GXS EDI Services

Expedite for Windows Software Development Kit Programming Guide

Version 6 Release 2

GC34-3285-02

Fifth Edition (November 2005)

This edition replaces the Version 6.1 edition.

Contents

To the reader

The term *network* is used in this book to refer to the worldwide communications network infrastructure provided by *AT&T Global Network Services.*

What this book covers

Expedite for Windows™ is a communications component of EDI Services, formerly IBM EDI Services. Expedite for Windows enables you to transmit data files and messages to trading partners through *Information Exchange,* the mailbox component of EDI Services.

This book explains how to program to the Expedite for Windows C-language application programming interface (API) and details other required information necessary to use Expedite for Windows for your company's applications.

This book also explains how to use the IBM-provided Java™ programming application to enable you to use Java to write programs for Expedite for Windows.

Who should read this book

This book is intended as a guide for programmers who want to use the Expedite for Windows C-language interface and graphical user interface. It is assumed that you understand C programming and have experience writing C applications. To write an interface to Expedite for Windows using the C-language interface, you should be familiar with dynamic link libraries (DLLs) under Windows and have a clear understanding of arrays, pointers, and pointers-to-pointers. It is also assumed that you are familiar with Java programming if you intend to use the provided Java sample programs.

How this book is organized

The book has the following chapters.

- Chapter 1, "Introducing Expedite for Windows," provides basic information on Information Exchange, Expedite for Windows components and protocols, and hardware and software requirements.
- Chapter 2, "Getting Started," provides sample information and explains the sample code provided in the product.
- Chapter 3, "Using the Expedite for Windows graphical user interface," provides information for using the project and address book functions with your trading partners.
- Chapter 4, "Programming to the C-language interface," provides information for creating programs with the C-language interface to use Expedite for Windows from your applications.
- Chapter 5, "Programming to the Java interface," provides information for creating programs with the Java interface to use Expedite for Windows from your applications.
- Chapter 6, "Sending and receiving files with Expedite for Windows," provides information and examples for using send and receive functions in Expedite for Windows and processing procedures for recovering after errors.
- Chapter 7, "Sending and receiving EDI data with Expedite for Windows," describes how you can use Expedite to send and receive data formatted for electronic data interchange (EDI).
- Appendix A, "Code examples," describes several scenarios that you may encounter in your work with your company's applications. Review each scenario to see if you can use the coding example.

This book also contains a notices section, a glossary, and an index.

Type conventions

The following type conventions are used in this book:

- Programming examples are in a monospaced font.
- Field names are in mixed case.
- Function names are in bold mixed case.
- Commands are in capital letters.
- Anything you are to type is in bold.
- Glossary words are in italics the first time they are used in this book.

Related books

You may find it helpful to refer to the following publications when performing the tasks described in this book:

- *Expedite for Windows Software Development Kit Programming Reference*, GC34-3284 (available on product CD-ROM)
- *Expedite for Windows User's Guide*, GC34-2341 (available on product CD-ROM)
- *Information Exchange Mailbox Command Reference*, GC34-2260
- *Information Exchange Messages and Formats*, GC34-2324
- *Information Exchange Message Charges Reference*, GX66-0653

These books are also available on the library page of the EDI Services Web site at **http://www.gxsolc.com/edi_bes.html.**

Introducing Expedite for Windows

Expedite for Windows provides a complete messaging management system that interfaces with the *Information Exchange* product from your environment. Information Exchange, the mailbox component of EDI Services, formerly IBM EDI Services, enables you and your trading partners to exchange messages and files, such as *electronic data interchange (EDI)* files.

With Expedite for Windows, you can use the *graphical user interface (GUI)* to easily complete tasks, such as setting up address books or handling problems with Information Exchange sessions. However, if you want direct control over a session between Expedite for Windows and Information Exchange, you can use the C-language interface or the Java interface.

The following sections provide information about the Expedite for Windows and Information Exchange environments.

Understanding Information Exchange

Information Exchange provides a means of sending, storing, and retrieving information electronically, and enables users on dissimilar computer systems to communicate with one another. By establishing a computer-to-computer communication network between different locations, Information Exchange can speed and simplify the delivery of files, EDI envelopes, and other data.

Information Exchange is an alternative to direct computer-to-computer communication. You can send files to an Information Exchange mailbox and retrieve waiting files from the mailbox.

Through the network, Information Exchange links geographically scattered locations of a single company or different companies. A manufacturing company in one geography, for example, can use Information Exchange to communicate with its suppliers or distributors in various other geographies.

To connect to Information Exchange, you need two corresponding sets of account IDs, user IDs, and passwords:

- The first set allows you to log on to the network. The individual fields in this set are the network account ID, user ID, and password.

  This information is configured and stored in the AT&T Global Network Dialer.

- The second set allows you to log on to Information Exchange. The individual fields in this set are the Information Exchange account ID, user ID, and password.

  This information is configured and stored in Expedite for Windows.

Understanding the Expedite for Windows model

The Expedite for Windows graphical user interface is a set of interrelated elements, referred to as *stations*, that perform functions similar to those of a shipping and receiving center, such as:

- Preparing a shipping order
- Specifying the sender and receiver
- Choosing whether a shipment includes a single item or groups of items
- Choosing how and when to send a shipment
- Tracking a shipment

The graphical user interface uses common, easy-to-recognize graphical controls and objects. You do not have to learn the system's underlying functions in order to use Expedite for Windows.

You will find more information about using the GUI in Chapter 3, "Using the Expedite for Windows graphical user interface."

Understanding the Expedite for Windows components

The following sections provide a brief overview of the main components of Expedite for Windows, which are:

- Request Manager
- Protocol handler
- Graphical user interface
- C-language interface
- Java interface

You can access Information Exchange as shown in Figure 1. The figure shows how Expedite for Windows, installed on your system with your user applications, communicates with the network and Information Exchange through a modem or leased line.

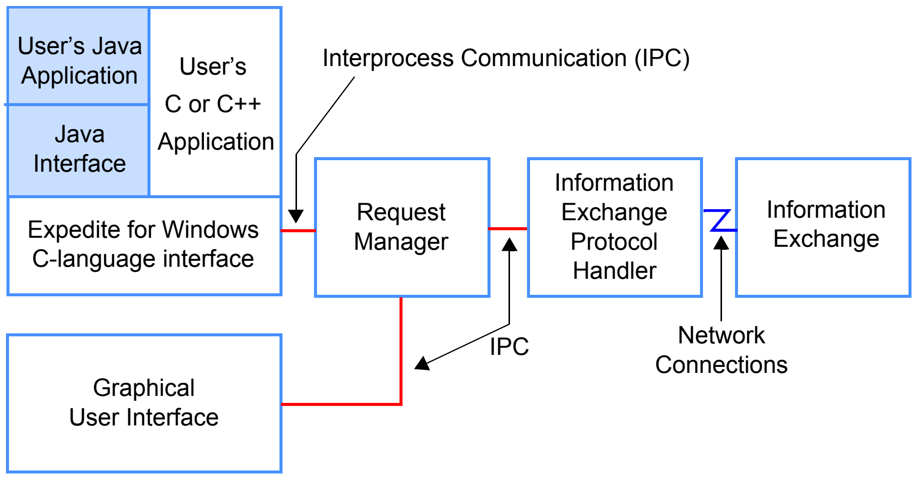

User's Java Application
User's C or C++ Application
Java Interface
Interprocess Communication (IPC)
Expedite for Windows C-language interface
Request Manager
Information Exchange Protocol Handler
Information Exchange
IPC
Network Connections
Graphical User Interface

Figure 1. Expedite for Windows components

Request Manager

Request Manager is the main component of Expedite for Windows. It manages the databases containing information on Information Exchange sessions, data sent and received, and the profiles (application, user, and trading). Request Manager manages your sessions and communications between the Expedite for Windows interfaces and Information Exchange. You can use any of several interfaces (one at a time) to Request Manager, and your selection has no affect on the operation of Request Manager.

Request Manager does not handle user data; it only manages information about databases (where user data is stored) and sessions (where and when user data was sent). However, the protocol handler, as described on page 5, does access user data for sending, receiving, or special formatting.

Request Manager communicates with the interfaces and the protocol handler over *Interprocess Communications (IPC)*, using a proprietary platform and implementation-independent language. To write an interface to Request Manager, you must use the C-language interface.

Request Manager handles requests such as:

- Start an Information Exchange session to send or receive a file
- Add an entry to an address book
- Update an Expedite for Windows configuration file

For some requests that involve a protocol handler and a session with Information Exchange, Request Manager maintains a record of the request and the results of the request. For other requests, such as creating an address book and adding an address, Request Manager takes action immediately and does not keep a record.

Request Manager supports *projects* that allow multiple users or applications (one at a time) to access a single copy of Expedite for Windows. A project identifies an information set that includes address book and mailbox databases, as well as information about the application or user that owns the project.

Protocol handler

The Expedite for Windows *protocol handler* gets commands (requests) from Request Manager through IPC and translates them to the Information Exchange protocol; unlike a similar function in Expedite Base for Windows that gets commands from free-format syntax files. The protocol handler accepts responses from Information Exchange, translates them to the Request Manager protocol, and delivers them over IPC, instead of writing them to an output file as does Expedite Base for Windows.

Your application does not need to start the protocol handler, because applications interface with Request Manager through the C-language interface and do not communicate directly with the protocol handler.

The protocol handler manages checkpoint-level restart with Information Exchange through its control files that are stored in the CONTROL subdirectory under the install directory.

Although the protocol handler manages session restart, Request Manager is responsible for managing Information Exchange account IDs and user IDs. If a session is in restart state, Request Manager does not allow another dropoff box with the same account ID and user ID to be processed. Information about whether an account ID and user ID is disabled or enabled is stored in the project database. An application can use the **ExpListIeLogon** function of the C-language interface to see if any of the logon IDs are disabled. For more information on this function, see the *Software Development Kit Programming Reference.*

With the protocol handler component separate from Request Manager, Expedite for Windows allows data transfer to run in the background freeing users to perform other tasks.

Graphical user interface

The Expedite for Windows graphical user interface (GUI) is designed to run on Windows 95, Windows 98, or Windows NT. The GUI uses Request Manager services in the same way as does the C-language interface.

You can use the GUI to easily resolve Information Exchange session problems. Without using your application software, you can perform tasks in the GUI such as:

- Updating your address book
- Creating trading profiles
- Setting up dropoff boxes

C-language interface

The C-language interface has a consistent naming convention for functions and structures. This makes it easier for you to find information about the interface and to remember the names of functions and structures while programming. All the functions work in a similar manner, whether you use them to add an object to a Request Manager database or implement a function of Information Exchange, such as send or receive.

The Expedite for Windows C-language interface works with one or more *dynamic link libraries (DLLs)* to link into your application. It contains structures that define parameter values to pass to Request Manager and to process responses from Request Manager.

You can use the C-language interface to set up a batch session, handle an interactive session with Information Exchange, or work with Request Manager services for:

- Opening and closing databases
- Adding, replacing, or deleting items in a database
- Creating or listing sets in a database
- Adding, deleting, or listing items in a set
- Processing a dropoff box
- Managing an interactive session with Information Exchange

For more information, see Chapter 4, “Programming to the C-language interface.”

Java interface

The Java interface gives Java programmers the ability to write Java programs to control Expedite for Windows through the existing C-language interface. Java programs can be written and compiled with either the Java Development Kit™ (JDK) by Sun Microsystems or with VisualAge® for Java; Expedite for Windows includes sample programs for each.

For more information, see Chapter 5, “Programming to the Java interface.”

