, and EHNDQ_Create) on 26-19.

WM_DDE_INITIATE

Purpose

This message is sent by the client application. It is used to start a DDE conversation with the data queues DDE server.

The client application should do the following when sending this message:

- Allocate the atoms for the aApplication and aTopic arguments by calling GlobalAddAtom.
- Send the WM_DDE_INITIATE message.
- Wait for a WM_DDE_ACK message from the DDE server.
- Delete the atoms for aApplication and aTopic.

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
DWORD SendMessage(
          HWND    hWndRecipient,
          WORD    wMsg,
          WORD    hWndSender,
          DWORD   lParam);
```

Parameters

hWndRecipient — Identifies the window of the data queues DDE server.

wMsg — Signals a DDE conversation should be started with the data queues DDE server. This value must be WM_DDE_INITIATE.

hWndSender — Identifies the window of the client application.

lParam

Arguments

aApplication — Is the low-order word of the lParam parameter. It is an atom that identifies the name of the application with which a conversation is desired. In this case, it is the name of the data queues DDE server (DQSERVER).

aTopic — Is the high-order word of the lParam parameter. It is an atom that identifies the topic the data queues DDE server is associated with. The value of this argument is DQSERVER.

WM_DDE_POKE

Purpose

This message is posted by the client application. It is used to put data on a queue.

The client application should do the following when posting this message:

- Allocate the global memory object for the hData argument by calling GlobalAlloc with the GMEM_DDESHARE option
- Allocate the atom for the altem argument by calling GlobalAddAtom
- Post the WM_DDE_POKE message
- Wait for a WM_DDE_ACK message from the DDE server
- Free the global memory object used for the hOptions argument if the DDE server responds with a negative acknowledgement or the fRelease flag of the DDEPOKE structure is zero

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H>"
BOOL PostMessage(
        HWND    hWndRecipient,
        WORD    wMsg,
        WORD    hWndSender,
        DWORD   lParam);
```

Parameters

hWndRecipient — Identifies the window of the data queues DDE server.

wMsg — Signals data should be put on a queue. This value must be WM_DDE_POKE.

hWndSender — Identifies the window of the client application.

lParam

Arguments

hData — Is the low-order word of the lParam parameter. It is a handle to a global memory data object that consists of a DDEPOKE data structure. This structure contains the data to be put on the queue. For a description of the DDEPOKE structure, see "DDEPOKE" on page 26-15. If the format specified in the DDEPOKE structure is CF_TEXT, the DDE server converts the data from ANSI to EBCDIC. Otherwise, no conversion is done.

altem — Is the high-order word of the lParam parameter. It is an atom that identifies the data queue on which the data should be placed. This data queue name must be in the following format:

```
//{system}/{library}/queue{/K=key}
```

Items contained in the braces ({ }) are optional. If the system name is not specified, the default system is used. If the library name is not specified, the library list is searched. If the key option is not specified, no key is used.

WM_DDE_REQUEST

Purpose

This message is posted by the client application. It requests a packet of data from the DDE server. The request can either be for data from a data queue or for attributes of a data queue.

The client application should do the following when posting this message:

- Allocate the atom for the altem argument by calling GlobalAddAtom.
- Post the WM_DDE_REQUEST message.
- Wait for either a WM_DDE_DATA or a WM_DDE_ACK message from the DDE server. (A WM_DDE_ACK message is sent only if the data is not available.)
- Free the global memory object for the hData argument of the WM_DDE_DATA message, if a WM_DDE_DATA message is received.

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
BOOL PostMessage(
        HWND    hWndRecipient,
        WORD    wMsg,
        WORD    hWndSender,
        DWORD   lParam);
```

Parameters

hWndRecipient
: Identifies the window of the data queues DDE server.

wMsg
: Signals data is requested from the DDE server. This value must be WM_DDE_REQUEST.

hWndSender
: Identifies the window of the client application.

lParam
: **Arguments**

  cfFormat
  : Is the low-order word of the lParam parameter that identifies a predefined or registered clipboard format number. If this format is CF_TEXT, the DDE server converts the data from EBCDIC to ANSI. Otherwise, no conversion is done.

  aItem
  : Is the high-order word of the lParam parameter. It is an atom that identifies the data being requested from the DDE server. This request must be one of the following:

    ```
    QRYDTAQ//{system}/{library}/queue
       or
    RCVDTAQ//{system}/{library}/queue{/K=key}{W=wait}
    ```

    Items contained in braces ({ }) are optional. If the system name is not specified, the default system is used. If the library name is not specified, the library list is searched for the queue. If the key option is not specified, no key is used. If no wait time is specified, zero is used.

    The QRYDTAQ command is used to request the attributes of the data queue. RCVDTAQ is used to request data. If the request is for attributes, the format of the data returned in the DDEDATA structure is defined by the QUERYSTRUCT structure. For a description of the QUERYSTRUCT structure, see "QUERYSTRUCT" on page 26-16.

WM_DDE_TERMINATE

Purpose

This message is posted by either the client application or the data queues DDE server. It requests the client application and the data queues DDE server to end the DDE conversation between them. The sender of this message should wait for a WM_DDE_TERMINATE message from the recipient, acknowledging the receipt of this message. Until the acknowledging WM_DDE_TERMINATE message is received, the sender of this WM_DDE_TERMINATE message should not acknowledge any other messages sent by the recipient of this message.

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
BOOL PostMessage(
         HWND    hWndRecipient,
         WORD    wMsg,
         WORD    hWndSender,
         DWORD   lParam);
```

Parameters

hWndRecipient	Identifies the window of the program to receive this message. This is either the window of the data queues server or the window of the client application.
wMsg	Signals the DDE conversation with the DDE server should be ended. This value must be WM_DDE_TERMINATE.
hWndSender	Identifies the window of the program sending the message. This is either the window of the data queues server or the window of the client application.
lParam	Is a reserved parameter.

WM_DDE_UNADVISE

Purpose

This message is posted by the client application. It requests the DDE server to stop sending WM_DDE_DATA messages every time data is put on a specified queue. To end all active WM_DDE_ADVISE conversations associated with a client, this message should be posted with the cfFormat and altem arguments set to null.

The client application should do the following when posting this message:

- Allocate the atom for the altem argument by calling GlobalAddAtom.
- Post the WM_DDE_UNADVISE message.
- Wait for a WM_DDE_ACK message from the DDE server.

Procedure Declaration

```
#include <WINDOWS.H>
#include <DDE.H>
#include "DQSERVER.H"
BOOL PostMessage(
        HWND    hWndRecipient,
        WORD    wMsg,
        WORD    hWndSender,
        DWORD   lParam);
```

Parameters

hWndRecipient
: Identifies the window of the data queues DDE server.

wMsg
: Signals notification when data is put on a queue should be stopped. This value must be WM_DDE_UNADVISE.

hWndSender
: Identifies the window of the client application.

lParam
: **Arguments**

  cfFormat
  : Is the low-order word of the lParam parameter. It identifies the predefined or registered clipboard format number specified on the WM_DDE_ADVISE. If the value of this argument is null, all active WM_DDE_ADVISE conversations associated with this client, and the queue specified by the altem argument, are ended.

  altem
  : Is the high-order word of the lParam parameter. It is an atom that identifies the data queue for which data is no longer requested. This data queue name must be in the following format:

    ```
    //{system}/{library}/queue{/K=key}
    ```

    Items contained in the braces ({ }) are optional. If the system name is not specified, the default system is used. If the library name is not specified, the library list is searched for the queue. If the key option is not specified, no key is used. If the value of this argument is null, all active WM_DDE_ADVISE conversations associated with this client and the format specified by the cfFormat argument are ended.

DDE Structures

Windows provides a set of predefined data structures to be used when implementing the DDE protocol. The following discussions of each structure describe how data queues makes use of these structures for its implementation of the DDE protocol.

DDEACK

Description

The DDEACK structure is used for the WM_DDE_ACK message. It is used to pass information to the recipient of the message regarding the DDE message that is being responded to.

```
typedef struct {
        unsigned bAppReturnCode:8,
                 reserved:6,
                 fBusy:1,
                 fAck:1;
} DDEACK;
```

Field	Description
bAppReturnCode	Is used to return a return code signalling the result of the DDE request being responded to. Possible values are: DQ_SUCCESS (0x00) DQ_CMD_INVALID (0x01) DQ_PARM_INVALID (0x02) DQ_ATOM_INVALID (0x03) DQ_SYSTEM_INVALID (0x04) DQ_NET_PATH_ERROR (0x05) DQ_LIBRARY_INVALID (0x06) DQ_QUEUE_INVALID (0x07) DQ_NO_MEMORY (0x08) DQ_NO_LOCKS (0x09) DQ_TRANS_TABLE_NOT_FOUND (0x0A) DQ_NO_ADVISE (0x0B) DQ_SEND_ERROR (0x0C) DQ_RECEIVE_ERROR (0x0D) DQ_RECEIVE_REQUEST_ERROR (0x0E) DQ_RECEIVE_DATA_ERROR (0x0F) DQ_QUERY_ERROR (0x10) DQ_CREATE_ERROR (0x11) DQ_DELETE_ERROR (0x12) DQ_CLEAR_ERROR (0x13) DQ_STOP_ERROR (0x14) DQ_COMMAND_SYNTAX (0x15) DQ_MEMORY_LOCKED (0x16) DQ_CANCEL_REQUEST_ERROR (0x17)
reserved	Is a reserved field.
fBusy	Is not used for the data queues implementation of DDE.
fAck	Is a flag that signals whether or not the request was accepted. If this flag is set, the request was accepted.

DDEADVISE

Description

The DDEADVISE structure is used for the WM_DDE_ADVISE message. It is used to specify how data is sent.

```
typedef struct {
        unsigned reserved:14,
                 fDeferUpd:1,
                 fAckReq:1;
        int      cfFormat;
} DDEADVISE;
```

Field	Description
reserved	Is a reserved field.
fDeferUpd	Is ignored by the DDE server.
fAckReq	Is a flag that signals whether the client application wants the DDE server to wait for a WM_DDE_ACK message in response to all WM_DDE_DATA messages sent. This can be used to avoid an overflow of data. If this flag is set, the server sets the fAckReq bit for all WM_DDE_DATA messages posted. This requires the client to respond with a WM_DDE_ACK message.
cfFormat	Is the preferred data type of the client application. If this format is CF_TEXT, the DDE server converts the data from EBCDIC to ANSI. Otherwise, no conversion is done.

DDEDATA

Description

The DDEDATA structure is used for the WM_DDE_DATA message. It is used to store the data available to the client application and additional information for the client.

```
typedef struct {
        unsigned unused:12,
                 fResponse:1,
                 fRelease:1,
                 reserved:1,
                 fAckReq:1;
        int      cfFormat;
        BYTE     Value[1];
} DDEDATA;
```

Field	Description
unused	Is an unused field.
fResponse	Is a flag that signals which message that the available data being sent is responding to. If this flag is set, the data is being sent in response to a WM_DDE_REQUEST message. Otherwise, it is being sent in response to a WM_DDE_ADVISE message.

fRelease	Is a flag that signals whether the client application must free the global memory object used for the hData argument. The DDE server always sets this flag. Therefore, the client application must free the global memory object.
reserved	Is a reserved field.
fAckReq	Is a flag that signals whether the client application is required to post a WM_DDE_ACK in response to the WM_DDE_DATA message. If this flag is set, the client application must post a WM_DDE_ACK message.
cfFormat	Specifies the format in which the data is returned. If this format is CF_TEXT, the DDE server converts the data from EBCDIC to ANSI. Otherwise, no conversion is done.
Value	Contains the data being returned.

DDEPOKE

Description

The DDEPOKE structure is used for the WM_DDE_POKE message. It is used to pass the data to be put on the queue.

```
typedef struct {
        unsigned unused:13,
                 fRelease:1,
                 fReserved:2;
        int      cfFormat;
        BYTE     Value[1];
} DDEPOKE;
```

Field	Description
unused	Is an unused field.
fRelease	Is a flag that signals which application is responsible for freeing the global memory object used for the hData argument. If this flag is set, the DDE server must free the global memory object. Otherwise, the client application must free it.
fReserved	Is a reserved field.
cfFormat	Is the preferred data type of the client application. If this format is CF_TEXT, the DDE server converts the data from ANSI to EBCDIC. Otherwise, no conversion is done.
Value	Contains the data to be put on the queue.

Data queues also provides a data structure definition to be used with the WM_DDE_DATA message. This structure is defined as follows and can be found in the DQSERVER.H file on the PC Support/400 tools folder (QIWSTOOL). For information on the tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

QUERYSTRUCT

Description

This structure is used for the WM_DDE_DATA message. It is used to define the format of the attributes returned to the client application, in response to a WM_DDE_REQUEST message.

```
typedef struct {
   long                lMaxLength;
   int                 nSequence;
   int                 nForce;
   BOOL                bSenderID;
   char                szText[51];
   int                 nKeyLen;
                  }
   QUERYSTRUCT;
```

Field	Description
lMaxLength	Specifies the length of each data queue entry.
nSequence	Identifies the ordering sequence of data in the data queue. Possible values are: EHNDQ_LIFO (0x0000) (last in, first out) EHNDQ_FIFO (0x0001) (first in, first out) EHNDQ_KEYED (0x0002) (orders by key)
nForce	Specifies whether data is forced to auxiliary storage. Possible values are: EHNDQ_FALSE (0x0000) (messages are not forced to auxiliary storage) EHNDQ_TRUE (0x0001) (messages are forced to auxiliary storage)
bSenderID	Specifies whether sender information is kept with each data queue entry. Possible values are: EHNDQ_FALSE (0x0000) (sender information is not kept) EHNDQ_TRUE (0x0001) (sender information is kept)
szText	Contains a description of the data queue.
nKeyLen	Identifies the length of the key field.

Data Queues DLL Routines

This section describes the individual routines, data structures, and return codes that make up the data queues Windows API. The descriptions make reference to predefined constants, data structures, and function prototypes that can be found in the file EHNDQW.H in the PC Support tools folder (QIWSTOOL). The PC Support tools folder also contains a sample program that demonstrates how to use the data queues Windows API. For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

EHNDQ_CancelRequest

Purpose

This routine cancels a previous EHNDQ_ReceiveRequest when an EHNDQ_ReceiveData has not already been issued and data returned. Any data that has been received is discarded.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_CancelRequest(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_CancelRequestKeyed

Purpose

This routine cancels a previous EHNDQ_ReceiveRequestKeyed when an EHNDQ_ReceiveDataKeyed has not already been issued and data returned. Any data that has been received is discarded.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_CancelRequestKeyed(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LPSTR         lpszKey);
```

Parameters

hWnd	Identifies the current window of the application.
lpszQueueName	Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*. If the library name is not specified, the library list is searched for the queue.
lpszQueueLocation	Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.
lpszKey	Points to a null-terminated character string containing the same key specified in the lpszKey parameter on the corresponding EHNDQ_ReceiveRequestKeyed parameter.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_Clear

Purpose

This routine clears all messages from a specified data queue.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_Clear(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation);
```

Parameters

hWnd	Identifies the current window of the application.
lpszQueueName	Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*. If the library name is not specified, the library list is searched for the queue.
lpszQueueLocation	Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_Create

Purpose

This routine creates a nonkeyed data queue.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_Create(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LONG          lMaxLength,
                    int           nSequence,
                    int           nForce,
                    int           nAuthority,
                    int           nSenderID,
                    LPSTR         lpszText);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue to create. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the data queue is created in the current library.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue should be created. If this pointer is null or points to a string with a length of zero, the default system is used.

lMaxLength — identifies the maximum length for a message on the data queue. The maximum length is 31 744 bytes.

nSequence — Identifies how data should be retrieved from the data queue. Possible values are:

EHNDQ_LIFO (0x0000) (last in, first out)
EHNDQ_FIFO (0x0001) (first in, first out)

nForce — Specifies whether messages are forced to auxiliary storage before they are added to the queue. Possible values are:

EHNDQ_FALSE (0x0000) (messages are not forced to auxiliary storage)
EHNDQ_TRUE (0x0001) (messages are forced to auxiliary storage)

This provides for data integrity but may impact performance.

nAuthority — Specifies the authority to give users who do not have specific authority to the queue. Possible values are:

0 EHNDQ_ALL (0x0000) All authority
1 EHNDQ_EXCLUDE (0x0001) Exclude authority
2 EHNDQ_CHANGE (0x0002) Change authority
3 EHNDQ_USE (0x0003) Use authority
4 EHNDQ_LIBCRTAUT (0x0004) Library create authority

Note: Users who do not have specific authority to a queue require at least change authority to send messages to a queue.

nSenderID — Indicates whether the process that adds messages to the queue should add information about itself to the message. Possible values are:

EHNDQ_FALSE (0x0000) (information should not be added)
EHNDQ_TRUE (0x0001) (information should be added)

lpszText — Points to a null-terminated character string containing the text description for the data queue. The maximum length of the text is 50 characters.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_CreateKeyed

Purpose

This routine creates a keyed data queue.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_CreateKeyed(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LONG          lMaxLength,
                    int           nForce,
                    int           nAuthority,
                    int           nSenderID,
                    LPSTR         lpszText,
                    int           nKeyLen);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue to create. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the data queue will be created in the current library.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue should be created. If this pointer is null or points to a string with a length of zero, the default system is used.

lMaxLength — Identifies the maximum length for a message on the data queue. The maximum length is 31744 bytes.

nForce — Specifies whether messages are to be forced to auxiliary storage before they are added to the queue. Possible values are:

EHNDQ_FALSE (0x0000) (messages are not forced to auxiliary storage)
EHNDQ_TRUE (0x0001) (messages are forced to auxiliary storage)

This provides for data integrity but may impact performance.

nAuthority — Specifies the authority to give users who do not have specific authority to the queue. Possible values are:

0 EHNDQ_ALL (0x0000) All authority
1 EHNDQ_EXCLUDE (0x0001) Exclude authority
2 EHNDQ_CHANGE (0x0002) Change authority
3 EHNDQ_USE (0x0003) Use authority
4 EHNDQ_LIBCRTAUT (0x0004) Library create authority

Note: Users who do not have specific authority to a queue require at least change authority to send messages to a data queue.

nSenderID — Indicates whether the process that adds messages to the queue should add information about itself to the message. Possible values are:

EHNDQ_FALSE (0x0000) (information should not be added)
EHNDQ_TRUE (0x0001) (information should be added)

lpszText — Points to a null-terminated character string containing the text description for the data queue. The maximum length of the text is 50 characters.

nKeyLen — Identifies the length of the key for the data queue. The maximum length is 256 bytes.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_Delete

Purpose

This routine clears all messages from a specified data queue and then deletes the data queue.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_Delete(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue to delete. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_GetCapability