Defining Expedite for Windows objects

Expedite for Windows *objects* are provided in the appropriate format for either the GUI or the C-language interface. These objects correspond to the electronic shipping and receiving metaphor, as follows:

- **Project**

  A *project* allows multiple users or applications (one at a time) to access the same copy of Expedite for Windows. A project represents one user or application, or one user's set of related tasks in Expedite for Windows. Each project has a name, creator, and description. Each project has its own address book database, mailbox database, and project configuration. Projects are designed to be flexible enough to meet your needs for managing multiple users or applications on a single installation.

- **Address book**

  The *address book* allows you to store trading partner information commonly found in an address book, such as names, addresses, phone numbers, and fax numbers. You can set up a nickname for each of your trading partners and relate this nickname to an Information Exchange address or other address book information.

- **Order**

  An *order* represents an Expedite command in Expedite for Windows.

- **Order shelf**

  The *order shelf* automatically stores all the orders you create until you delete them. The order shelf set is provided with Expedite for Windows and includes no orders initially.

- **Dropoff box**

  A *dropoff box* represents a session with Information Exchange. Once orders are created, you can place them in a dropoff box for processing. You can select a copy of an order from the order shelf to copy into a dropoff box at any time. The dropoff box object is similar to the input message file in Expedite Base for Windows.

- **Order receipt**

  An *order receipt* shows the results of a processed order. This corresponds to a record in the output file in Expedite Base for Windows.

- **Receipt shelf**

  The *receipt shelf* is the set, or composite, of all receipts in the mailbox database. The receipt shelf set is provided with Expedite for Windows and includes no receipts initially.

- **Session receipt**

  A *session receipt* provides information about the Information Exchange session. It is similar to the session start and session end records in Expedite Base for Windows. The session receipt includes a set of order receipts.

- **Trading profile**

  A *trading profile* provides a set of default information associated with a receiver to be used to configure send orders. You can create an order with a special configuration that overrides the receiver's trading profile.

Defining Expedite for Windows databases

The Expedite for Windows databases are organized as follows:

- **Address book**

  The *address book database* contains addresses, distribution lists, and trading profiles. An address or distribution list can have only one trading profile associated with it. A trading profile can have many addresses or distribution lists associated with it.

- **Project**

  The *project database* has records that contain information about the current projects defined to Expedite for Windows. The project database contains the network account ID, user ID, and password.

- **Mailbox**

  The *mailbox database* has records that represent orders and receipts for processed orders.

Defining hardware and software requirements

Expedite for Windows is intended to run on any PC or workstation hardware that supports Windows 95, Windows 98, or Windows NT 4.0.

The minimum required configuration is a 120 Mhz Pentium workstation with 16 MB of memory. The required disk space is 29 MB without the books, or 37 MB with the books installed. The Software Development Kit requires 2 MB, for a total disk space of 39 MB.

Supported compiler

The sample *makefiles* provided are for use with the Microsoft Visual C++ 6.0 compiler. Inside the makefiles are the definitions for the type of compiler option flags and linker option flags used to compile the sample programs. You can use these definitions and compiler options as guidelines for creating makefiles to compile your application.

If you are using a compiler other than Microsoft, refer to your compiler documentation on how to link a DLL to your application.

Chapter 2

Getting Started

Expedite for Windows includes a graphical user interface (GUI) that you can use for most of the functions of the application. However, if this GUI does not meet the needs of your users, you may want to write your own interface.

Using the Expedite for Windows GUI

You can use the Expedite for Windows GUI to view your application's project configuration and to do problem determination if necessary. If you install the GUI with your application, GXS Community Support can provide assistance to your users if needed. You can enable your users to do problem determination with the GUI; however, by using project passwords, you can prevent them from changing your project configuration. For more information about the GUI, see Chapter 3, "Using the Expedite for Windows graphical user interface."

Writing your own interface

You can write your own interface to Expedite for Windows if you want to:

- Integrate the functions of Information Exchange into your application and present your own interface to your customers
- Enable only a small number of functions, such as sending and receiving files, for your customers
- Have your application control the session with Information Exchange
- Write an e-mail application

Distributing your project to users

Expedite for Windows is designed to assist you in distributing your project to users. You can do all, some, or none of the configuration in your application, using the GUI to reduce the amount of code you must include in your application.

Creating and exporting your project

To create an application and distribute it to your users, do the following:

1. Install Expedite for Windows on your system.
2. Compile and test the sample applications as described in “Building the sample program” on page 14.
3. Build your application using the C-language interface DLL.
4. Export (distribute) your project configuration to a file using the Expedite for Windows GUI. From the main menu, select “File,” then select “Distribute project.”

Installing the project

Once you have exported your project, your users can then install the project by doing the following:

1. Install Expedite for Windows and your application on their systems.
2. Install your project using the Expedite for Windows Toolbox.

Configuring the project

Your project can be configured either by your application, or by your users through the Expedite for Windows GUI as follows:

1. If dial connectivity will be used, configure the AT&T Global Network Dialer.
2. Configure the Expedite for Windows communications profile.
3. Add the mailbox ID information (Information Exchange account ID, user ID, and password) for Expedite.
4. Assign the new mailbox ID to any dropoff boxes defined in the project.

Programming experience requirement

C programming experience is required to fully utilize the information provided in this book. To write an interface to Expedite for Windows using the C-language interface, you should be familiar with using DLLs with Windows. Some of the Expedite for Windows calls are straight-forward; however, other calls, such as listing multiple types of records from the database, require an understanding of arrays of structures, pointers, and pointers-to-pointers in the C programming language.

Before using the **expc32.dll** file, you need a basic knowledge of DLLs and how they work. Samples provided with the product are a good starting point to show how to compile and link with a DLL. These are located in the SDK\SAMPLE\CLANG directory under the install directory.

You should also be familiar with Java if you want to use the Java interface. For more information, see Chapter 5, “Programming to the Java interface.”

Understanding the C-language interface files

The C-language interface includes the files you need to compile your application in the SDK\INCLUDE and SDK\BIN directories under the Expedite for Windows install directory. These are:

- The header file **SDK\INCLUDE\expc32.h**
- The link library **SDK\BIN\expc32m.lib**
- The dynamic link library **SDK\BIN\expc32.dll**

expc32.h header file

The **expc32.h** header file includes other header files as a convenience for your application. These files and their contents are as follows:

Header file	Contents
exp32def.h	Definitions
exp32err.h	Constant definitions for errors
exp32fnc.h	Function prototypes
exp32len.h	Constant definitions for field lengths
exp32str.h	Structure definitions
exp32val.h	Constant definitions

expc32m.lib link library

The **expc32m.lib** *link library* is the link library specifically for **expc32.dll,** which you link to your application to access the Expedite C-language API function calls. To use **expc32m.lib**, move it to the path specified in your LIBPATH environment variable.

Building the sample program

The sample code in the Expedite for Windows SDK includes samples for C in the SDK\CLANG subdirectory, for Microsoft Visual C in the SDK\visualc subdirectory, and for Microsoft Visual Basic in the SDK\visualbasic subdirectory. Each subdirectory contains a readme.txt file with an explanation of the files included and how to use them. This section gives an overview of using the sample code in the SDK\CLANG directory.

To execute the sample program successfully:

1. Set up the environment according to the hardware and software requirements.
2. Install the Microsoft Visual C++ compiler, if it is not installed already.
3. Copy **expc32.dll** to your LIBPATH directory.
4. Edit the sample makefile to specify the directories for the compiler and include files.
5. Compile the sample program using the sample makefiles.
6. Run the sample program.
7. After the sample program finishes, start the Expedite for Windows GUI to review the results of the execution of the sample program.

Deleting the sample project and address

If you want to delete the project and address created by the sample program, you can use the GUI or create a C-language interface program to do these tasks.

To delete the project and address using the GUI:

1. Start Expedite for Windows using the icon or the Start/Programs menu.

   If no other projects are installed, Expedite opens the project automatically for you.

2. If the navigator bar icons remain disabled after the splash screen closes, do the following:

   a. Select **File/Open** from the menu bar on the main window.

   b. Select **Project1** from the list in the dialog box.

   c. Click OK.

3. Select **Delete** from the File menu.

4. Click Yes to confirm the deletion.

   The project and address from the sample program are deleted.

To delete the project and address created by the sample program using the C-language interface, use the **ExpOpenProj** and **ExpDeleteProj** function calls in your application code.

Chapter 3

Using the Expedite for Windows graphical user interface

Using the graphical user interface (GUI) to perform tasks has several advantages over writing Expedite for Windows C-language interface code to perform the same tasks. The following sections detail how you can use the capabilities of the GUI to your advantage.

Using the GUI during setup

In most cases, rather than use the C-language interface, you can effectively use the GUI. Using the GUI during setup, you can minimize the amount of code required to use your application with Information Exchange.

You can use the GUI to:

- Set up the project

  In most cases, you set up a project once and distribute it with an application. (The only reason to use the project-related functions of the C-language interface for setup is if you want your users to create new projects using your application.)

- Set up address book records

  You can set up addresses at a central location and distribute the address book to your users. Or, you can let your users set up addresses, distribution lists, and trading profiles by using the GUI themselves. (There is no need to write a C-language interface to manipulate address book records.)

- Set up dropoff boxes

  You can set up dropoff boxes specifically for your application's needs. A dropoff box represents an Information Exchange session. You usually need to define in your application only a few dropoff boxes using the GUI. If your users need to change dropoff box settings to handle special problems, you can give them access to the dropoff box definitions through the GUI. (You do not need to write an interface to the dropoff boxes with the C-language interface.)

- Set up orders

  If your application performs the same data transfer activities each time it runs, use the GUI to set up orders. (If your application performs different activities each time, you can use the C-language interface to define orders as needed by the application.)

  Once orders are created, they are stored automatically on the order shelf for repeated use. If you do not want to store orders for your application, delete them from the order shelf after they process successfully.

- Maintain the receipt database

  Receipts for order processing are associated with the session receipt they are generated under and are stored on the receipt shelf. You can set up automated receipt maintenance in your project.

  If you set "Retain receipts" (the **RcpRetain** field in the **ExpAddProjReqStruct** structure) to a value other than 999 days, Expedite for Windows automatically deletes receipts older than the number of days you specify.

  You can let your users access and maintain receipts through the GUI, or, if you prefer, you can write your own receipt shelf maintenance code through the C-language interface.

NOTE: While the GUI provides an easy way to complete many setup tasks, the session setup tasks apply only to unattended or batch sessions. If your application needs to take action based on the result of each session command, use the C-language interface interactive session functions. For more information, see "Setting up unattended sessions" on page 21.

The following sections provide general information on using the GUI for some basic tasks. For more detailed information, see the *Expedite for Windows User's Guide.*

Setting up a project

If you provide the project creator name and additional information about your application and company in the description section of your project, it helps the GXS Community Support if problems arise. Customer Care can help resolve some problems and review session results, but you should let your users know how to reach your support and development staff to resolve problems for which Customer Care cannot provide help.

You should password protect your projects to prevent your users from changing previously created dropoff boxes, orders, trading profiles, or addresses. If you specify a *read password,* you must specify a *write password*. A read password allows your users to view information, create Information Exchange mailbox IDs, process dropoff boxes, or recover sessions. A write password allows your users to change information in any station.

Using the address book

The Expedite for Windows GUI requires that you set up an address book with Information Exchange addresses to create an order and process a dropoff box.

Expedite for Windows makes this process easy and convenient by using nicknames for the Information Exchange addresses. Each nickname must have at least one Information Exchange address format associated with it; that is, account and user ID, or alias type and alias. You can use the GUI to create a nickname for each Information Exchange address when you set up your address book. Then, use the nickname instead of the Information Exchange address when you create a new order. Or, you can let your users set up their nicknames and addresses using the GUI.