Purpose

This routine returns the functional level of the data queue module supporting communications to the host system.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_GetCapability(
                    HWND          hWnd,
                    int far *     lpnFunctionalLevel);
```

Parameters

hWnd — Identifies the current window of the application.

lpnFunctionalLevel — Points to an integer variable that is used to return the functional level. The integer value that is returned indicates the functional level of the module that the data queues DLL uses to send messages to and receive messages from the host system.

1 Data queues supports the following entry points:

EHNDQ_Create	Create data queue
EHNDQ_CreateKeyed	Create keyed data queue
EHNDQ_Send	Send data to a queue
EHNDQ_SendKeyed	Send data to a keyed queue
EHNDQ_Put	Put data to a queue
EHNDQ_PutKeyed	Put data to a keyed queue
EHNDQ_Receive	Receive data from a queue
EHNDQ_ReceiveKeyed	Receive data from a keyed queue
EHNDQ_ReceiveRequest	Receive request for data
EHNDQ_ReceiveRequestKeyed	Receive request for data from a keyed queue
EHNDQ_ReceiveData	Receive data previously requested
EHNDQ_ReceiveDataKeyed	Receive data previously requested from a keyed queue
EHNDQ_CancelRequest	Cancel previous request for data
EHNDQ_CancelRequestKeyed	Cancel previous request for data from a keyed queue
EHNDQ_Clear	Clear data queue
EHNDQ_Query	Query data queue attributes
EHNDQ_Delete	Delete data queue
EHNDQ_SetMode	Set data queues mode
EHNDQ_Stop	Stop data queues conversations
EHNDQ_GetCapability	Get capability
EHNDQ_GetMessage	Get error message

>1 Reserved for future enhancements to data queues.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_GetMessage

Purpose

This routine retrieves an error message associated with the last data queue API call to a specified host system to result in a nonzero return code. To obtain all messages associated with a request, this routine should be called multiple times until the value returned through the lpnMsgLength parameter is zero. If an error message is not retrieved when the error occurs, it is discarded when the next successful API is called.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_GetMessage(
                    HWND          hWnd,
                    LPSTR         lpszQueueLocation,
                    LPSTR         lpszMsgBuffer,
                    int far *     lpnMsgLength);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system associated with the error. If this pointer is null or points to a string with a length of zero, the default system is used.

lpszMsgBuffer — Points to a character buffer that is used to return the error message. This buffer must be at least 152 characters in length.

If the return code received on the associated request is EHNDQ_COMMERROR (X'0001') or EHNDQ_PCERROR (X'0003'), the message is returned in this buffer in the following form:

xxxx - Text

where xxxx is the PC Support/400 message number.

If the return code received on the associated request is EHNDQ_EXCEPTERROR (X'0002'), the message is returned in the following form:

xxxxxxx: Text

where xxxxxxx is the host message identifier.

lpnMsgLength — Points to an integer variable that is used to store the length of the retrieved message.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41.

EHNDQ_Put

Purpose

This routine sends data or messages to a nonkeyed data queue. To improve performance, the personal computer data queues program returns to the caller of this routine without waiting for acknowledgement from the host data queue program. Use the EHNDQ_Send routine instead if receiving data has a higher priority than performance. To obtain both the performance and data verification advantages, call EHNDQ_Put for all requests to put messages on a queue, except for the last request which can be made with a call to EHNDQ_Send.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_Put(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LPSTR         lpachDataBuffer,
                    LONG          lDataLength);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

lpachDataBuffer — Points to a buffer that contains the data or message to be put on the data queue.

lDataLength — Identifies the length of the message in the buffer pointed to by the lpachDataBuffer parameter.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_PutKeyed

Purpose

This routine sends data or messages to a keyed data queue. To improve performance, the personal computer data queues program returns to the caller of this routine without waiting for acknowledgement from the host data queue program. Use the EHNDQ_SendKeyed routine instead if receiving data has a higher priority than performance. To obtain both performance and data verification advantages, call EHNDQ_PutKeyed for all requests except the last which can be made with a call to EHNDQ_SendKeyed.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_PutKeyed(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LPSTR         lpachDataBuffer,
                    LONG          lDataLength,
                    LPSTR         lpszKey);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

lpachDataBuffer — Points to a buffer that contains the data or message to be put on the data queue.

lDataLength — Identifies the length of the message in the buffer pointed to by the lpachDataBuffer parameter.

lpszKey — Points to a null-terminated character string that identifies the key associated with the data being put on the data queue. This key must be the same length as specified when the queue was created (not including the null terminator). EHNDQ_Query can be used to find the length used when the queue was created.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_Query

Purpose

This routine returns information about the specified data queue. The information returned contains the attributes specified when the queue was created.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_Query(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LONG far *    lplMaxLength,
                    int far *     lpnSequence,
                    int far *     lpnForce,
                    int far *     lpnSenderID,
                    LPSTR         lpszText,
                    int far *     lpnKeyLen);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

lplMaxLength — Points to a LONG integer variable that is used to return the maximum message length for the data queue.

lpnSequence — Points to an integer variable that is used to return the value that specifies how messages are retrieved from the queue. Possible values are:

EHNDQ_LIFO (0x0000) (last in, first out)
EHNDQ_FIFO (0x0001) (first in, first out)
EHNDQ_KEYED (0x0002) (orders by key)

lpnForce — Points to an integer variable that is used to store whether or not messages are forced to auxiliary storage before they are added to the queue. Possible values are:

EHNDQ_FALSE (0x0000) (messages are not forced to auxiliary storage)
EHNDQ_TRUE (0x0001) (messages are forced to auxiliary storage)

lpnSenderID — Points to an integer variable that is used to store whether or not the process that adds messages to the queue should add information about itself to the message. Possible values are:

EHNDQ_FALSE (0x0000) (information should not be added)
EHNDQ_TRUE (0x0001) (information should be added)

lpszText — Points to a character buffer that is used to store the null-terminated character string containing the text description for the data queue. The buffer should be 51 bytes in length.

lpnKeyLen — Points to an integer variable that is used to store the length of the key for a message on the data queue.

Note: This parameter is valid only if the queue is a keyed queue as indicated by the lpnSequence parameter.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_Receive

Purpose

This routine retrieves data or messages from a nonkeyed data queue. When this routine is called, all other activity on the personal computer is suspended. If no data is available on the specified queue, the personal computer data queues program waits until a message is available or the specified time-out value has expired. Therefore, the EHNDQ_ReceiveRequest function may be used in conjunction with the EHNDQ_ReceiveData Function instead of using this routine.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_Receive(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LONG          lWaitTime,
                    INT           nSenderID,
                    LPSTR         lpachDataBuffer,
                    LONG far *    *lplLengthRecord,
                    LPSTR         lpachSenderIDInfo);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

lWaitTime — Identifies a LONG integer indicating how long to wait for a message to be retrieved from the AS/400 system. The possible values are:

A positive value indicates the number of seconds to wait.
A negative value indicates to wait until a message is retrieved.
A value of zero indicates not to wait and return immediately if no messages are available.

nSenderID — Indicates whether information about the process that added the message to the queue should be returned. Possible values are:

EHNDQ_FALSE (0x0000) (information should not be added)
EHNDQ_TRUE (0x0001) (information should be added)

lpachDataBuffer — Points to a buffer that is used to store the retrieved message.

lplLengthRecord — Points to a LONG integer variable that contains the length of the buffer pointed to by the lpachDataBuffer parameter. On output, it is used to return the length of the retrieved message placed in the buffer.

lpachSenderIDInfo — Points to a buffer that is used to store information about the process that added the retrieved message to the queue. If the value of the nSenderID parameter is EHNDQ_FALSE (0x0000), this parameter is ignored. If the value of the nSenderID parameter is EHNDQ_TRUE (0x0001), the following strings, padded with blanks, are stored in this buffer.

Job name (10 characters)
User name (10 characters)
Job number (6 characters)
User profile (10 characters)

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_ReceiveData

Purpose

This routine determines if a request to retrieve a message from a nonkeyed data queue made by a previous call to EHNDQ_ReceiveRequest has completed. If the request has completed, this routine returns the length of the retrieved message written where specified on the EHNDQ_ReceiveRequest call.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_ReceiveData(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

Additional Information

The current mode determines whether read operations are destructive (receive) or nondestructive (peek). See "EHNDQ_SetMode" on page 26-39 for more information.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_ReceiveDataKeyed

Purpose

This routine determines if a request to retrieve a message from a keyed data queue (made by a previous call to EHNDQ_ReceiveRequestKeyed) has completed. If the request has completed, this routine returns the length of the retrieved message written where specified on the EHNDQ_ReceiveRequest call.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_ReceiveDataKeyed(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LPSTR         lpszKey);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

lpszKey — Points to a null-terminated character string containing the same key specified in the lpszKey parameter on the corresponding EHNDQ_ReceiveRequestKeyed parameter.

Additional Information

The current mode determines whether read operations are destructive (receive) or nondestructive (peek). See "EHNDQ_SetMode" on page 26-39 for more information.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_ReceiveKeyed

Purpose

This routine retrieves data or messages from a keyed data queue. When this routine is called, all other activity on the personal computer is suspended. If no message is available on the specified queue, the personal computer data queues program waits until a message is available or the specified time-out value has expired. Therefore, the EHNDQ_ReceiveRequestKeyed function in conjunction with the EHNDQ_ReceiveDataKeyed function may be used in place of this routine.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_ReceiveKeyed(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LONG          lWaitTime,
                    LPSTR         lpszKey,
                    LPSTR         lpachSearchOrder,
                    int           nSenderID,
                    LPSTR         lpachDataBuffer,
                    LONG far *    lplLengthRecord,
                    LPSTR         lpszKeyOutput,
                    LPSTR         lpachSenderIDInfo);
```

Parameters

hWnd
: Identifies the current window of the application.

lpszQueueName
: Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

  If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation
: Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

lWaitTime
: Identifies a LONG integer indicating how long to wait for a message to be retrieved from the AS/400 system. The possible values are:

  A positive value indicates the number of seconds to wait.
  A negative value indicates to wait until data is retrieved.
  A value of zero indicates not to wait and return immediately if no messages are available.

lpszKey
: Points to a null-terminated character string containing the key to be used in conjunction with the lpachSearchOrder parameter. This pointer determines which message should be retrieved from the queue. This key must be the same length as specified when the queue was created (not including the null terminator). EHNDQ_Query can be used to find the length used when the queue was created.

lpachSearchOrder
: Identifies the 2-character relational operator to be used in conjunction with the lpszKey parameter. Possible values are:

  EQ Equal to
  GT Greater than
  LT Less than
  GE Greater than or equal to
  LE Less than or equal to
  NE Not equal

nSenderID — Indicates whether information about the process that added the message to the queue should be returned. Possible values are:

EHNDQ_FALSE (0x0000) (information should not be added)
EHNDQ_TRUE (0x0001) (information should be added)

lpachDataBuffer — Points to a buffer that is used to store the retrieved message.

lplLengthRecord — Points to a LONG integer variable that contains the length of the buffer pointed to by the lpachDataBuffer parameter. When returning data it is used to return the length of the retrieved message placed in the buffer.

lpszKeyOutput — Points to a character buffer that is used to return the null-terminated key of the retrieved message.

lpachSenderIDInfo — Points to a buffer that is used to store information about the process that added the retrieved data to the queue. If the value of the nSenderID parameter is EHNDQ_FALSE (0x0000), this parameter is ignored. If the value of the nSenderID parameter is EHNDQ_TRUE (0x0001), the following strings, padded with blanks, are stored in this buffer.

- Job name (10 characters)
- User name (10 characters)
- Job number (6 characters)
- User profile (10 characters)

Additional Information

The current mode determines whether read operations are destructive (receive) or nondestructive (peek). See "EHNDQ_SetMode" on page 26-39 for more information.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_ReceiveRequest

Purpose

This routine requests messages from a nonkeyed data queue. The personal computer data queues program returns control to the caller of this routine without waiting for the message to be returned. The EHNDQ_ReceiveData routine can be called to determine when the requested message has been retrieved. This routine avoids the potential for a long suspension of all other personal computer activity by the EHNDQ_Receive routine.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_ReceiveRequest(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LONG          lWaitTime,
                    int           nSenderID,
                    LPSTR         lpachDataBuffer,
                    LONG far *    lplLengthRecord,
                    LPSTR         lpachSenderIDInfo);
```

Parameters

hWnd
: Identifies the current window of the application.

lpszQueueName
: Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

  If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation
: Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

lWaitTime
: A LONG integer indicating how long to wait for a message to be retrieved from the AS/400 system. The possible values are:

  A positive value indicates the number of seconds to wait.
  A negative value indicates to wait until a message is retrieved.
  A value of zero indicates not to wait and return immediately if no messages are available.

nSenderID
: Indicates whether information about the process that added the message to the queue should be returned. Possible values are:

  EHNDQ_FALSE (0x0000) (information should not be added)
  EHNDQ_TRUE (0x0001) (information should be added)

lpachDataBuffer
: Points to a buffer that is used to store the retrieved message.

lplLengthRecord
: Points to a LONG integer variable that contains the length of the buffer pointed to by the lpachDataBuffer parameter. On output, this is used to store the length of the retrieved message.

lpachSenderIDInfo
: Points to a buffer that is used to store information about the process that added the retrieved data to the queue. If the value of the nSenderID parameter is EHNDQ_FALSE (0x0000), this parameter is ignored. If the value of the nSenderID parameter is EHNDQ_TRUE (0x0001), the following strings, padded with blanks, are stored in this buffer.

- Job name (10 characters)
- User name (10 characters)
- Job number (6 characters)
- User profile (10 characters)

Additional Information

If EHNDQ_ReceiveRequest is called more than once before issuing the corresponding call to EHNDQ_ReceiveData, the following buffers must be unique:

lpachDataBuffer
lplLengthRecord
lpachSenderIDInfo

The current mode determines whether read operations are destructive (receive) or nondestructive (peek). See "EHNDQ_SetMode" on page 26-39 for more information.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_ReceiveRequestKeyed

Purpose

This routine requests messages from a keyed data queue. The personal computer data queues program returns control to the caller of this routine without waiting for the message to be returned. The EHNDQ_ReceiveDataKeyed routine can be called to determine when the requested message has been retrieved. This routine avoids the potential for a long suspension of all other personal computer activity by the EHNDQ_ReceiveKeyed routine.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_ReceiveRequestKeyed(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LONG          lWaitTime,
                    LPSTR         lpszKey,
                    LPSTR         lpachSearchOrder,
                    int           nSenderID,
                    LPSTR         lpachDataBuffer,
                    LONG far *    lplLengthRecord,
                    LPSTR         lpszKeyOutput,
                    LPSTR         lpachSenderIDInfo);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueName — Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*.

If the library name is not specified, the library list is searched for the queue.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.

lWaitTime — A LONG integer indicating how long to wait for a message to be retrieved from the AS/400 system. The possible values are:

A positive value indicates the number of seconds to wait.
A negative value indicates to wait until a message is retrieved.
A value of zero indicates not to wait and return immediately if no messages are available.

lpszKey — Points to a null-terminated character string containing the key to be used in conjunction with the lpachSearchOrder parameter. This pointer determines which message should be retrieved from the queue. This key must be the same length as specified when the queue was created (not including the null terminator). EHNDQ_Query can be used to find the length used when the queue was created.

lpachSearchOrder — Identifies the 2-character relational operator to be used in conjunction with the lpszKey parameter. Possible values are:

EQ Equal to
GT Greater than
LT Less than
GE Greater than or equal to
LE Less than or equal to
NE Not equal

SenderID — Indicates whether to return information about the process that added the message to the queue.

lpachDataBuffer — Points to a buffer that is used to store the retrieved message.

lplLengthRecord — Points to a LONG integer variable that contains the length of the buffer pointed to by the lpachDataBuffer parameter. For returning data it is used to return the length of the retrieved message placed in the buffer.

lpszKeyOutput — Points to a character buffer that is used to return the null-terminated key of the retrieved message.

lpachSenderIDInfo — Points to a buffer that is used to store information about the process that added the retrieved data to the queue. If the value of the nSenderID parameter is EHNDQ_FALSE (0x0000), this parameter is ignored. If the value of the nSenderID param-

eter is EHNDQ_TRUE (0x0001), the following strings, padded with blanks, are stored in this buffer.

- Job name (10 characters)
- User name (10 characters)
- Job number (6 characters)
- User profile (10 characters)

Additional Information

If EHNDQ_ReceiveRequestKeyed is called more than once before the corresponding call to EHNDQ_ReceiveDataKeyed, the following buffers must be unique:

lpachDataBuffer
lplLengthRecord
lpachSenderIDInfo
lpszKeyOutput

The current mode determines whether read operations destructive (receive) or nondestructive (peek). See "EHNDQ_SetMode" on page 26-39 for more information.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_Send

Purpose

This routine sends data or messages to a nonkeyed data queue. To ensure data is not lost during transmission, the personal computer data queues program waits for acknowledgement from the host data queue program before returning to the caller of this routine. Use the EHNDQ_Put routine instead if performance has a higher priority than receiving an acknowledgement. To obtain both the performance and data verification advantages, call EHNDQ_Put for all requests to send messages to a queue, except for the last request which can be made with a call to EHNDQ_Send.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_Send(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LPSTR         lpachDataBuffer,
                    LONG          lDataLength);
```

Parameters

hWnd	Identifies the current window of the application.
lpszQueueName	Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*. If the library name is not specified, the library list is searched for the queue.
lpszQueueLocation	Points to a null-terminated character string containing the name of the system where the data queue exists. If this pointer is null or points to a string with a length of zero, the default system is used.
lpachDataBuffer	Points to a buffer that contains the message to be put on the data queue.
lDataLength	Identifies the length of the message in the buffer pointed to by the lpachDataBuffer parameter.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_SendKeyed

Purpose

This routine sends data or messages to a keyed data queue. To ensure data is not lost during transmission, the personal computer data queues program waits for acknowledgement from the host data queue program before returning to the caller of this routine. Use the EHNDQ_PutKeyed routine instead if performance has a higher priority than receiving an acknowledgement. To obtain both the performance and data verification advantages, call EHNDQ_PutKeyed for all requests to send messages to a queue, except for the last request which can be made with a call to EHNDQ_SendKeyed.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_SendKeyed(
                    HWND          hWnd,
                    LPSTR         lpszQueueName,
                    LPSTR         lpszQueueLocation,
                    LPSTR         lpachDataBuffer,
                    LONG          lDataLength,
                    LPSTR         lpszKey);
```

Parameters

hWnd	Identifies the current window of the application.
lpszQueueName	Points to a null-terminated character string containing the name of the data queue. This string must be in the form of *library/queue* or *queue*. If the library name is not specified, the library list is searched for the queue.
lpszQueueLocation	Points to a null-terminated character string containing the name of the system to set the conversion mode for. If this pointer is null or points to a string with a length of zero, the default system is used.
lpachDataBuffer	Points to a buffer that contains the message to be put on the data queue.
lDataLength	Identifies the length of the message in the buffer pointed to by the lpachDataBuffer parameter.
lpszKey	Points to a null-terminated character string that identifies the key associated with the message being put on the data queue. This key must be the same length as specified when the queue was created (not including the null terminator). EHNDQ_Query can be used to find the length used when the queue was created.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_SetMode

Purpose

This routine sets the current running mode of data queues. The mode controls whether the data is converted between ASCII and EBCDIC, and whether receive operations are destructive (receive) or nondestructive (peek). The mode is set for all subsequent calls.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_SetMode(
                    HWND          hWnd,
                    LPSTR         lpszQueueLocation,
                    LONG          lMode);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system to set the conversion mode for. If this pointer is null or points to a string with a length of zero, the default system is used.

lMode — Specifies whether data is converted between ASCII and EBCDIC, and whether any following instructions are destructive (receive) or nondestructive (peek). Use the following constant values to set the lMode parameter:

Constant Value	Function
DQ_YES_XLAT	Translate between ASCII and EBCDIC
DQ_NO_XLAT	No translation
DQ_YES_PEEK	Peek data queue
DQ_NO_PEEK	Receive data queue
DQ_XLAT_PEEK	Translate and peek
DQ_XLATN_PEEKY	No translate and peek
DQ_XLATY_PEEKN	Translate and receive
DQ_NO_XLAT_PEEK	No translate and receive

Additional Information

A data queue conversation must already exist with the host system.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API" on page 26-41. If a nonzero return code is received, the EHNDQ_GetMessage routine can be called to obtain additional information about the failure.

EHNDQ_Stop

Purpose

This routine signals all data queues activity for a specified system has ended. If no other applications are currently using data queues on the specified system, the personal computer data queues program ends the data queues conversations for the system and frees all system resources associated with the conversations. The host data queues program is also stopped. The next call to data queues establishes communications again and restarts the host data queues program.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDQW.H"
int far pascal EHNDQ_Stop(
                    HWND          hWnd,
                    LPSTR         lpszQueueLocation);
```

Parameters

hWnd — Identifies the current window of the application.

lpszQueueLocation — Points to a null-terminated character string containing the name of the system to stop conversations with. If this pointer is null or points to a string with a length of zero, the default system is used. A value of *ALL indicates all data queues activity for this application has ended.

Return Codes

For return codes, see "Return Codes for the Data Queues Windows API."

Return Codes for the Data Queues Windows API

Functions in the data queues Windows API use the following return codes. These are constants defined in EHNDQW.H.

Table 26-1. Return Codes for the Data Queues Windows API

Return Code	Hex Value	Description
EHNDQ_SUCCESS	0000	Successful.
EHNDQ_COMMERROR	0001	A communications error occurred.
EHNDQ_EXCEPTERROR	0002	An exception error occurred on the remote AS/400 system.
EHNDQ_PCERROR	0003	A PC resource error has occurred (for example, extended or Windows memory could not be allocated).

Chapter 27. Data Transform Windows Application Program Interface

The data transform API provides the capability to convert data between AS/400 and PC formats. Translation may be needed when sending and receiving data to and from the AS/400. The data transform API supports conversion of text and numerous numeric formats.

This chapter describes the individual routines and return codes which make up the data transform Windows API. The descriptions make reference to predefined constants, data structures, and function prototypes which can be found in the file EHNDTW.H in the PC Support tools folder (QIWSTOOL). The PC Support tools folder also contains a sample program which demonstrates how to use the data transform Windows API. For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

For information on the requirements for developing and running applications that use the PC Support Windows APIs, see "Requirements for Using Windows APIs" on page 19-9.

Data Transform Windows API Routines

The following discussions of each data transform Windows API routine describe in detail:

- Purpose
- Procedure declaration
- Parameters
- Return codes

EHNDT_ANSIToEBCDIC

Purpose

This function translates a string from the Windows ANSI code page to EBCDIC. The router must be loaded so that this routine can access the ASCII-to-EBCDIC translation table.