Instead of using the GUI to set up addresses, you can use the C-language interface to access addresses stored in a different format in another application. Also, because the C-language interface provides a complete set of functions to maintain the Expedite for Windows address book, you can write a more customized interface to the address book. For more detailed information on the C-language interface address book functions, see the *Software Development Kit Programming Reference.*

Using distribution lists in the address book

With Expedite for Windows, you can store distribution lists in the address book. These distribution lists are interpreted as temporary lists when Expedite for Windows defines them to Information Exchange. Because these distribution lists are used only for the duration of the session in which they were defined, your users can maintain their own. You can write an interface to distribution lists using the C-language interface, or you can have your users manage them using the GUI.

Working without an address book

The GUI requires users to set up address book entries with nicknames corresponding to the Information Exchange addresses (accounts and user IDs). The nicknames, not the accounts and user IDs, are used when communicating with Information Exchange. C-language interface users can communicate with Information Exchange using only the accounts and user IDs by creating orders with the **ExpAddSendOrdr** function call, and processing orders with the **ExpDoSendOrdr** function call.

If you store orders without nicknames, view them in the GUI, and then try to save the orders, the GUI asks you to specify a nickname and adds default values to other fields. To avoid changing orders added by the C-Language interface application, just select **Cancel**.

Using trading profiles

Trading profiles are a convenient way to define a set of attributes for an individual trading partner or group of trading partners, because you do not need to repeat the trading profile information on the orders.

While most of the information in a trading profile is used to send data to a trading partner, some information is required to receive data from a trading partner.

If a trading profile is set up for the address specified and no values are specified on the order, Expedite for Windows uses defaults as values for sending and receiving data. Trading profiles are a convenience and not required by either the GUI or the C-language interface. For definitions of the default values, see the *Software Development Kit Programming Reference*.

You can write an interface to trading profiles using the C-language interface or have your users manage the profiles through the GUI.

Setting up unattended sessions

If you want your application to process orders or groups of orders in unattended (batch) sessions, do one of the following:

- Use the GUI to set up the dropoff boxes and orders. Then, use the application to call the **ProcDropBox** function with the name of the dropoff box to be processed.

  OR

- Use the C-language interface to create the dropoff boxes and orders. Then, run them in unattended mode.

For more information, see "Preparing to send and receive files" on page 57.

For easy error recovery for unattended sessions, use the automated recovery options of Expedite for Windows. When the session ends, Expedite for Windows examines the return code in the session receipt and automatically recovers the session according to the option you selected.

Chapter 4

Programming to the C-language interface

This chapter explains how to program the Expedite for Windows C-language interface. It describes the function call input and output structures, and explains how to use the C-language interface to manage interactive or batch Information Exchange sessions. For a detailed description of each function call, see the *Software Development Kit Programming Reference.*

Input structures

The input structure fields have enough space to specify the maximum number of characters allowed by Request Manager. When using the structures to specify data for Request Manager, initialize the structure to blanks before adding data to it. Do not use nulls (binary zeros) in the structures, unless you want to use a Replace function to delete the value currently stored in a field in the database. Always left-justify the values and pad on the right with blanks.

Initializing the structure

Expedite for Windows provides a function named **ExpResetMem** to initialize the structure. Use the function **ExpAddField** to copy information to a data structure. The data should be passed in a null-terminated string, and the source string cannot exceed the maximum length for the target structure field.

The following example shows how to add (initialize) a project name to the **ExpAddProjReqStruct**.

```
ExpAddProjReqStruct Project;
ExpResetMem((char *)Project,
        sizeof(struct ExpAddProjReqStruct));
ExpAddField(Project.ProjName, “Purchasing”);
```

Replacing values in the structures

Expedite for Windows also provides a function to replace values in the structures. For **ExpReplaceField**, pass in the length of the field so it can be padded with blanks. The following example shows how to replace the project name specified in the previous example:

```
ExpReplaceField(Project.Creator,
                    EXP_MAXLEN_PJ_CREATOR,  “John Smith”);
```

To delete a field stored in the database, set the field in the structure to nulls (binary zeros or ‘\0’) instead of blanks.

Input structure naming conventions

In the C-language interface, function calls require an input structure to specify information about a request and a pointer to a data area to store the response information. The structure names are easy to remember because they reflect the name of the function they are used with, based on the following naming conventions:

- The function names and structures all begin with **Exp** to indicate that the function or structure is specific to Expedite for Windows.
- The **Exp** is followed by the verb or action of the request, such as **Add**, **Delete**, **Replace**, **List**, **Open**, **Close**, **Asg** (assign), and **Usg** (unassign).
- Finally, the verb is followed by the object of the request, such as: **DistList** (distribution list), **Addr** (address), **SendOrdr** (send order) or **Proj** (project).

For a detailed list of all the structures used to input C-language interface function calls, see the *Software Development Kit Programming Reference*.

Case sensitivity in input fields

For certain fields, especially one-character option fields, the values are case sensitive. For example, you must specify yes or no for some fields using **Y** or **N**. If you use a lowercase **y** or **n**, Expedite for Windows does not recognize the fields as valid.

Generally, names of records are not case sensitive. For example, nicknames on address records, dropoff box names, or order names are not case sensitive. Also, file IDs are not case sensitive in Windows or in Expedite for Windows.

If you search on a non-case-sensitive field, Expedite for Windows finds matches, regardless of case. For example, if you search for the nickname George, the following variations will match: GEORGE, george, and GEorge.

Refer to the *Software Development Kit Programming Reference* to confirm which fields are case sensitive.

Output structures

The following commands have a single output structure to hold the response information:

Add	Adds a new, unique object to the database. The response reflects what was stored in the database. Your application can use this information to make screen updates or to verify what was added to the database.
Replace	Replaces information specified about an existing database object. This object must exist in the database. The response echoes what was stored in the database. Your application can use this information to make screen updates or to verify what was changed in the database. The unique key value cannot be replaced. To remove the value of a field, you must set its value to binary zeros ('\0'). If you set the field to blanks, Expedite will leave its value unchanged.
Do	The **ExpStartSess**, **ExpEndSess**, and **Do** functions are for interactive sessions. For session start, session end, and send and receive functions, the output contains the receipt that is also stored in the database.

If you pass NULL (char *0) in place of a pointer to a response structure, you get only the return code and possible error information.

The following commands have no response output other than the return code:

Open	Opens the C-language interface or the database (mailbox, address book, project).
Close	Closes the C-language interface or database previously opened.
Delete	Removes an object from the database.
Assign	Attaches ownership of one object to another object.
Unassign	Removes previously assigned ownership of one object from its owner object.

The List command may have one or more structures in an array of response structures:

List	Contains information about an object or multiple objects. The List command may have multiple response structures passed back in an array. See the section on List functions below.

List functions

List functions have additional arguments besides the input and output structures. The output structure is actually an *array* of output structures. The **ExpDoQuery** function call also acts like a List function, since it lists the contents of the Information Exchange mailbox. The call below shows the general format of a List call.

```
ExpListObject( struct ExpListReqStruct *Input,
          unsigned short  *EntriesReturned,
          struct ExpObjectListedRspStruct *Output );
```

The following sections provide details for specifying the maximum number of structures to return and for qualifying a list.

Returning entries

To allocate quantities of memory to best fit the requirements of your application, the C-language interface function calls allow you to specify the maximum number of record structures to return on the response.

To read all the entries in the database, your application must include a loop that calls the list function, processes the results, initializes the response array, and then calls the list function again with the next start index. To do this:

1. Allocate memory for the response array by multiplying the size of the response structure by the number of entries you wish to retrieve with each list function call. For more information, see “Allocating space” on page 31.
2. Specify StartIndex as **1** and NumEntries as the number of entries you wish to retrieve with each list function call.
3. Look at the **EntriesReturned** argument to see how many data structures were returned.
   a. If the value of the **EntriesReturned** argument is equal to the number specified in the **NumEntries** field of the list request structure, there may be more records to retrieve.
   b. If the value of the **EntriesReturned** argument is less than the number specified in the **NumEntries** field of the list request structure, then there are no more data structures to be retrieved until you make a new call.
4. To continue, modify the **StartIndex** field of your request structure to the next index value from which to start your request.
5. Issue the same list function call again to retrieve more records that match the original request.
6. Repeat the **List** call until the C-language interface sets the **NumEntries** argument to a number less than **NumEntries** in the list request structure. This process is known as *paging through the database*.

Qualifying a list

In your application, you may find it necessary to list objects in the database that match certain specifications. In the C-language interface, this is called *qualifying* the list. Most of the **List** functions support the qualifiers.

The following is an example of the format of the qualifier structure:

```
struct ExpListReqStruct
  {
  char ProjName[EXP_MAXLEN_PJ_PROJ_NAME];
  char SearchParm[EXP_MAXLEN_SEARCH_PARM];
  char SearchOp[EXP_MAXLEN_SEARCH_OPER];
  char SearchValue[EXP_MAXLEN_SEARCH_VALUE];
  char SortParm[EXP_MAXLEN_SORT_PARM];
  char SortOrder[EXP_MAXLEN_SORT_ORDER];
  char NumEntries[EXP_MAXLEN_NUM_ENTRIES];
  char StartIndex[EXP_MAXLEN_START_INDEX];
  };
```

The structure qualifiers are defined as follows:

ProjName	Specifies the name of the project that this request is issued against.
SearchParm	Used with **SearchValue** and **SearchOp**, allows you to search for objects in the database that match the search criteria. **SearchParm** is a character field identifying the field on which to search.
SearchOp	Used with **SearchValue** and **SearchParm**, allows you to search for objects in the database that match the search criteria. **SearchOp** allows you to search for items using these parameters:

=	equal to
<>	not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

SearchValue	Used with **SearchParm** and **SearchOp**, allows you to search for objects in the database that match the search criteria. **SearchValue** is the character string to be matched with the **ParmID**.

SortParm	Specifies a field in the database on which to sort the **List** command responses.
SortOrder	Specifies a sort order, ascending or descending, for responses to the List command. **A** Ascending **D** Descending
NumEntries	Specifies the total number of entries to retrieve from the database for a **List** command. When used with **StartIndex**, you can page through the database entries. This value is required.
StartIndex	Specifies where in the database (on which record number) to start the list. When used with **NumEntries**, you can page through the database entries.

Response structure arrays with multiple kinds of objects

The array of response structures may contain structures of different types. In your application, allocate space for the responses by using the following formula, where **ObjectStructure** is the largest structure that could possibly be returned.

len = (sizeof(ExpObjectListedRspStruct)) * NumEntries;

ExpObjectListedRspStruct is the name of the structure used in the response for the function.

When the response structure array is returned, look at the **Type** field (usually the first field in the structure) to see how to cast the rest of the response information.

For example, if **OrdrType** is EXP_SEND_ORDR, cast using **ExpSendOrderRsp-Struct** to access the elements of the returned structure.

Following is an example of how to process response structures when there are multiple types of structures in the response, as in **ExpListOrdrShlf:**

```
ExpListReqStruct Input;
ExpListRpsStruct *Output;
ExpSendOrderListedRspStruct *SNptr;
ExpRecvOrderListedRspStruct *RVptr;



/* load input */
...
/* allocate space for output */
...
/* make the call */
rc = ExpListOrdrShlf(&Input, &EntriesReturned, Output);
if (rc)
  /* process error */
else
   {
  /* process the response */
  for (i = 0; i < MaxEntries; i++)
    {
    switch(Output[index]->OrdrType)
       {
      case EXP_SEND_ORDR:
         SNptr =  (ExpSendOrderListedStruct *)
                  Output[index];
         /* process data for send order */
         break;
      case EXP_RECV_ORDR:
         RVptr = (ExpRecvOrderListedStruct *)
                 Output[index];
         /* process data for send order */
         break;
       }
    }
 }
```

Allocating space

When allocating space, you need to allocate enough for the number of entries requested; that is, specify that number in the NumEntries field in the request structure of the **List** function. NumEntries is a required field that indicates how many structures of data should be retrieved by Request Manager for this command.

For example, to get entries 1 through 100 from the database, but only allocate space for 20 at a time:

1. Specify NumEntries as **20**.
2. Specify StartIndex as **1**.
3. After you issue the call once, process the 20 records returned.
4. Repeat the same call with StartIndex specified as **21**.
5. Continue this process with successive request calls, incrementing the StartIndex value each time until you obtain all results, or until Expedite sets the **EntriesReturned** argument to 0.

Evaluating the results of a function call

Each function returns an integer value that acts as a return code to let you know whether the request completed successfully. The return codes are:

0	No error exists. The request processed successfully.
4	Information-level error. The request processed, but some unusual condition occurred that may have caused incomplete processing. For example, if you tried to delete a record that was not in the database, the return is 4.
8	Error. The request was not processed because an error occurred. For example, if you tried to assign an address to a distribution list that does not exist, the return code is 8.
12	Interface terminated. The C-language component ended abnormally because of an error. For example, if Expedite/Base for Windows tried to write to a database file that was corrupted, the return code is 12.

Managing Information Exchange sessions

With the C-language interface, you can manage Information Exchange sessions as either interactive sessions or unattended sessions.

- Use an interactive session if your application does not use dropoff boxes and does need to maintain control between Information Exchange commands.
- Use an unattended (batch) session if a dropoff box is already set up and your application does not need to maintain control between Information Exchange commands.

Interactive sessions

An interactive session indicates that your application maintains control between Expedite for Windows C-language interface function calls. To execute an interactive session, use these functions in your application:

- **ExpOpenMsgSys**
- **ExpStartSess**
- **ExpDoSendOrdr**
- **ExpDoRecvOrdr**
- **ExpDoQuryOrdr**
- **ExpDoPurgOrdr**
- **ExpEndSess**
- **ExpCloseMsgSys**

On the response to the **ExpStartSess** function call, Expedite for Windows indicates the name of the session receipt for this session. Both the send and receive functions may result in multiple files being transferred. Each **ExpDoSendOrdr** and **ExpDoRecvOrdr** provides a summary of the file transfer activity in the response.

To obtain the detailed information, use the session receipt name in the **ExpListSessRcpt** function. For order receipts, use **ExpListRcptsOnSessRcpt**, specifying the session receipt name. You can also use the GUI to view the results of the session.

ExpDoQuryOrdr and **ExpDoPurgOrdr** do not store receipts in the database. The **ExpDoQuryOrdr** lists the contents of the Information Exchange mailbox, and as such, acts like a **List** function. **ExpDoPurgOrdr** returns only the return code for the processing.

If an error occurs during an interactive session that does not cause the line to be disconnected, your application can either continue or call the **ExpEndSess** function. If the line was disconnected, your application must call the **ExpStartSess** function again before issuing any data transfer commands.

Whether or not an error occurred, if your application does not end the session and closes the project or otherwise exits, Request Manager tries to send the session end command to Information Exchange. It also sets the session end return code to 0, marks the session as completed, and erases all control files associated with the session. The user ID from the **ExpStartSess** function is enabled. The next time your application starts a session, it is a brand new session or a “reset” session.

Interactive session recovery

Only file-level recovery is allowed for interactive sessions. Because each file transfer is either successful or not, and your application is in control, the actions of your application determine the recovery. Expedite for Windows’ automated recovery is not applicable to interactive sessions.

Unattended sessions

An unattended (batch) session indicates that your application does not maintain control between Expedite for Windows C-language interface function calls.

To execute an unattended session, use either the GUI or the C-language interface.

- If the following tasks are always to be done the same way, use the GUI once to:

    a. Set up a dropoff box and orders.

    a. Assign the orders to the dropoff box.

- If different orders are needed for a particular session, you can use the C-language interface to:

  a. Set up the dropoff boxes and orders as required using these functions:

     - **ExpOpenMbox**
     - **ExpAddDropBox**
     - **ExpAddSendOrder**
     - **ExpAddRecvOrdr**
     - **ExpAsgSendOrdrToDropBox**
     - **ExpAsgRecvOrdrToDropBox**
     - **ExpCloseMbox**

  b. Use the **ExpProcDropBox** function to start the session:

     - **ExpOpenMsgSys**
     - **ExpProcDropBox**
     - **ExpCloseMsgSys**

Control returns to your application when the session is completed. The response structure you passed in the **ExpProcDropBox** function contains the session receipt information, including a return code of 0, 4, 8, 12, or 13 to indicate the highest severity of any errors that may have occurred during the session.

Unattended session recovery

If your unattended (batch) session ends in error, the session is left in *restart recovery* state if the following occurred:

- You specified session recovery as checkpoint-level or file-level (**C** or **F**).
- You specified dropoff box recovery as “Leave as-is” (**L**).
- Expedite took a checkpoint with Information Exchange during the session.

The mailbox ID (account ID and user ID) associated with the session is disabled; that is, it cannot be used for any other sessions or dropoff boxes until this session is recovered.

Always fix the restart problem from Expedite, and avoid resetting the session using Information Exchange Administration Services. This is because Expedite needs to coordinate the session reset with Information Exchange. If you recover the session using Expedite, Expedite resets its control files. Then, the next time it starts a session with Information Exchange, it resets the Information Exchange side of the session as well. If you reset the session using only Information Exchange Administration Services, and you try to use Expedite to start a session, the session ends in *reset recovery* state with a "commit level mismatch" error. You then need to reset or cancel the session using Expedite.

You can recover a session manually by using the **ExpRecoverSessRcpt** function (see "Manual session recovery" on page 36), or you can direct Expedite for Windows to automatically recover by specifying the **ExpProcDropBox** function (see "Automated session recovery" on page 38).

Restart state: A restart state exists if a session connection breaks, but the session can be completed if reconnected. Common causes of this error are:

- A network component becomes unavailable.
- The modem disconnects during a session.
- An unexpected response is received from Information Exchange.

Reset state: A reset state exists when an unrecoverable error occurs. A common cause of this error is when the same account ID and user ID are used for two simultaneous sessions, either of which was using one of the checkpoint-level session recovery options.

NOTE: Do not use automated recovery type **R** if you are receiving multiple files with a single receive order, or if you are sending multiple EDI envelopes from a single file. This is because Expedite will remove the receipts for the data already sent and received for orders that are to be marked as "Pending," and your application will not be able to do recovery for data already sent or received.

In either restart or reset states, the Information Exchange account ID and user ID associated with the dropoff box (or session) cannot be used until the problem is resolved.

Manual session recovery

For manual recovery, set the DropRecovery option in the **ExpAddDropBoxReq-Struct** to blank or set to **L** for leave as-is. If session processing ends unsuccessfully, consider one of the following options in the design of the application:

Restart state: Design the application to restart upon an unsuccessful session end in one of the following ways:

- Restart immediately by using **ExpRecoverSessRcpt** with the Session Receipt name and the Recover fields set to **F** (finish processing immediately).
- Leave the session as-is to be completed in the future using the same **ExpRecov-erSessRcpt** function as above. The mailbox ID associated with the session may be disabled.
- Reset the session by using **ExpRecoverSessRcpt** with the Session Receipt and Recover fields set to **R** (reset session). All orders not committed are marked as *Pending*. To complete the processing, use **ExpRecoverSessRcpt** and set the Session Receipt and Recover fields to **F** (finish processing immediately).

  NOTE: Before you reset the session with the **ExpRecoverSessRcpt** function, you need to perform the following data recovery steps:

  1. Check the receive order receipt to see which ones were marked "Committed."
  2. Process the data received so it is not overwritten when the session is resumed.
  3. Check the send order receipts for orders in which you are sending multiple EDI envelopes from a single file.
  4. Remove each envelope that was already committed from the file before resuming the session.

  For more information, see "Resetting sessions after recovery" on page 72.

- Cancel the session by using **ExpRecoverSessRcpt** with the Session Receipt and Recover fields set to **C** (cancel session). Expedite sets the OrdrStatus on all orders not committed as **N** (canceled). The session, account ID, and user ID are considered recovered. However, the unprocessed orders cannot be completed at a future time.

Reset state: Design the application to reset upon an unsuccessful session end.

Resetting a session marks all uncompleted orders as *pending*, even if they were partially committed. Information Exchange only delivers sent and received files that were committed in their entirety when the session was reset.

Information Exchange discards any partially committed files. Resetting a session allows Expedite for Windows and Information Exchange to resume at the last agreed-upon checkpoint.

The application can leave the session as-is to be completed in the future using one of the options described below.

- Reset the session by using **ExpRecoverSessRcpt** with the Session Receipt name and the Recover field set to **R** (reset session). All orders not committed will be marked as pending. To complete the processing, use **ExpRecoverSessRcpt** and set the Session Receipt and Recover fields to **F** (finish processing immediately).

NOTE: Before you reset the session with the **ExpRecoverSessRcpt** function, you need to perform the following data recovery steps:

1. Check the receive order receipt to see which ones were marked “Committed.”
2. Process the data received so it is not overwritten when the session is resumed.
3. Check the send order receipts for orders in which you are sending multiple EDI envelopes from a single file.
4. Remove each envelope that was already committed from the file before resuming the session.

For more information, see “Resetting sessions after recovery” on page 72.

- Cancel the session by using **ExpRecoverSessRcpt** with the Session Receipt and Recover fields set to **C** (cancel session). The session, account ID, and user ID are considered recovered. However, the unprocessed orders cannot be completed at a future time.

Automated session recovery

Expedite for Windows allows your application to specify postprocessing parameters in advance, should Expedite for Windows be unable to complete a session with Information Exchange.

For such automated postprocessing for an unsuccessful session, set the DropRecovery option in the **ExpAddDropBoxReqStruct** or **ExpStartSessReqStruct** to:

R	Reset the session. Expedite for Windows marks the unprocessed orders as *Pending* for future processing.
C	Cancel the session. Expedite for Windows marks unprocessed orders as *Canceled*.

NOTE: Do not use automated recovery type **R** if you are receiving multiple files with a single receive order, or if you are sending multiple EDI envelopes from a single file. This is because Expedite will remove the receipts for the data already sent and received for orders that are to be marked as “Pending,” and your application will not be able to do recovery for data already sent or received.

Displaying session status

When processing a dropoff box or an interactive session, Expedite will display a session status window showing:

- The current command processing
- The result of the processing
- The number of files sent and received
- The number of bytes sent or received for the current file
- The progress indicator for files sent or received

You can turn off the display by specifying **N** in the AppStat field of the **ExpAddDropBoxReqStruct** or **ExpStartSessReqStruct** structures.

Chapter 5

Programming to the Java interface

This chapter explains how to write programs using the Expedite for Windows Java interface, and it describes the Java database *classes* that you use with your applications.

It is recommended that you have an understanding of the C-language interface used with Expedite for Windows. Review other chapters in this book for information on programming with the C-language interface. For a detailed description of the C-language function calls and their input and output structures, see the *Software Development Kit Programming Reference.*

The three samples included with the product are examples of how to use the Java classes provided with Expedite for Windows 6.2. The samples offer a graphical user interface example and two console examples, as further described in this chapter.

Overview of the Java interface

The Expedite for Windows Java interface enables Java programmers to write Java programs to control Expedite for Windows through the existing C-language interface. The Java programs can be written and compiled with the Sun Microsystems, Inc., Java Development Kit (JDK) or with a visual tool such as VisualAge for Java. This chapter includes sample programs written with both JDK and VisualAge.