If the target string is not large enough to contain the translated string, the translation stops at the end of the target string. If the target string is larger than required, it is filled with blanks to the end of the string.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned int _far _pascal EHNDT_ANSIToEBCDIC(
                HWND            hWnd,
                LPSTR           lpsTarget,
                LPSTR           lpsSource,
                unsigned int    wSource,
                LPWORD          lpwTarget );
```

Parameters

hWnd	Identifies the current window of the application.
lpsTarget	Points to the target (translated) string.
lpsSource	Points to the source (ANSI) string to convert.
wSource	Identifies the length of the source string in bytes.
lpwTarget	Points to a word variable containing the size of the target buffer. This variable will be updated with the total number of translated characters in the target buffer.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If the router is not loaded, EHNDT_A2E_TABLE_NOT_FOUND (X'FFFC') is returned. If an error occurs while attempting to allocate a temporary buffer, EHNDT_MEMALLOC(X'FFFF') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_ASCII6ToBin2

Purpose

This function converts a 6-byte ASCII string to a 2-byte integer.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_ASCII6ToBin2(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to a 2-byte integer stored with the most significant byte first (AS/400 format).
lpSource	Points to a 6-byte array of ASCII characters. Valid characters are 0 through 9, +, –, and a space. Leading and trailing spaces are ignored.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If an overflow error occurs, EHNDT_OVERFLOW (X'FFFE') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_ASCII11ToBin4

Purpose

This function converts an 11-byte ASCII string to a 4-byte integer.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_ASCII11ToBin4(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to a 4-byte integer stored with the most significant byte first (AS/400 format).
lpSource	Points to an 11-byte array of ASCII characters. Valid characters are 0 through 9, +, –, and a space. Leading and trailing spaces are ignored.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If an overflow error occurs, EHNDT_OVERFLOW (X'FFFE') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_ASCIIToEBCDIC

Purpose

This function translates a string from ASCII to EBCDIC. The router must be loaded so that this routine can access the ASCII-to-EBCDIC translation table.

If the target string is not large enough to contain the translated string, the translation stops at the end of the target string. If the target string is larger than required, it is blank filled to the end of the string.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned int _far _pascal EHNDT_ASCIIToEBCDIC(
                HWND           hWnd,
                LPSTR          lpsTarget,
                LPSTR          lpsSource,
                unsigned int   wSource,
                LPWORD         lpwTarget );
```

Parameters

hWnd	Identifies the current window of the application.
lpsTarget	Points to the target (translated) string.
lpsSource	Points to the source (ASCII) string to convert.
wSource	Identifies the length of the source string in bytes.
lpwTarget	Points to a word variable containing the size of the target buffer. This variable will be updated with the total number of translated characters in the target buffer.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If the router is not loaded, EHNDT_A2E_TABLE_NOT_FOUND (X'FFFC') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_ASCIIToHex

Purpose

This function converts ASCII data to hexadecimal data.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_ASCIIToHex(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length );
```

Parameters

hWnd	Identifies the current window of the application
lpTarget	Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter.
lpSource	Points to the ASCII data to be converted. The length of this data must be greater than or equal the value of the length parameter multiplied by two.
length	Identifies the number of bytes of hexadecimal data to be stored at the lpTarget parameter.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_ASCIIToPacked

Purpose

This function converts ASCII numeric data to packed decimal data in EBCDIC format.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_ASCIIToPacked(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length,
                int            decimal_position );
```

Parameters

hWnd — Identifies the current window of the application.

lpTarget — Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter.

lpSource — Points to the ASCII numeric data. Leading and trailing spaces are ignored.

decimal_position	Data Length
≠ 0	≥ length parameter x 2 + 1
0	≥ length parameter x 2

length — Identifies the number of bytes of packed decimal data to be stored at lpTarget.

decimal_position — Identifies the position of the decimal in the ASCII data.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If an overflow error occurs, EHNDT_OVERFLOW (X'FFFE') is returned. If an error occurs while attempting to allocate a temporary buffer, EHNDT_MEMALLOC (X'FFFF') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_ASCIIToZoned

Purpose

This function converts ASCII numeric data to zoned decimal data.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_ASCIIToZoned(
                HWND          hWnd,
                char _far *   lpTarget,
                char _far *   lpSource,
                int           length,
                int           decimal_position );
```

Parameters

hWnd — Identifies the current window of the application.

lpTarget — Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter.

lpSource — Points to the ASCII data to be converted. Leading and trailing spaces are ignored.

decimal_position	Data Length
≠ 0	≥ length parameter + 2
0	= length parameter + 1

length — Identifies the number of bytes of zoned decimal data to be stored at lpTarget.

decimal_position — Identifies the position of the decimal in the ASCII data. This value must be less than or equal to the value of the length parameter.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If an overflow error occurs, EHNDT_OVERFLOW (X'FFFE') is returned. If an error occurs while attempting to allocate a temporary buffer, EHNDT_MEMALLOC (X'FFFF') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_Bin2ToASCII6

Purpose

This function converts a 2-byte integer to a 6-byte ASCII string.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_Bin2ToASCII6(
                HWND          hWnd,
                char _far *   lpTarget,
                char _far *   lpSource );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the 6-byte target buffer. The returned data will not be null-terminated.
lpSource	Points to a 2-byte integer to be transformed to 6-byte ASCII. The integer must be stored with the most significant byte first (AS/400 format).

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned.

EHNDT_Bin4ToASCII11

Purpose

This function converts a 4-byte integer to an 11-byte ASCII string.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_Bin4ToASCII11(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the 11-byte target buffer. The returned data will not be null-terminated.
lpSource	Points to a 4-byte integer to be transformed to 11-byte ASCII. The integer must be stored with the most significant byte first (AS/400 format).

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned.

EHNDT_DosToPacked

Purpose

This function converts packed decimal data in ASCII format to packed decimal data in EBCDIC format.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_DosToPacked(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter.
lpSource	Points to the packed decimal data in ASCII format. The sign half-byte must be X'B' or X'3'.
length	Identifies the number of bytes to be converted.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_DosToZoned

Purpose

This function converts packed decimal in ASCII format to zoned decimal data in EBCDIC format.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_DosToZoned(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter.
lpSource	Points to the packed decimal data in ASCII format. Each left half-byte (X'03') is converted to X'0F' except for the last left half-byte (the sign), which is not converted.
length	Identifies the number of bytes to be converted.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_EBCDICToANSI

Purpose

This function converts a string from EBCDIC to the Windows ANSI code page. The router must be loaded so that this routine can access the ASCII-to-EBCDIC translation table.

If the target string is not large enough to contain the translated string, the translation stops at the end of the target string. If the target string is larger than required, it is blank filled to the end of the string.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned int _far _pascal EHNDT_EBCDICToANSI(
                HWND            hWnd,
                LPSTR           lpsTarget,
                LPSTR           lpsSource,
                unsigned int    wSource,
                LPWORD          lpwTarget );
```

Parameters

hWnd	Identifies the current window of the application.
lpsTarget	Points to the target (translated) string.
lpsSource	Points to the source (EBCDIC) string to convert.
wSource	Identifies the length of the source string in bytes.
lpwTarget	Points to a word variable containing the size of the target buffer. This variable will be updated with the total number of translated characters in the target buffer.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If the router is not loaded, EHNDT_E2A_TABLE_NOT_FOUND (X'FFFC') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_EBCDICToASCII

Purpose

This function converts a string from EBCDIC to ASCII. The router must be loaded so that this routine can access the ASCII-to-EBCDIC translation table.

If the target string is not large enough to contain the translated string, the translation stops at the end of the target string. If the target string is larger than required, it is blank filled to the end of the string.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned int _far _pascal EHNDT_EBCDICToASCII(
                HWND            hWnd,
                LPSTR           lpsTarget,
                LPSTR           lpsSource,
                unsigned int    wSource,
                LPWORD          lpwTarget );
```

Parameters

hWnd	Identifies the current window of the application.
lpsTarget	Points to the target (translated) string.
lpsSource	Points to the source (EBCDIC) string to convert.
wSource	Identifies the length of the source string in bytes.
lpwTarget	Points to a word variable containing the size of the target buffer. This variable will be updated with the total number of translated characters in the target buffer.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If the router is not loaded, EHNDT_E2A_TABLE_NOT_FOUND (X'FFFC') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_EBCDICToEBCDIC

Purpose

This function copies an EBCDIC string pointed to by the lpSource parameter into an area pointed to by the lpTarget parameter. If an EBCDIC character less than X'40' is encountered, translation is stopped and an error code is returned.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned int _far _pascal EHNDT_EBCDICToEBCDIC(
                HWND            hWnd,
                char _far *     lpTarget,
                char _far *     lpSource,
                int             length );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the translated string.
lpSource	Points to the EBCDIC string to convert.
length	Identifies the number of bytes to be translated.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_GetASCIIToEBCDICTable

Purpose

This function returns a pointer to the ASCII-to-EBCDIC translation table of the router.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern LPSTR _far _pascal EHNDT_GetASCIIToEBCDICTable(
                HWND            hWnd );
```

Parameters

hWnd	Identifies the current window of the application.

Return Codes

If successful, the function returns a pointer to the ASCII-to-EBCDIC translation table of the router. If the function is not successful, NULL is returned.