The Expedite for Windows Java interface contains both a C dynamic link library (DLL) and Java classes.

- The C DLL contains the native Java functions that are called through the Java Native Interface (JNI). The Java interface provides the same level of functionality as the C-language interface.

- The Java code contains Java classes that represent the data and functions in the Expedite for Windows C-language interface. Each Java class contains data members that correspond to a record in the Expedite for Windows database. Each class also contains *methods* that call the actual C-language interface functions.

Comparing the C-language interface and the Java interface

To use the C-language interface, you must create the required input and output structures and call the C-API functions by passing the structures as parameters to the function. When the function is complete, the output structure contains data about the function that was called.

The Java interface hides the direct use of the C-API functions by allowing you to instantiate an object and call its methods. The methods actually call the C-API functions. Any data returned in the output structures is used to set the object's data members.

Using Java native methods

You do not need to call the Java native methods in your Java programs except to open and close the C-language interface; that is, your Java programs must contain the following calls: **ExpJni.jniOpenCInterface** and **ExpJni.jniCloseCInterface**. Other than this exception, you can write your programs using only the database classes provided.

In addition, you must also use the loadLibrary method of the System class to enable your Java application to use the Java interface DLL. See "Example of Java interface code" on page 41 to see this used in coding:

```
System.loadLibrary(“expjni”);
```

Example of Java interface code

The following is a simple console application that adds a new address to the Expedite for Windows database.

```
import ExpStructs.*;
import ExpDB.*;

public class SampleApp
{
    public static void main(String args[])
         {
         int rc;

         // Load the JNI DLL.
         System.loadLibrary("expjni");

         try
            {
            // Open the C-Language interface.
            rc = ExpJni.jniOpenCInterface();
            System.out.println("Interface opened. RC = " + rc);

            // Open the project.
            rc = ExpProject.open("SampleProject", " ");
            System.out.println("Project opened. RC = " + rc);

            // Create an ExpAddress object and set its data members.
            ExpAddress newAddress = new ExpAddress();
            newAddress.setNickname("joe");
            newAddress.setIeMboxId("account joe");
            newAddress.setFirstname("Joe");
            newAddress.setLastname("Smith");

            // Add the new address to the Expedite database.
            rc = newAddress.add();
            System.out.println("Add address. RC = " + rc);

            // Close the project.
            rc = ExpProject.close();
            System.out.println("Project closed. RC = " + rc);
            }

        catch (ExpException e)
            {
            reportException(e);
            }
            try
                {
                rc = ExpJni.jniCloseCInterface();
                System.out.println("Interface closed. RC = " + rc);
                }

            catch (ExpException e)
                {
                reportException(e);
                }
            }

       public static void reportException(ExpException theException)
            {
            System.out.println("Message from exception " +
 theException.getMessage());
            System.out.println("ReasonCode: " + theException.getReasonCode());
            System.out.println("ErrorMsg: " + theException.getErrorMsg());
            System.out.println("ErrorTxt: " + theException.getErrorTxt());
            System.out.println("Response: " + theException.getResponse());
            }
}
```

In the above example, the **ExpAddress** object (a database class) is created by calling **new ExpAddress()**. Its data members are set by using the “set” methods in the next four lines. Each database class contains get and set methods for all data members. The line that actually calls the C-language interface function to add the address to the database is **newAddress.add()**. When you call methods on any of the database objects, the C-language interface functions are called in the underlying code.

After the C-language interface function is called, the data members of **ExpAddress** are set with the data returned from the function call. This is not significant for the **ExpAddress** object, because the data returned from the function call is exactly the same as the data before the call was made. However, this is significant with other functions; for example, when the **ExpDropBox.process()** method is called, several data members, such as the receipt name and the session key, provide valuable information about the session. Refer to the response structures in the *Software Development Kit Programming Reference* to determine the data that is returned.

Handling errors

Use the **ExpException** class to handle errors in the C-language interface. All Java interface calls must be called within a *try/catch block*. When an **ExpException** is caught, the message of the exception is set to the return code from the C-language interface function call. The reason code, error message, error text, and user response messages can be obtained by calling the "get" methods of the **ExpException** class.

In the program sample on the previous pages, several Java interface calls are included in the first try block. An error in any of those calls will cause the program to jump into the catch block and run the exception code. The exception code simply calls **reportException,** which prints the exception onto the screen. Note that the call to **ExpJni.jniCloseCInterface** is in its own try/catch block, enabling the call that closes the interface to execute even if an exception occurred earlier in the program.

NOTE: The sample in "Example of Java interface code" on page 41 includes several calls in one try block to simplify the sample program, but this may not be the way you want to structure your program.

Compiling and running the sample Java programs

Three sample Java programs are included with Expedite for Windows. Two programs are simple console applications, and one program is a GUI application that was created with VisualAge® for Java. All sample programs can be compiled and run with either the JDK or VisualAge.

- The two console applications are located in the \Expedite\sdk\Java directory. The file names are SendFile.java and ReceiveFile.java.
- The GUI application is distributed as both a jar file and a VisualAge repository file.

If you are using the JDK only, follow the directions in "Compiling and running sample programs with the JDK" on page 44. If you are using VisualAge for Java, follow the directions in the "Compiling and running sample programs with VisualAge" on page 45.

Compiling and running sample programs with the JDK

Before running the samples with the JDK, copy the two DLLs in the \Expedite\sdk\bin directory into the current directory.

To compile the two console sample applications, do the following:

1. Bring up a DOS window and change the directory to:

   Expedite\sdk\java

2. Set the classpath to include the ExpJniClasses.jar file as follows:

   set CLASSPATH=%CLASSPATH%;ExpJniClasses.jar;.

3. To compile the sample programs with the Java compiler, type one of the following commands (depending on if you are sending or receiving) and press Enter:

   a. **javac SendFile.java**

   b. **javac ReceiveFile.java**

4. To run the sample programs, type one of the following commands and press Enter:

   a. **java SendFile**

   b. **java ReceiveFile**

The sample GUI application is distributed as a jar file that contains both Java source files and class files. It is not necessary to recompile the application before running it. However, you can compile the application using the following steps:

1. Bring up a DOS window and change the directory to:

   c:\Expedite\sdk\java

2. Set the classpath to include the ExpJniClasses.jar file and the vasample.jar file:

   set CLASSPATH=%CLASSPATH%;ExpJniClasses.jar;vasample.jar;.

3. To compile the sample program, type the following command and press Enter:

   javac com/ibm/expedite/jnisample/*.java

4. To run the sample program, type the following command and press Enter:

```
java com/ibm/expedite/jnisample/Driver
```

If you want to extract the source files from the jar file, type the following command:

```
jar -xvf ExpJniClasses.jar
```

Compiling and running sample programs with VisualAge

Before running the samples with VisualAge, you should either copy the two DLLs in the \Expedite\sdk\bin directory to the \windows directory, or add the \Expedite\sdk\bin directory to the path statement.

Before using the Expedite Java interface with VisualAge, you must import the \Expedite\sdk\java\ExpJniClasses.jar file. To do this:

1. Create a project in VisualAge called "Expedite Java Interface."
2. Import the \Expedite\sdk\java\ExpJniClasses.jar file by using the File/Import utility.

To import the two console sample applications, import the SendFile.java and ReceiveFile.java files using the File/Import utility. VisualAge compiles the programs during the import.

To run the programs, select the program and click on the run icon.

To import the GUI application, use the File/Import utility to import the VisualAge repository file \Expedite\sdk\java\vasample.dat.

To run the application, select the Driver class and click the run icon.

Using the sample GUI program

The following sample, created using VisualAge®, goes through an entire cycle of sending and receiving a file. Although this sample can be considered a functioning application, there are certain assumptions and limitations that you should consider.

Assumptions are:

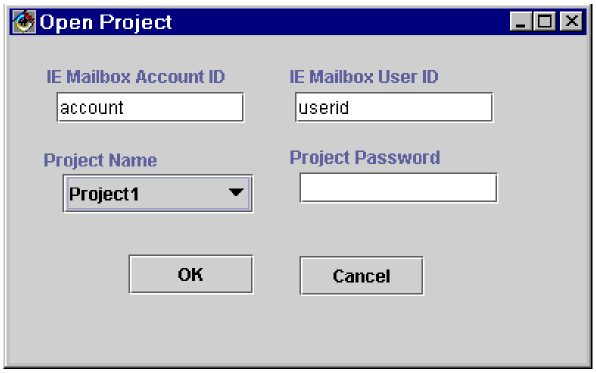
- An Information Exchange mailbox with a logon already exists for the user.
- A communications profile has already been created.
- Projects have already been added to the database.
- The sample uses only quick send and receive rather than drop-off boxes.

Limitations are:

- The sample opens only existing Information Exchange user ID logon
- The sample opens only existing projects

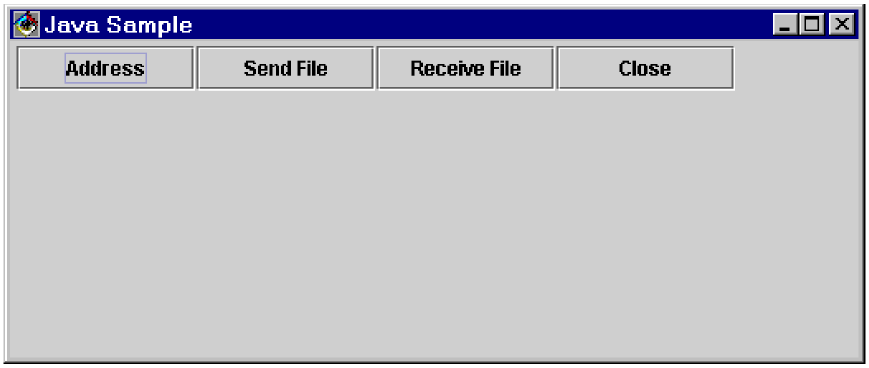
Launching the program

When you launch the program, the Open Project window displays. (Refer to previous sections for information on launching either the JDK or VisualAge programs.)



1. Enter an existing Information Exchange account ID and user ID.
2. Select the project you want to open from the drop-down list.
3. Enter the password if one exists for that project.
4. Click OK to continue with the Java sample. (Cancel ends the application.)

The main Java Sample window displays.



5. Click on one of the following selections:

   a. Address

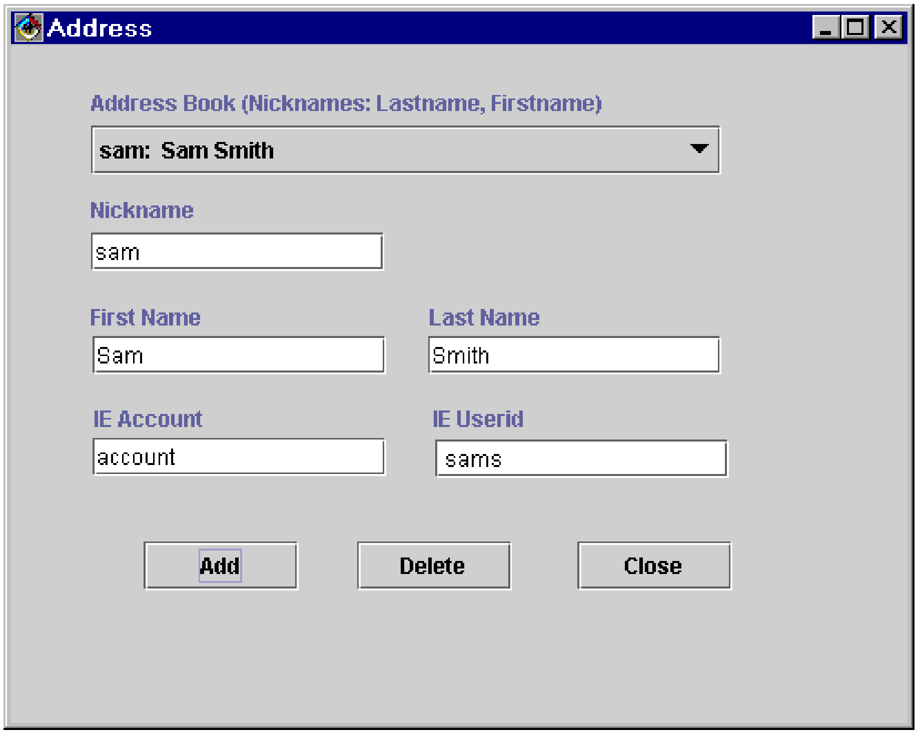
   b. Send File

   c. Receive File

   d. Close (closes the application)

Using the Address option

If you chose the Address option, the Address window displays. This window shows the current addresses in your address book in a drop-down list.