EHNDT_GetEBCDICToASCIITable

Purpose

This function returns a pointer to the EBCDIC-to-ASCII translation table of the router.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern LPSTR _far _pascal EHNDT_GetASCIIToEBCDICTable(
                HWND           hWnd );
```

Parameters

hWnd	Identifies the current window of the application.

Return Codes

If successful, the function returns a pointer to the EBCDIC-to-ASCII translation table of the router. If the function is not successful, NULL is returned.

EHNDT_HexToASCII

Purpose

This function converts each byte of hexadecimal data into 2 ASCII characters.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_HexToASCII(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter multiplied by two.
lpSource	Points to the hexadecimal data to be transformed. The length of this data must be greater than or equal to the value of the length parameter.
length	Identifies the number of source bytes to be transformed.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned.

EHNDT_PackedToASCII

Purpose

This function converts a packed decimal number to ASCII numeric format.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_PackedToASCII(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length,
                int            decimal_position );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the target buffer. If decimal_position is not zero, the output data length is the value of the length parameter x 2 + 1. If the value of the decimal_position is zero, the output data length is the value of the length parameter x 2.
lpSource	Points to the packed decimal data. If the sign half-byte is X'B' or X'D', it is converted to a negative number. Any other sign half-byte is converted to a positive number.
length	Identifies the number of bytes of source data to be converted.
decimal_position	Identifies the position of the decimal point in the lpTarget parameter. This value must be greater than or equal to zero, and must be less than the value of the length parameter multiplied by two.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_PackedToDos

Purpose

This function converts packed decimal data in EBCDIC format to packed decimal data in ASCII format.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_PackedToDos(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter.
lpSource	Points to the packed decimal data in EBCDIC format. If the sign half-byte is X'B' or X'D', it is converted to X'B'. Any other sign half-byte is converted to X'3'.
length	Identifies the number of bytes to be converted.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_PackedToPacked

Purpose

This function copies packed decimal data into a buffer pointed to by the lpTarget parameter.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_PackedToPacked(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter.
lpSource	Points to the packed decimal data to be copied.
length	Identifies the number of bytes to be copied.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_ZonedToDos

Purpose

This function converts zoned decimal data in EBCDIC format to zoned decimal data in ASCII format.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_ZonedToDos(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter.
lpSource	Points to the zoned decimal data in EBCDIC format. The left half-byte must be X'F', and the right half-byte must be between X'0' and X'9'.
length	Identifies the number of bytes to be converted.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_ZonedToASCII

Purpose

This function converts zoned decimal data in EBCDIC format to ASCII numeric format.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_ZonedToASCII(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length,
                int            decimal_position );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the target buffer. If the decimal_position parameter is zero, the output data length is the value of the length parameter multiplied by 2.
lpSource	Points to the EBCDIC zoned decimal data. The sign half-byte (X'0B' or X'0D') is converted to a negative number; everything else is converted to a positive number. The high half-byte in the zoned decimal data must be X'F' except for the last byte which must be X'B', X'0', or X'F' The low half-byte must be 0 through 9.
length	Identifies the number of bytes to be converted.
decimal_position	Identifies the position of the decimal point in the lpTarget parameter. This must be greater than or equal to zero, and must be less than the value of the length parameter.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

EHNDT_ZonedToZoned

Purpose

This function copies zoned decimal data into an area pointed to by the lpTarget parameter.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNDTW.H"
extern unsigned _far _pascal EHNDT_ZonedToZoned(
                HWND           hWnd,
                char _far *    lpTarget,
                char _far *    lpSource,
                int            length );
```

Parameters

hWnd	Identifies the current window of the application.
lpTarget	Points to the target buffer. The size of this buffer must be greater than or equal to the value of the length parameter.
lpSource	Points to the zoned decimal data to be copied. The left half-byte must be X'F', and the right half-byte must be between X'0' and X'9'.
length	Identifies the number of bytes to be copied.

Return Codes

If the function is successful, EHNDT_SUCCESS (X'0000') is returned. If a NULL pointer was passed for the lpTarget or lpSource parameters, EHNDT_NULLPOINTER (X'FFFD') is returned. If invalid data is found during translation, the return code is the location of the first untranslated character plus one.

Chapter 28. Network Redirector Windows Application Program Interface

The network redirector API provides an interface for Windows programmers to the DOS network redirection functions. This API may be used with shared folders, virtual print, and any other network program that supports these functions to assign (redirect) and release (cancel redirection) network devices. A programmer can use this API to write an application to access PC Support drives and printers as well as devices connected to other networks.

For a more detailed discussion on the DOS network redirection functions, refer to the *DOS Technical Reference*.

This chapter describes the individual routines and return codes which make up the network redirector Windows API. The descriptions make reference to predefined constants, data structures, and function prototypes which can be found in the file EHNNETW.H in the PC Support tools folder (QIWSTOOL). The PC Support tools folder also contains a sample program which demonstrates how to use the network redirector Windows API. For more information on the PC Support tools folder and how to use it, see Appendix C, "PC Support Tools Folder."

Throughout this chapter the term network path is used. A **network path** defines a system name and a remote object. For PC Support/400, the network path is defined for network drives as:

```
\\system{\folder}{\folder}{\folder}
```

where `system` is the name of the AS/400 system and `folder` is the name of a folder defined on that system.

The following example describes the shared folder MRI2924 in the shared folder QIWSFL2 on the system SYSTEM1.

```
\\SYSTEM1\QIWSFL2\MRI2924
```

For network printers:

```
\\system{\printer}{{\prtfilelib}\printfile}
```

where `system` is the name of the AS/400 system and `printer` is the name of a printer on the system. `Prtfilelib` is the name of a library where the printer file is located. If none is specified, *USRLIBL is searched. `Printfile` is the name of a printer file on the AS/400 system. If none is specified, QPVPRINT in library QIWS is used.

The following example describes a network printer that prints to printer PRT01 using printer file PRTFILE1 in a library found in the user library list on the system SYSTEM1:

```
\\SYSTEM1\PRT01\\PRTFILE1
```

For network programs other than PC Support/400, the network path is defined in its generic form as:

```
\\computername\{shortname:printdevice}
```

For information on the requirements for developing and running applications that use the PC Support Windows APIs, see "Requirements for Using Windows APIs" on page 19-9. In addition to these requirements, if the network redirector API is used with shared folders, shared folders (STARTFLR) must be loaded. If it is used with virtual print, virtual print (VPRT) must be loaded.

Network Redirector Windows DLL Routines

The following discussions of each network redirector Windows DLL routine describe in detail:

- Purpose
- Procedure declaration
- Parameters
- Return codes

EHNNET_CancelRedirection

Purpose

This function cancels a network redirection for a device.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNNETW.H"
word far pascal EHNNET_CancelRedirection(
                HWND            hWnd,
                LPSTR           lpszDeviceName);
```

Parameters

hWnd	Identifies the current window of the application.
lpszDeviceName	Points to a null-terminated character string that specifies the name of the device for which the redirection is to be cancelled. For a drive device, it must be in the form: drive letter followed by a colon (d:). Otherwise, it is considered a printer device.

Return Codes

For return codes, see "Return Codes for the Network Redirector DLL" on page 28-5.

EHNNET_GetCapability

Purpose

This function returns the functional level of the network redirector Windows DLL.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNNETW.H"
word far pascal EHNNET_GetCapability
                HWND           hWnd);
```

Parameters

hWnd — Identifies the current window of the application.

Return Codes

The return value is the functional level of the network redirector Windows DLL. Possible values are:

0 The network redirector DLL cannot be run in the current environment. A possible reason is that Windows is running in real mode.

1 The network redirector DLL contains support for the following entry points:
- EHNNET_GetCapability
- EHNNET_RedirectDevice
- EHNNET_CancelRedirection
- EHNNET_QueryNetworkPath
- EHNNET_GetRedirectEntry

>1 Reserved for future enhancements of the network redirector DLL.

EHNNET_GetRedirectEntry

Purpose

This function indexes into the current redirection list to determine if an entry exists for the given index. If a redirection entry is found, this routine optionally returns the device type, the device name, and the network name associated with the redirection. To receive all redirections, this entry point should be called multiple times with incrementing values of wIndex.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNNETW"
word far pascal EHNNET_GetRedirectEntry(
                HWND           hWnd,
                WORD           wIndex,
                char far *     lpcDeviceType,
                LPSTR          lpszDeviceName,
                LPSTR          lpszNetworkName);
```

Parameters

hWnd — Identifies the current window of the application.

wIndex — Identifies the index number into the redirection list. To receive the entire list of redirections, this index should have a value of zero the first time this entry point is called. It should then be incremented on successive calls until the entire list has been returned.

lpcDeviceType	Points to a character field which is used to return the device type associated with the redirection. If the pointer is null, the device type is not returned.
lpszDeviceName	Points to a character buffer which is used to return the device name associated with the redirection. This buffer must be at least 128 bytes in length. If the pointer is null, the device name is not returned.
lpszNetworkName	Points to a character buffer which is used to return the network path associated with the redirection. This buffer must be at least 128 bytes in length. If the pointer is null, the network name is not returned.

Return Codes

For return codes, see "Return Codes for the Network Redirector DLL" on page 28-5.

EHNNET_QueryNetworkPath

Purpose

This function queries a device to determine if it is redirected. If the device is redirected, this routine optionally returns the network path associated with the redirection.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNNETW.H"
word far pascal EHNNET_QueryNetworkPath(
                HWND            hWnd,
                LPSTR           lpszDevice,
                LPSTR           lpszNetworkPath);
```

Parameters

hWnd	Identifies the current window of the application
lpszDevice	Points to a null-terminated character string which specifies the name of the device to be queried. For a drive device, it must be in the form: drive letter followed by a colon (d:). Otherwise, it is considered a printer device.
lpszNetworkPath	Points to a buffer in which the network path is to be returned. This buffer must be at least 128 bytes in length. If the pointer is null, the network path is not returned.

Return Codes

For return codes, see "Return Codes for the Network Redirector DLL" on page 28-5.

EHNNET_RedirectDevice

Purpose

This entry point establishes a network redirection for a device.

Procedure Declaration

```
#include <WINDOWS.H>
#include "EHNNETW.H"
word far pascal EHNNET_RedirectDevice
                HWND            hWnd,
                LPSTR           lpszDeviceName,
                LPSTR           lpszNetworkName,
                LPSTR           lpszPassword);
```

Parameters

hWnd	Identifies the current window of the application.
lpszDeviceName	Points to a null-terminated character string which specifies the name of the device to be redirected. For a drive device, it must be in the form: Drive letter followed by a colon (d:) Otherwise, it is considered a printer device.
lpszNetworkName	Points to a null-terminated character string that specifies the name of the system and the path to which the device is to be redirected.
lpszPassword	Points to a null-terminated character string that identifies the password to be used for the redirect request. A null pointer or password with zero length is considered to be no password.

Return Codes

For return codes, see "Return Codes for the Network Redirector DLL."

Return Codes for the Network Redirector DLL

Functions in the network redirector Windows DLL use the following return codes. These are constants defined in EHNNETW.H.

Table 28-1. Return Codes for the Network Redirector API

Return Code	Hex Value	Description
EHNNET_SUCCESS	0000	No error.
EHNNET_INVALIDFUNCTION	0001	Function not valid.
EHNNET_ENVIRONMENTERROR	000A	Network Redirector DLL does not support the current environment. (Windows may be running in real mode.)
EHNNET_INVALIDDEVICE	000F	Invalid device.
EHNNET_NOMOREFILES	0012	No more files.
EHNNET_GENERALFAILURE	001F	General failure.
EHNNET_NETNOTSUPPORTED	0032	Network request not supported.
EHNNET_NORESPONSE	0033	Remote computer not listening.
EHNNET_PATHNOTFOUND	0035	Network path not found.
EHNNET_UNEXPECTNETERROR	003B	Unexpected network error.
EHNNET_NETACCESSDENIED	0041	Network access denied.
EHNNET_NETNAMENOTFOUND	0043	Network name not found.
EHNNET_NETNAMELIMIT	0044	Name too long.
EHNNET_TEMPCANNOTREDIR	0048	Redirect function temporarily suspended.
EHNNET_TOOMANYREDIRECT	0054	Too many redirections.
EHNNET_DUPLICATEREDIR	0055	Duplicate redirection.
EHNNET_INVALIDPASSWORD	0056	Password not valid.
EHNNET_INVALIDPARAMETER	0057	Parameter not valid.
EHNNET_INVALIDLEVEL	007C	DLL is not at a functional level to support this API.

Part 4. Reference Information

This part contains reference information. Appendix A, "Router Problem Analysis" lists some programming methods and experiences that can help avoid some of the more common problems found when trying to debug communications programs.

Appendix B, "Table of Router Conversation States" provides the possible conversation state changes that can occur when a transaction program issues a conversation verb. An explanation of the abbreviations and symbols for conversation states is followed by a table showing the correlation of verbs and function calls with conversation.

Appendix C, "PC Support Tools Folder" describes the PC Support tools folder and how to use it. The tools folder contains packages of information, include files, and sample programs for most of the PC Support APIs.

Appendix A. Router Problem Analysis

Debugging communications programs can be very frustrating because there are two ends of the conversation that can go wrong. Add to that the lack of debugging tools in DOS and you can be looking at some very long debugging sessions.

Using a library of subroutines for communications, such as those in the sample programs, can help you. This section lists some programming methods and experiences to help you avoid some of the more common problems.

Probably the most common problem that occurs is that the verb structures passed to the router are not byte-aligned. Several compilers attempt to improve performance by aligning on word boundaries. If your compiler has an option to force alignment to be byte rather than packed, be sure you use this option.

Another common problem is not understanding the library list used by APPC programs. It is not normally the same one you get when you sign on to a 5250 display session. The library list is set in the default job description name in your user profile. Make sure the library and necessary AS/400 objects are in libraries in this library list.

Return Codes

You should first check the router return codes when your program does not work. Your program should have a method of displaying all unexpected return codes. See "Program Examples" on page 4-60 for a method of checking return codes.

AS/400 Messages and Job Logs

Often when something does not work correctly, you can check the AS/400 QSYSOPR message queue for messages relating to your personal computer. The job log also has messages that contain information about failing APPC conversations. ALLOCATE failures are reported with CPF1269 messages and include a reason code. These reason codes should always be checked when you cannot get a conversation allocated.

Router Status

Make sure the router is still connected to your partner system after a failure. You can use the STARTRTR command and specify /D as a command line option to determine the status of the router connection.

System Stopped or Looping

If your personal computer seems to be stopped or in a loop while running your application, check the status at the AS/400 system. If you issued a RECEIVE_AND_WAIT verb and the AS/400 program is not responding, your personal computer will appear to be stopped.

Your personal computer may possibly appear stopped when it is sending large volumes of data, when the partner AS/400 system has not issued a receive opera-

tion yet. Eventually, all buffers are filled and the program stops until more buffers are available.

To determine if a personal computer program is looping or waiting for host system data, restart the personal computer. If you can restart the personal computer with a Ctrl-Alt-Del key sequence, then the personal computer is probably looping or waiting on the host system. If you have to restart the personal computer by turning it off and then back on, then probably another personal computer program is causing the problem.

One common cause of a personal computer stopped or looping condition is that the buffer passed to the router on the ALLOCATE request is not in conventional memory. You must make sure that the buffer is in conventional memory and that your program never writes to this area after the ALLOCATE request. Another problem may be that the DS and DX register do not point at 8 bytes past the start of the ALLOCATE verb.

Traces

Tracing can be a very effective method for determining the status of both sides of the conversation when a failure occurs. The AS/400 system has two trace functions that can be used to help your debugging, Trace Intersystems Communications Functions (TRCICF) and line traces.

TRCICF is an AS/400 command that traces an I/O request from your program to the ICF interface on the AS/400 system. It shows return codes, data received and sent, length of data, and control information. To use TRCICF, you must place the command in a CL program and use that program to call your actual APPC program. Your PC program then has to name the CL program as the target program on its ALLOCATE verb.

Line traces are also available on the AS/400 system. Line traces show the actual data that is sent and received through an AS/400 communications adapter. Line tracing is only available on LAN and SDLC connections.

Appendix B. Table of Router Conversation States

The following table shows the possible conversation state changes that can occur when a transaction program issues a conversation verb. The columns show the states and the rows show the verbs. A verb is listed more than once if that verb causes a state change.

The three causes of state changes are:

- Verbs the local program issues
- Verbs the remote program issues
- Network errors

A state change caused by the remote program or a network error is indicated by a return code with error information. For the RECEIVE_AND_WAIT or RECEIVE_IMMEDIATE verbs, the router can indicate state changes on the data_received parameters.

Some errors, such as PARAMETER_CHECK or STATE_CHECK, do not cause a state change and are not listed separately for each verb.

Refer to the abbreviations on the following page to translate the abbreviations and symbols that follow each verb. Abbreviations and symbols are used for parameters, return codes, and the what_received values.

Each listed verb applies to basic and mapped conversations.

For the purpose of this state table, Confirm state has been split into three states: Confirm, Confirm-Send, or Confirm-Deallocate, depending on whether the Confirm state was entered by means of the corresponding status_received parameter.

Abbreviations and Symbols for Conversation States

The following sections explain and provide examples of the various abbreviations and symbols that are used in the Table B-1.

Parameter Abbreviations (...)

The abbreviations for the verb parameter values in the Conversation States table are enclosed by parentheses. The following abbreviations are used:

Table B-1. Parameter Abbreviations (...)

Abbreviation	Meaning
A	type(ABEND_PROG)
B	type(NONE)
C	type(SYNC_LEVEL) with synchronization level CONFIRM
D	type(DEALLOCATE_ABEND) type(DEALLOCATE_FLUSH) type(DEALLOCATE_SYNC_LEVEL)
F	type(FLUSH)

Return Code Abbreviations [...]

The abbreviations for the return codes in the Conversation States table are enclosed by brackets. The following abbreviations are used:

Table B-2. Return Code Abbreviations [...]

Abbreviation	Corresponding Return Code
ae	ALLOCATION_FAILURE_NO_RETRY ALLOCATION_FAILURE_RETRY CONV_TYPE_MISMATCH SYNC_LEVEL_NOT_SUPPORTED TP_NAME_NOT_RECOGNIZED TRANS_PGM_NOT_AVAIL_NO_RETRY TRANS_PGM_NOT_AVAIL_RETRY
da	DEALLOC_ABEND DEALLOC_ABEND_PROG DEALLOC_ABEND_SVC DEALLOC_ABEND_TIMER
dn	DEALLOC_NORMAL
en	ERROR_NO_TRUNC or SVC_ERROR_NO_TRUNC
ep	PROG_ERROR_PURGING or SVC_ERROR_PURGING
et	PROG_ERROR_TRUNC or SVC_ERROR_TRUNC
ok	OK (successful completion of command)
pc	PARAMETER_CHECK or STATE_CHECK
ps	Any common product-specific return code such as APPC__BUSY
rf	CONV_FAILURE_RETRY or CONV_FAILURE_NO_RETRY
un	UNSUCCESSFUL

What_Received Abbreviations {...}

The abbreviations for the what_received values in the Conversation States table are enclosed by braces. The following abbreviations are used:

Table B-3. Abbreviations for What_Received Values { , }

Abbreviation	Meaning
dr	The meaning is one of the following: DATA DATA_COMPLETE DATA_INCOMPLETE
cd	CONFIRM_DEALLOCATE
cs	CONFIRM_SEND
co	CONFIRM
no	NONE
se	SEND

Symbols Used in the State Table

The following symbols are used in the Conversation States table:

Table B-4. Symbols Used in the State Table

Symbol	Meaning
/	Verb cannot be issued or the indicated result cannot occur in this state
—	Remains in current state
1-6	Number of next conversation state

Conversation States Table

Table B-5 (Page 1 of 3). Correlation of Verbs and Function Calls with Conversation

Verb or DOS Call:	Conversation States:					
	Reset	Send	Receive	Confirm	Confirm Send	Confirm Deal-locate
	1	2	3	4	5	6
ALLOCATE [ok]	2	/	/	/	/	/
ALLOCATE [ae]	—	/	/	/	/	/
ALLOCATE [pc]	—	/	/	/	/	/
ALLOCATE [ps]	—	/	/	/	/	/
ALLOCATE [un]	—	/	/	/	/	/
CONFIRM [ok]	/	—	/	/	/	/
CONFIRM [ae]	/	1	/	/	/	/
CONFIRM [da]	/	1	/	/	/	/
CONFIRM [ep]	/	3	/	/	/	/
CONFIRM [rf]	/	1	/	/	/	/
CONFIRM [pc]	/	—	/	/	/	/
CONFIRM [ps]	/	—	/	/	/	/
CONFIRMED [ok]	/	/	/	3	2	1
CONFIRMED [pc]	/	/	/	—	—	—
CONFIRMED [ps]	/	/	/	—	—	—
DEALLOCATE (F) [ok]	/	1	/	/	/	/
DEALLOCATE (F) [pc]	/	—	/	/	/	/
DEALLOCATE (F) [ps]	/	—	/	/	/	/
DEALLOCATE (C) [ok]	/	1	/	/	/	/
DEALLOCATE (C) [ae]	/	1	/	/	/	/
DEALLOCATE (C) [da]	/	1	/	/	/	/
DEALLOCATE (C) [ep]	/	3	/	/	/	/
DEALLOCATE (C) [rf]	/	1	/	/	/	/
DEALLOCATE (C) [pc]	/	—	/	/	/	/
DEALLOCATE (C) [ps]	/	—	/	/	/	/
DEALLOCATE (A) [ok]	/	1	1	1	1	1
DEALLOCATE (A) [pc]	/	—	—	—	—	—
DEALLOCATE (A) [ps]	/	—	—	—	—	—
FLUSH [ok]	/	—	/	/	/	/
FLUSH [pc]	/	—	/	/	/	/
FLUSH [ps]	/	—	/	/	/	/
GET_ATTRIBUTES	/	—	—	—	—	—

Table B-5 (Page 2 of 3). Correlation of Verbs and Function Calls with Conversation