Address

Address Book (Nicknames: Lastname, Firstname): sam: Sam Smith

Nickname: sam

First Name: Sam

Last Name: Smith

IE Account: account

IE Userid: sams

To select an address:

1. Choose an address from the drop-down list.

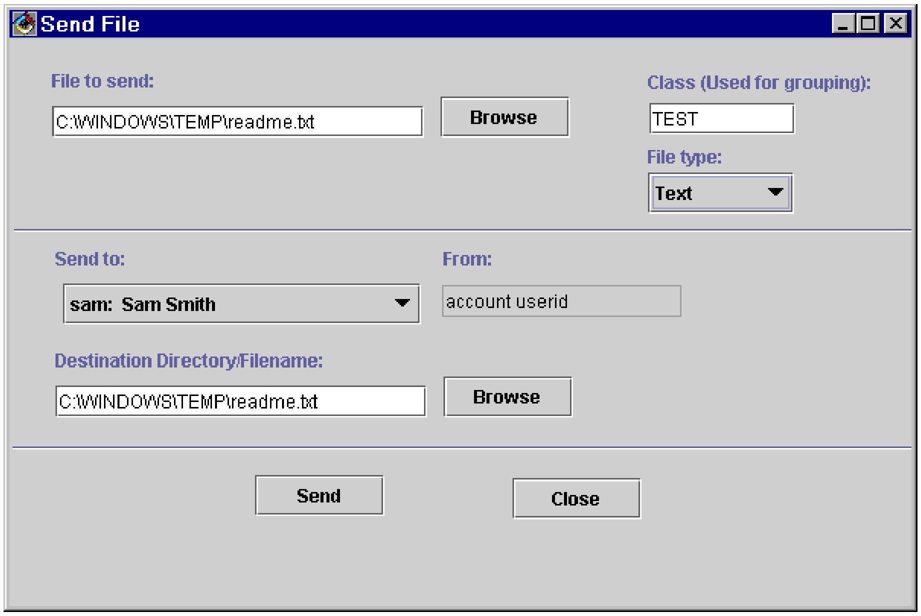
   Or, enter information in the other fields to add a new address.

2. Click either **Add** or **Delete** to add or delete the address you selected.

   Or, click **Close** to return to the previous window.

Using the Send File option

If you chose the Send File option, the Send File window displays. This window enables you to browse your drive to select the file you wish to send.



File to send: C:\WINDOWS\TEMP\readme.txt

Class (Used for grouping): TEST

File type: Text

Send to: sam: Sam Smith

From: account userid

Destination Directory/Filename: C:\WINDOWS\TEMP\readme.txt

To send files:

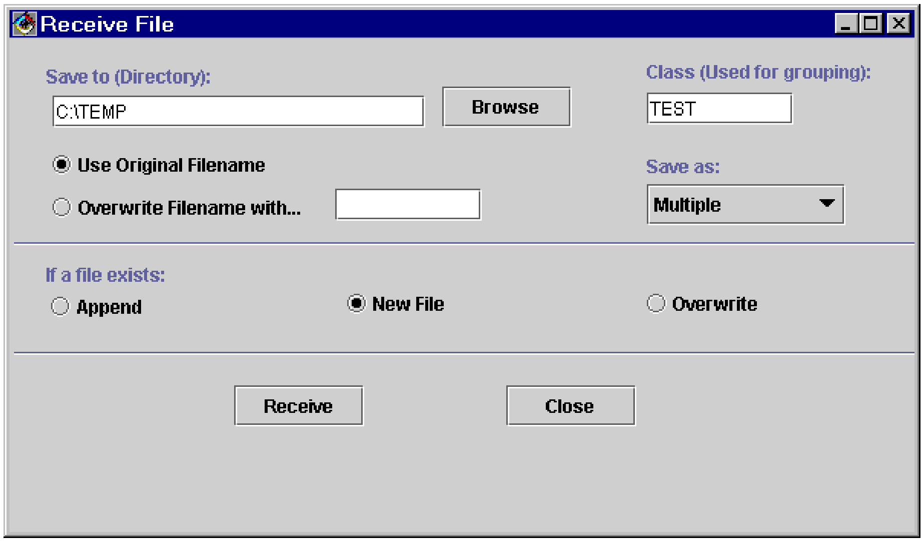
1. In the **Class** entry field, enter a group name to group files together.
2. From the **File type** drop-down list, select either Binary or Text.
3. From the **Send to** drop-down list, select an address to send the file to.

   (The **From** field indicates the logon ID you are using to log on to Information Exchange.)
4. In the **Destination Directory/Filename** field, either specify a directory path or browse your drive to enter this value.
5. Click **Send** to send the file or files.

   Or, click **Close** to return to the previous window.

Using the Receive File option

If you chose the Receive File option, the Receive File window displays. This window enables you to select where and how you want to receive files.



Save to (Directory): C:\TEMP

Class (Used for grouping): TEST

[✓ Use Original Filename]

Overwrite Filename with...

Save as: Multiple

If a file exists: Append [✓ New File] Overwrite

To receive files:

1. In the **Class** entry field, enter a group name to receive files only from that group.
2. Highlight one of the two buttons:
   a. **Use Original Filename** if you want to receive the file with its original file name.
   b. **Overwrite Filename with...** if you want to overwrite the original file name with a file name of your choice. Then, specify the name and extension of the file in the entry field to the right of this button.
3. From the **Save as** drop-down list, select whether you want to receive the files as a single file or as multiple files.

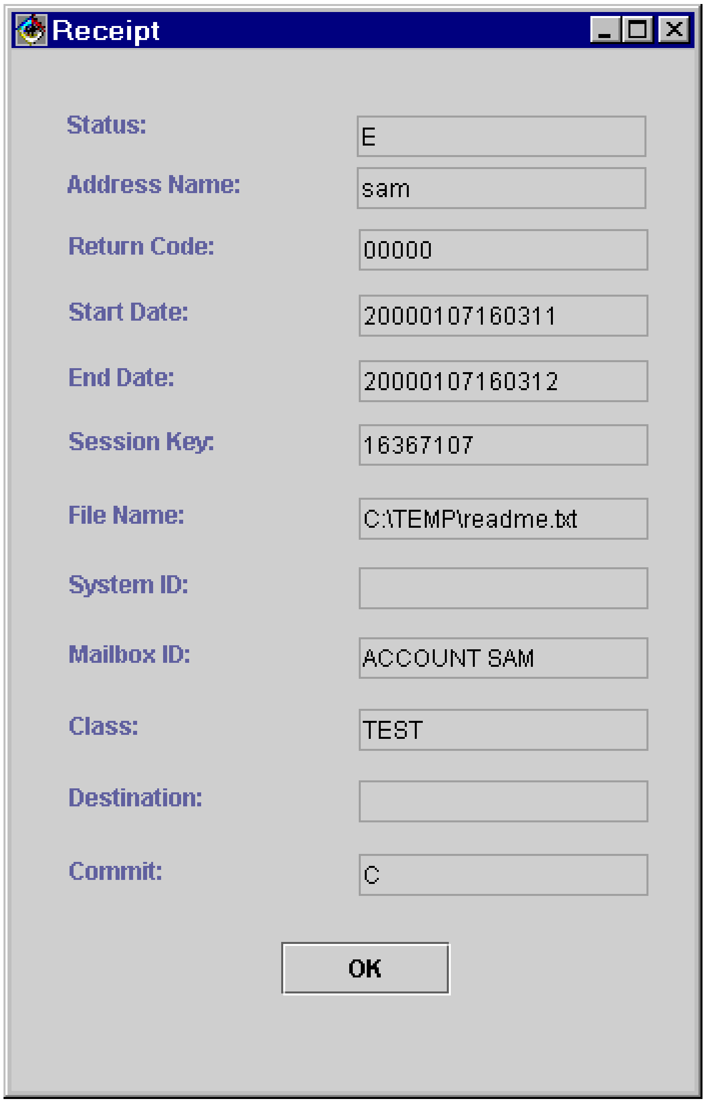
   When you save the files as multiple files, the file name you enter will have subsequent extensions of .002, and so on, for each separate file received.

4. If a file already exists in the directory you have chosen, select one of the following ways to receive the file:
   - **Append** appends the file to the existing file.
   - **New File** creates a new file with the .002 extension naming convention.
   - **Overwrite** writes over the existing file and replaces it with the new file.
5. Click **Receive** to receive the file or files.

   Or, click **Close** to return to the previous window.

Using the Receipt window

When you click either **Send** (on the Send File window) or **Receive** (on the Receive File window), the Receipt window displays showing the file transfer status. After the session completes and the files are transferred, the Receipt window lists detailed information about the file transfer.



Status:	E
Address Name:	sam
Return Code:	00000
Start Date:	20000107160311
End Date:	20000107160312
Session Key:	16367107
File Name:	C:\TEMP\readme.txt
System ID:	
Mailbox ID:	ACCOUNT SAM
Class:	TEST
Destination:	
Commit:	C

Click **OK** to return to either the Send or Receive window.

Using the sample console programs

This section describes two console samples that go through the cycle of sending a file and receiving a file, respectively. Each console file is written using the JDK and can be considered a functioning application except for certain assumptions and limitations.

The assumptions are:

- An Information Exchange mailbox with a logon already exists for the user.
- A communications profile has already been created.
- Projects have already been added to the database.
- The sample uses only quick send and receive rather than drop boxes.

The limitations are:

- The sample opens only existing Information Exchange user ID logons.
- The sample opens only existing projects.

NOTE: Before you can run the samples, you should ensure that you do NOT have Expedite for Windows 6.2 running.

SendFile prompts

To open the sample console programs, click the Run icon if you are using the VisualAge sample, or type **java sendfile** if you are using the JDK sample.

To send a file, answer the following questions as you are prompted to do so. (The default answers are shown below each question.)

1. What project would you like to use?

   Default = Pj1

2. What is the password for this project?
   (Press Enter if project does not have a password.)

   Default = <return>

3. What Information Exchange logon user ID would you like to use?
   (Press Enter to accept the default of Ig6000000.)

lg6000000

4. What directory and file name would you like to send?

   c:\temp\afile.txt

5. Who would you like to send the file to?
   (Enter the nickname.)

   jsmith

After you answer all the questions, the SendFile program executes and the Expedite Session Status window displays.

ReceiveFile prompts

To receive a file, answer the following questions as you are prompted to do so. (The default answers are shown below each question.)

1. What project would you like to use?

   Default = Pj1

2. What is the password for this project?
   (Press Enter if the prlject does not have a password.)

   Default = <return>

3. What Information Exchange logon user ID would you like to use?
   (Press Enter to accept the default of Ig6000000.)

   lg6000000

4. What directory and file name would you like to receive?

   c:\temp\recfile.txt

5. Would you like to receive as multiple files?
   (Enter Y or N.)

   y

6. Would you like to use the original sending file names?
   (Enter Y or N.)

   y

The ReceiveFile program executes, the Session Status window displays, and all the files in your mailbox are received. The DOS window then displays the receipt information.