Verb or DOS Call:	Conversation States:					
	Reset	Send	Receive	Confirm	Confirm Send	Confirm Deal-locate
	1	2	3	4	5	6
PREPARE_TO_RECEIVE (F) [ok]	/	3	/	/	/	/
PREPARE_TO_RECEIVE (F) [pc]	/	—	/	/	/	/
PREPARE_TO_RECEIVE (F) [ps]	/	—	/	/	/	/
PREPARE_TO_RECEIVE (C) [ok]	/	3	/	/	/	/
PREPARE_TO_RECEIVE (C) [ae]	/	1	/	/	/	/
PREPARE_TO_RECEIVE (C) [da]	/	1	/	/	/	/
PREPARE_TO_RECEIVE (C) [ep]	/	3	/	/	/	/
PREPARE_TO_RECEIVE (C) [rf]	/	1	/	/	/	/
PREPARE_TO_RECEIVE (C) [pc]	/	—	/	/	/	/
PREPARE_TO_RECEIVE (C) [ps]	/	—	/	/	/	/
RECEIVE_AND_WAIT [ok] {dr}	/	3	—	/	/	/
RECEIVE_AND_WAIT [ok] {se}	/	2	2	/	/	/
RECEIVE_AND_WAIT [ok] {co}	/	4	4	/	/	/
RECEIVE_AND_WAIT [ok] {cs}	/	5	5	/	/	/
RECEIVE_AND_WAIT [ok] {cd}	/	6	6	/	/	/
RECEIVE_AND_WAIT [ae]	/	1	1	/	/	/
RECEIVE_AND_WAIT [da]	/	1	1	/	/	/
RECEIVE_AND_WAIT [dn]	/	1	1	/	/	/
RECEIVE_AND_WAIT [en]	/	3	—	/	/	/
RECEIVE_AND_WAIT [ep]	/	3	—	/	/	/
RECEIVE_AND_WAIT [et]	/	/	—	/	/	/
RECEIVE_AND_WAIT [rf]	/	1	1	/	/	/
RECEIVE_AND_WAIT [pc]	/	—	—	/	/	/
RECEIVE_AND_WAIT [ps]	/	—	—	/	/	/
RECEIVE_IMMEDIATE [ok] {dr}	/	/	—	/	/	/
RECEIVE_IMMEDIATE [ok] {se}	/	/	2	/	/	/
RECEIVE_IMMEDIATE [ok] {co}	/	/	4	/	/	/
RECEIVE_IMMEDIATE [ok] {cs}	/	/	5	/	/	/
RECEIVE_IMMEDIATE [ok] {cd}	/	/	6	/	/	/
RECEIVE_IMMEDIATE [ae]	/	/	1	/	/	/
RECEIVE_IMMEDIATE [da]	/	/	1	/	/	/
RECEIVE_IMMEDIATE [dn]	/	/	1	/	/	/
RECEIVE_IMMEDIATE [en]	/	/	—	/	/	/
RECEIVE_IMMEDIATE [ep]	/	/	—	/	/	/
RECEIVE_IMMEDIATE [et]	/	/	—	/	/	/
RECEIVE_IMMEDIATE [rf]	/	/	1	/	/	/
RECEIVE_IMMEDIATE [pc]	/	/	—	/	/	/
RECEIVE_IMMEDIATE [ps]	/	/	—	/	/	/
RECEIVE_IMMEDIATE [un]	/	/	—	/	/	/
REQUEST_TO_SEND [ok]	/	/	—	—	/	/
REQUEST_TO_SEND [pc]	/	/	—	—	/	/
REQUEST_TO_SEND [ps]	/	/	—	—	/	/
SEND_DATA (B) [ok]	/	—	/	/	/	/
SEND_DATA (F) [ok]	/	—	/	/	/	/
SEND_DATA (C) [ok]	/	—	/	/	/	/
SEND_DATA (P) [ok]	/	3	/	/	/	/
SEND_DATA (D) [ok]	/	1	/	/	/	/

Table B-5 (Page 3 of 3). Correlation of Verbs and Function Calls with Conversation

Verb or DOS Call:	Conversation States:					
	Reset	**Send**	**Receive**	**Confirm**	**Confirm Send**	**Confirm Deal-locate**
	1	**2**	**3**	**4**	**5**	**6**
SEND_DATA [ae]	/	1	/	/	/	/
SEND_DATA [da]	/	1	/	/	/	/
SEND_DATA [ep]	/	3	/	/	/	/
SEND_DATA [pc]	/	—	/	/	/	/
SEND_DATA [rf]	/	1	/	/	/	/
SEND_DATA [ps]	/	—	/	/	/	/
SEND_ERROR [ok]	/	—	2	2	2	2
SEND_ERROR [ae]	/	1	/	/	/	/
SEND_ERROR [da]	/	1	/	/	/	/
SEND_ERROR [dn]	/	/	1	/	/	/
SEND_ERROR [ep]	/	3	/	/	/	/
SEND_ERROR [pc]	/	—	—	—	—	—
SEND_ERROR [rf]	/	1	1	1	1	1
SEND_ERROR [ps]	/	—	—	—	—	—

Appendix C. PC Support Tools Folder

The PC Support tools folder is available on the AS/400 system after the PC Support/400 product is installed on the system. The tools folder contains packages of information, include files, and sample programs for most of the PC Support/400 application program interfaces (APIs).

Using the Tools Folder

To use the tools folder, do the following:

1. Start a PC Support connection, and make sure the I: drive is assigned as a system drive.
2. Type `I:\QIWSFL2\IWSTOOL`. If you are using a double-byte character set, type `I:\QIWSFL2D\IWSTOOLD`. Press the Enter key.

   Note: This command works only in DOS mode. If you are using the OS/2 program, change to DOS mode before entering the command.
3. To display the abstract for a package, use the mouse or cursor to select the package and press the Enter key.
4. To use the displayed package, page down to the end of the abstract, type the name of a valid (existing) directory or folder in the input field, and press the Enter key. The files listed in the abstract for the selected package are copied to the specified directory or folder.
5. When you have finished selecting packages, press the Esc key to return to the PC prompt.
6. Change to the directory or folder. For information on how to use a package, you can browse or print any files with an extension of .ABS, .DOC, or TXT. Some of the sample programs contain additional information at the beginning of the source file.

Bibliography

Some of the manuals below are listed with their full title and base order number. When these manuals are referred to in this manual, a shortened version of the title is used.

Publications of IBM:

- *Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084

  Short title: TP Reference for LU 6.2.

  Contains information needed for writing application programs conforming to logical unit (LU) type 6.2.

- *Communications: Operating System/400* Communications Configuration Reference*, SC41-0001

  Short title: *OS/400* Communications Configuration Reference*.

  Contains general configuration information, including descriptions of network interface, line controller, device, mode, and class-of-service descriptions, configuration lists and connection lists.

- *Communications: Advanced Peer-to-Peer Networking Guide*, SC41-8188

  Short title: *APPN Guide*.

  Provides information about the concepts of AS/400 advanced peer-to-peer networking (APPN) and about planning APPN networks.

- *Communications: Advanced Program-to-Program Communications Programmer's Guide*, SC41-8189

  Short title: *APPC Programmer's Guide*.

  Describes the advanced program-to-program communications (APPC) support for the AS/400 system and provides the information necessary for developing communications application programs.

- *Communications: Intersystem Communications Function Programmer's Guide*, SC41-9590

  Short title: *ICF Programmer's Guide*.

  Provides the information needed to write application programs that use AS/400 communications and the ICF file. It also contains examples of communications programs and describes return codes.

- *Systems Application Architecture* Structured Query Language/400 Reference*, SC41-9608

  Short title: *SQL/400* Reference*.

  Contains reference information for the tasks of system administration, database administration, application programming, and operation using IBM Structured Query Language/400 (SQL/400*).

- *Systems Application Architecture* Structured Query Language/400 Programmer's Guide*, SC41-9609

  Short title: *SQL/400* Programmer's Guide*.

  Contains information for programmers and database managers on how to use the IBM Structured Query Language/400 (SQL/400), how to access data in a database, and how to prepare, run, and test an application program containing SQL statements.

- *Publications Guide*, GC41-9678

  Short title: *Publications Guide*.

  Lists manuals in the AS/400 library and lists tasks that are described in the manuals.

- *OS/2 Extended Edition Version 1.2 APPC Programmer's Reference*.

  Short title: OS/2 EE V1.2 APPC Programmer's Reference.

  Contains information needed for writing application programs for advanced program-to-program communications (APPC) using the OS/2 operating system.

Publications of other companies:

- *Microsoft Windows Software Development Kit Reference*

  Provides a detailed description of the Windows dynamic data exchange (DDE) protocol.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Customer Satisfaction Feedback

Application System/400
PC Support/400:
Application Program Interface Reference
Version 2

Publication No. SC41-8254-02

Overall, how would you rate this manual?

	Very Satisfied	Satisfied	Dissatis-fied	Very Dissatis-fied
Overall satisfaction				

How satisfied are you that the information in this manual is:

Accurate				
Complete				
Easy to find				
Easy to understand				
Well organized				
Applicable to your tasks				
THANK YOU!				

Please tell us how we can improve this manual:

_____

_____

_____

_____

_____

May we contact you to discuss your responses? __ Yes __ No

Phone: (____) _____ Fax: (____) _____

To return this form:

- Mail it
- Fax it
  United States and Canada: **800+937-3430**
  Other countries: **(+1)+507+253-5192**
- Hand it to your IBM representative.

Note that IBM may use or distribute the responses to this form without obligation.

Name _____ Address _____

Company or Organization _____ _____

Phone No. _____ _____

IBM®

Cu
Ad

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DEPT 245
IBM CORPORATION
3605 HWY 52 N
ROCHESTER MN 55901-7899

Fold and Tape **Please do not staple** Fold and Tape

Cu
Ad

Customer Satisfaction Feedback

Application System/400
PC Support/400:
Application Program Interface Reference
Version 2

Publication No. SC41-8254-02

Overall, how would you rate this manual?

	Very Satisfied	Satisfied	Dissatis-fied	Very Dissatis-fied
Overall satisfaction				

How satisfied are you that the information in this manual is:

Accurate				
Complete				
Easy to find				
Easy to understand				
Well organized				
Applicable to your tasks				
THANK YOU!				

Please tell us how we can improve this manual:

May we contact you to discuss your responses? __ Yes __ No

Phone: (___) _____ Fax: (___) _____

To return this form:

- Mail it
- Fax it
  United States and Canada: **800+937-3430**
  Other countries: **(+1)+507+253-5192**
- Hand it to your IBM representative.

Note that IBM may use or distribute the responses to this form without obligation.

Name _____ Address _____

Company or Organization _____ _____

Phone No. _____ _____

IBM®

Cut
Alo

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DEPT 245
IBM CORPORATION
3605 HWY 52 N
ROCHESTER MN 55901-7899

Fold and Tape | **Please do not staple** | Fold and Tape

Cut
Alo

Program Number: 5738-PC1

Printed in Denmark by
Aalborg Stiftsbogtrykkeri A/S

SC41-8254-02