Database classes and methods for each class

Following is a complete list of the database classes. For more information on the descriptions of each class and its methods, see the Appendix on the Java interface in the *Software Development Kit Programming Reference*.

public class ExpAddress
public class ExpAddressList extends LinkedList
public class ExpCommProfile
public class ExpDistributionList
public class ExpDistListList extends LinkedList
public class ExpDropBox
public class ExpDropBoxList extends LinkedList
public class ExpException extends Exception
public class ExpIeLogon
public class ExpIeLogonList extends LinkedList
public class ExpJni
public class ExpMailboxItem
public class ExpMailboxItemList extends LinkedList
public class ExpOrderList extends LinkedList
public class ExpProject
public class ExpProjectList extends LinkedList
public class ExpReceiveOrder
public class ExpReceiveReceipt
public class ExpReceiptList extends LinkedList
public class ExpReceiveOrderList extends LinkedList
public class ExpReceiveReceiptList extends LinkedList
public class ExpSendOrder
public class ExpSendReceipt
public class ExpSendOrderList extends LinkedList
public class ExpSendReceiptList extends LinkedList
public class ExpSessionReceipt
public class ExpSessionReceiptList extends LinkedList
public class ExpSession
public class ExpTradingProfile
public class ExpTradingProfileList extends LinkedList

Sending and receiving files with Expedite for Windows

This chapter explains how to use Expedite for Windows to send and receive text and binary files. It also details information about the common data header (CDH) and the Information Exchange translate table. Typical scenarios are included to help you learn more about sending and receiving files.

ExpAddSendOrdr and **ExpDoSendOrdr** functions are considered equivalent, except that the first *adds* a send order to the database to be processed later in an unattended session, and the second *processes* the send order immediately in an interactive session.

Settings described in this book for the **ExpAddSendOrdr** function also apply to the **ExpDoSendOrdr** function. Similarly, what is described for the **ExpAddRecvOrdr** function also applies for the **ExpDoRecvOrd**r function.

Preparing to send and receive files

How your application sends and receives data using the Expedite for Windows C-language interface depends on whether you choose to use an:

- Unattended (batch) session

  Where you process a list of orders in an unattended session.

- Interactive session

  Where you start the session, process the orders, retain control between order processing, and then end the session.

Processing in an unattended session

To process a list of orders in an unattended session:

1. Define a dropoff box.
2. Define send and receive orders on the order shelf.
3. Assign orders to the dropoff box.
4. Start dropoff box processing.
5. Decide what to do if there is a dropoff box processing error.

You can use the graphical user interface or the C-language interface for any of these steps. Even if you use the C-language interface to store orders on the order shelf or to create dropoff boxes, you can use the GUI to view and modify the orders or process a dropoff box.

Because your users can use the GUI to fix project problems, this simplifies your job of writing code to make your application work effectively with Expedite for Windows.

Processing in an interactive session

To process a list of orders with an application that uses an interactive session:

1. Start a session.
2. Issue send and receive orders.
3. Decide what to do if an error occurs with any order.
4. End the session.
5. Decide what to do if an error occurs within the session.

Planning for problem determination

You can write-protect your project so that users cannot change your order and dropoff box definitions without knowing the password. However, for problem determination reasons, if you specify a write password, you also must specify a read password to allow Customer Care to view the project definitions while helping users resolve problems.

The following sections provide more information about processing options.

Using unattended (batch) sessions

It may be easier to use the GUI to create orders, store them on the order shelf, define dropoff boxes, and add orders to them, than it is to use the C-language interface to create unattended sessions.

However, if you cannot anticipate which files should be sent to which addresses for a given session, you may want to use the C-language interface to dynamically create unattended sessions.

Creating unattended sessions

To use the Expedite for Windows C-language interface to create unattended sessions:

1. Create a dropoff box using the Expedite for Windows C-language **ExpAddDropBox** function, or create one in the graphical user interface.

   You can add and remove orders from this dropoff box as needed, or you can create multiple dropoff boxes to suit your needs.

2. Create send and receive orders using the **ExpAddSendOrdr** and **ExpAddRecvOrdr** functions, specifying the file names and recipient addresses, as needed.

3. Assign the orders to the dropoff box using the **ExpAsgSendOrdrToDropBox** and **ExpAsgRecvOrdrToDropBox** functions.

4. Process the dropoff box using the **ExpProcDropBox** function.

When all the orders are processed, the **ExpProcDropBox** function returns control to your application.

Expedite specifies the results of the processing as well as the name of the associated session receipt in the **ExpDropBoxProcRspStruct** structure passed in the **ExpProcDropBox** function.

Viewing processing results

To see the processing results for each order, use the **ExpListRcptsOnSessRcpt** function to pass in the name of the session receipt returned to your application from the output structure in the **ExpProcDropBox** function call.

Example of an unattended session

The following code shows some examples of how to set up an unattended session.

```
...
int rc = 0;
struct ExpAddDropBoxReqStruct DrInput;
struct ExpDropBoxAddedRspStruct DrOutput;
struct ExpAddSendOrdrReqStruct SnInput;
struct ExpSendOrdrAddedStruct SnOutput;
struct ExpAddRecvOrdrReqStruct RvInput;
struct ExpAddRecvOrdrAddedStruct RvOutput;
struct ExpAsgSendOrdrToDropBoxReqStruct AsgSendInput;
struct ExpAsgRecvOrdrToDropBoxReqStruct AsgRecvInput;
struct ExpProcDropBoxReqStruct ProcDrbxInput;
struct ExpDropBoxProcRspStruct ProcDrbxOuput;
struct ExpRecoverSessRcptReqStruct SessRcptInput;
struct ExpSessRcptRecoveredStruct SessRcptOutput;

/* Set up the dropoff box attributes */
...
/* Add the new dropoff box, exit if error */
rc =  ExpAddDropBox( &DrInput, DrOutput );
if (rc)
   goto exit;

/* Set up the send and receive orders */
...
/* Add send order */
rc = ExpAddSendOrdr( &SnInput,&SnOutput );
if (rc)
   goto exit;

rc = ExpAddRecvOrdr( &RvInput, &RvOutput );
if (rc)
   goto exit;

/* Assign them to the dropoff box */
rc = ExpAsgSendOrdrToDropBox( AsgSendInput );
if (rc)
   goto exit;
rc = ExpAsgRecvOrdrToDropBox( AsgRecvInput );
if (rc)
   goto exit;
/* Wait while Expedite processes the dropoff box */
rc = ExpProcDropBox( ProcDrbxInput, &ProcDrbxOutput);
```

```
/* Save the session receipt name */
strncpy( SRcptName, ProcDrbxOutput.SRcptName,
           EXP_MAXLEN_PD_SRCPT_NAME);

/* Process the results */
if (rc > 0)
  {
  /*Reset the session */
   ExpAddField( SessRcptInput.Recovery, "R");
   ExpAddField( SessRcptInput.SRcptName, SRcptName);
   rc = ExpRecoverSessRcpt( &SessRcptInput, &SessRcptOutput );
   ...
   }
```

Using interactive sessions

With the Expedite for Windows C-language interface, an application can control the flow of the session.

To control the session:

1. Use the **ExpStartSess** function to ask Request Manager to start interactive order processing.
2. To process an order immediately, use the **ExpDoSendOrdr** or **ExpDoRecvOrdr** functions for orders not kept on the order shelf.
3. Use the **ExpDoQuryOrdr** function to get a list of items in the mailbox.
4. Use the **Msgkey** field from the **ExpDoQuryOrdrDoneRspStruct** structure for the **ExpDoPurgOrdr** function to delete an item from the mailbox.
5. If an error has occurred, use the **ExpGetError** function immediately after checking the return code to get a text message describing the problem.
6. When you are done processing orders, use the **ExpEndSess** function to end the session with Information Exchange.

    a. Expedite for Windows adds a receipt for each order to the session receipt for the session processing, and returns control to your application as soon as each order is processed. The session receipt information is returned to the application in the output structure you provided for the **ExpEndSess** function call.

    b. For each send or receive order, you will receive a summary of the order's processing, showing how many files were sent or received as a result of the order.

7. Use the **ExpListRcptsOnSessRcpt** function to see the details of the order processing.
8. If you need to see the detailed results of a specific order, specify the search criteria in the **ExpListSendRcpt** or **ExpListRecvRcpt** functions to retrieve specific receipts.
9. To get a detailed message describing an error on a receipt, use the **ExpRetrieveErrTxt** function with the receipt name and error number.

Session recovery for interactive sessions

If an error occurs, the action of your application determines the recovery of the session. If either a function or the order processing fails, your application determines whether or not to continue. If your application does not end the session, but closes the project or otherwise exits, Expedite ends the session automatically.

You can select only file-level recovery for an interactive session recovery. In file-level recovery, the file is committed as each send or receive file order is processed.

Example of an interactive session

The following code sample shows some examples of using an interactive session. It does not include the code to set up the orders.

```
...
int rc;
struct ExpDoSendOrdrReqStruct SnInput;
struct ExpDoRecvOrdrReqStruct RvInput;
struct ExpSendOrdrDoneRspStruct SnOutput;
struct ExpRecvOrdrDoneRspStruct RvOutput;
struct ExpStartSessReqStruct StartSessInput;
struct ExpSessStartedRspStruct StartSessOutput;
struct ExpEndSessReqStruct EndSessInput;
struct ExpSessEndedRspStruct EndSessOutput;
/* Initialize structures */

ExpResetMem( (char *)&SnInput,
                sizeof(struct ExpDoSendOrdrReqStruct));
...

/*
 * Calls to open the C-API and open the Project, and open
 * the message system are not shown here.
 */
...

/*
 * Add the required fields for the session start
 */
 ExpAddField( StartSessInput.ProjName, “Personnel”);
 ExpAddField(StartSessInput.IeMboxIdName, “USER1”);

rc = ExpStartSess( &StartSessInput, &StartSessOutput);
if (rc)
   {
  /* Display the session start error and exit */
  ...
  goto exit;
```

```
   }

/* Send a file */

ExpAddField( SnInput.Fileid, “c:\records\employee.rec”);
ExpAddField( SnInput.Nickname, “Headquarters”);
ExpAddField( SnInput.AddrType, “E”);
rc = ExpDoSendOrdr( &SnInput, &SnOutput );

/*If only a warning, then continue, else exit */
if (rc > 4)
   {
   /* Display the send-file error and exit */
   ...
   goto exit;
   }

/* Receive a file */
ExpAddField( RvInput.Fileid, “c:\money\employee.sal”);
ExpAddField( RvInput.Class, “Salary”);
rc = ExpDoRecvOrdr( &RvInput, &RvOutput );
if (rc)
   {
   /* Display the receive-file error and exit */
   ...
   goto exit;
   }

exit:

/* End the session */
rc = ExpEndSess( &EndSessInput,  &EndSessOutput);
if (rc)
   {
   /* Display the error */
   }

/* Calls to close the message system, project, and the */
/* C-API are not shown.*/
...
```

Addressing files

Just as the address on an envelope must contain the proper address elements for delivery, so must the files you send to Information Exchange contain the proper address. When you send files, specify the Information Exchange mailbox address.

Expedite for Windows recognizes the following four address formats:

- Nicknames
- Account IDs, user IDs, and system IDs
- Centralized Information Exchange alias tables
- Distribution lists

Using nicknames

The most common and easiest way to address files is to use a nickname. You can create a nickname for each address when you set up your address book, and use the nickname when you create an order. For more information, see the *Expedite for Windows User's Guide.*

Using account IDs, user IDs, and system IDs

The Information Exchange mailbox address has two parts unique to each Information Exchange system: the account ID and the user ID.

When sending files to a user on the same Information Exchange system, specify the account ID and user ID. Each ID field is 1 to 8 characters in length.

When sending files to a user on another Information Exchange system, specify the system ID, the account ID, and the user ID. The system ID is 1 to 3 characters.

Using centralized Information Exchange alias tables

An alias table is a list of alternate names that you can use to send files to other users. Alias tables are permanent tables residing within Information Exchange that you can make available to your users. These include:

- *Global alias table* - for all Information Exchange users
- *Organization alias table* - for members of a particular account
- *Private alias table* - for a single user

Use Information Exchange Administration Services to create and maintain alias tables.

Using distribution lists

A distribution list is another way to send files to more than one person at a time by making a list of users and sending the file to the list. There are two types of distribution lists:

- *Temporary*

  Temporary distribution lists are created by Information Exchange and last only for the duration of your Information Exchange session. When your Information Exchange session ends, the system deletes the temporary lists.

- *Permanent*

  Not currently available in Expedite for Windows

Sending and receiving e-mail

Electronic mail (e-mail) is correspondence in the form of a file transmitted over a computer network. Various software packages handle e-mail differently, but all should ensure that the e-mail file looks the same to the receiver as it does to the sender.

Information Exchange e-mail files are made up of 79-byte records, padded with blanks if necessary. The 79-byte records are each followed by the characters that normally delimit records for the type of platform being used; for example, carriage-return line-feed (CRLF) characters. Information Exchange interfaces mark e-mail files as user class FFMSG001, so the receiving system knows how to format the e-mail.

Creating an e-mail file

To create an e-mail file:

1. Use an editor to create the text for the file with lines no longer than 79 bytes.

2. End each line with a CRLF character.

Sending an e-mail file

To send the file, use the **ExpAddSendOrdr** function with the Format field of the **ExpAddSendOrdrReqStruct** structure set to **Y**.

This tells Expedite for Windows to:

- Pad each line of text with blanks up to 79 bytes, or split lines that are greater than 79 bytes
- Add CRLF characters to each line
- Send the file with a user class of FFMSG001

Receiving an e-mail file

When you receive e-mail, set the Format field in the **ExpAddRecvOrdrReqStruct** structure to **Y** for the **ExpAddRecvOrdr** function, so that the file is properly received in the Expedite for Windows e-mail format for viewing.

NOTE: You can use the MsgClass field of the **ExpAddSendOrdr** function to specify a user class other than FFMSG001. However, the receiving system may not automatically recognize the files as having the Information Exchange e-mail format.

Sending and receiving ASCII text files

Generally, readable text on PCs consists of ASCII characters; whereas, readable text on most mainframe computers consists of EBCDIC characters. When Expedite for Windows sends a file, it does not know the receiving system type. Because the Information Exchange application resides on a host (mainframe) system, Expedite for Windows translates all ASCII (text) files to EBCDIC and marks the files as EBCDIC in the common data header (CDH) of each file.

When Expedite for Windows receives a file, it checks the CDH to see if the file is EBCDIC or binary. If the file is EBCDIC, it translates it to ASCII. If the file type is unknown because there is no CDH, Expedite for Windows assumes the file is EBCDIC and translates it to ASCII.

Sending and receiving ASCII binary files

When sending ASCII binary data to Information Exchange, avoid translating the binary data from ASCII to EBCDIC, because any changes to the binary data may render it unusable. Set the Datatype field to **B** on the **ExpAddSendOrdrReqStruct** structure to send ASCII binary files to Information Exchange and specify that it not be translated from ASCII to EBCDIC.

When Expedite for Windows receives a file, it checks the CDH to see if the file is binary. If so, it does not translate the data from EBCDIC to ASCII when receiving it.

Understanding the translate table

You can use the Information Exchange translate table, or an alternative Expedite for Windows translate table, for ASCII to EBCDIC translation.

The Information Exchange translate table, IESTDTBL, cannot be altered as it is contained within Expedite for Windows. It is not included on your program files. For a description of the Information Exchange translate table, see the *Software Development Kit Programming Reference*.

The Expedite for Windows program provides two alternate translate tables. The first table, IBM3270.XLT, corresponds to IBM Personal Communications/3270 program. The second table, NOXLATE.XLT, provides no translation at all. When Expedite for Windows receives a file that does not have a CDH, it assumes the file is EBCDIC and translates it to ASCII. If your trading partner uses a product that does not support the CDH and sends you a binary file, use the NOXLATE.XLT table to receive the file to ensure that it will not be altered.

To change translate tables, use the XlateTbl field in the input structure for the **ExpAddSendOrdr**, **ExpAddRecvOrdr**, and **ExpAddDropBox** functions.

NOTE: When you send a file to another PC, the alternate translate table used to send the file must be available on the receiving PC. If the alternate translate table is not available, the data is received as-is and may be unusable.

Using an alternate translate table

The following scenario and its code example illustrate specifying an alternate translate table using the XlateTbl field on the **ExpAddSendOrdr** function.

Company A uses a PC to send files to a trading partner. The trading partner uses a mainframe computer to receive and download the files to a PC using a 3270 emulation program. Because Company A knows how its trading partner receives files, it uses the IBM3270.XLT alternate translate table. This ensures translation on the sending system is the same as on the receiving system, and ensures that data is not damaged during translation.

NOTE: The **ExpAddField** function, which is included in the Expedite for Windows C-language interface DLL, is used in the following example. **ExpAddField** takes a value and puts it in the structure field. You should initialize the structure to blanks first.

```
ExpAddSendOrdrReqStruct Input;
int rc;

ExpResetMem( (char *)&Input, sizeof(ExpAddSendStruct ) );

ExpAddField(Input.OrdrName,SendOrder1);

ExpAddField(Input.Fileid, “file1.fil);

ExpAddField(Input.IeMboxId, “acct    user01”);

ExpAddField(Input.XlateTbl, “IBM3270”);

rc = ExpAddSendOrdr( &Input, NULL );
if (rc)
  goto exit;
...
```

The XlateTbl field tells Expedite for Windows to use IBM3270.XLT to translate the data in the files from ASCII to EBCDIC. When the trading partner receives and downloads the files to the PC, the data looks the same as it did on Company A’s mainframe computer.

Understanding recovery levels

Information Exchange supports several levels of recovery for a failed session, and Expedite for Windows can automate the recovery process for your application on these levels.

For unattended sessions, you can use one of the following recovery methods:

- Select a dropoff box recovery level before the session is run
- Allow the application to determine whether to recover the dropoff box using checkpoint-level or file-level recovery

Session-level, checkpoint-level, and file-level recovery are Information Exchange methods that Expedite for Windows uses to recover data at specific checkpoints. If you use session-level recovery when an error occurs, Expedite for Windows must retransmit all data for the entire session. If you use checkpoint-level or file-level recovery, Expedite for Windows can recover data more efficiently. Checkpoint-level recovery is the default.

To request a recovery method, use one of the following values for the DropRecovery field on the **ExpAddDropBox** function.

S	Session-level recovery
C	Checkpoint-level recovery (default)
F	File-level recovery

The processes for using checkpoint-level and file-level recovery are very similar. Considerations for restarting after an error and resetting the Expedite for Windows session apply to all three recovery methods.

Before you decide to restart or reset a session as part of recovery following a session error, consider the following:

- You should not run multiple sessions for the same Information Exchange account ID and user ID from different machines.

  If you begin an Information Exchange session using checkpoint-level or file-level recovery while another Information Exchange session with the same account and user ID is running, Information Exchange ends the first session and continues the second session. The results in the first session depend upon whether a checkpoint ended successfully.

- If a checkpoint ends successfully, Information Exchange delivers any data sent prior to the checkpoint, and deletes any received data from the mailbox prior to the checkpoint.
- If a checkpoint does not end successfully, Information Exchange does not deliver any data and does not delete any received data from the mailbox.

  This means that data received in the first session may be received again in error. In either case, you may get an error when you restart the first session.

The following table describes when Expedite does data recovery:

Checkpoint-level recovery	File-level recovery
• After sending the number of bytes you specify in the CommitData field of the **ExpAddDropBox** function (default is 141,000 bytes) • After processing a send order, if the next order is not another send order • While receiving files, if the files were segmented when they were sent • At the end of processing a receive order	• After each file or EDI envelope is sent. • After each file is received

Post-session processing for checkpoint-level and file-level recovery

When a session receipt status code is greater than 4, the session can be recovered.

The Information Exchange logon ID associated with the session must exist and must be disabled. The session control files must exist in the logon ID control file directory.

Under normal circumstances, the logon ID and the control files will be intact. Expedite for Windows does not allow a logon ID to be deleted if it is disabled or if it is used by any dropoff boxes.

Do not modify or delete Expedite control files in the CONTROL subdirectory under the install directory (for example, c:\expedite\control\*.*). If any of the conditions are not met, Expedite for Windows cancels the session automatically (since it cannot be recovered), enables the logon ID, and returns an error.

Resetting sessions after recovery

If you reset a session, Expedite for Windows marks as *pending* those orders that are not completely committed and completed, and it enables the Information Exchange logon ID. If the session receipt is selected, the processing starts with the first pending order at the *beginning* of the file and completes the session.

NOTE: Some of the files associated with the reset session may be committed and will not be sent or received again.

To reset sessions that end unsuccessfully, consider the following:

- If the session is unsuccessful, process the files received and committed or move them to another location. The order receipts will have a **C** (committed) **CmmtCode** for the files that were sent and received successfully.
- Erase the files sent or move them to another location. If you have multiple EDI envelopes in a file and some of the envelopes were processed, remove the processed envelopes from the file so Expedite for Windows does not send them again.
- When creating a receive order, specify MultFiles as **Y** (yes) and Overwrite as **G** (generate a new file) so that any additional files received are written to new files.
- If you select MultFiles as **N** (no) and/or Overwrite as **A** (append), the new files are appended to the files received before the session ended unsuccessfully.

Example of a session reset

The following example illustrates when a session reset is necessary. In this case, the session end return code is 24100, which indicates that the session and Information Exchange checkpoints do not match. This usually is caused by using the same account and user ID to access Information Exchange at the same time on different machines.

In this example, you are sending 10 files to account ID *act1* and user ID *user01,* and the session ends in error. The following, which is a code fragment of a batch session, shows how to reset the session if dropoff box processing fails:

```
ExpProcDropBoxReqStruct ProcDrbxInput;
ExpDropBoxProcRspStruct ProcDrbxOutput;
char SRcptName[EXP_MAXLEN_PD_SRCPT_NAME +1 ];

/* Initialize structures */

ExpResetMem( ProcDrbxInput,
                    sizeof( struct ExpProcDropBoxReqStruct));
...

/* Specify required ProcDropBox fields */
ExpAddField( ProcDrbxInput.DropBoxName, “Midnight ship”);
ExpAddField(ProcDrbxInput.ProjName, “Shipping”);
...
/* Wait while Expedite processes the dropoff box */
rc = ExpProcDropBox( ProcDrbxInput, &ProcDrbxOutput);

 /* Save the session receipt name */
 strncpy( SRcptName, ProcDrbxOutput.SRcptName,
             EXP_MAXLEN_PD_SRCPT_NAME);

 /* Process the results */
 if (rc > 0)
    {
   /*Reset the session */
    ExpAddField( SessRcptInput.Recovery, “R”);
    ExpAddField( SessRcptInput.SRcptName, SRcptName);
    rc = ExpRecoverSessRcpt( &SessRcptInput, &SessRcptOutput );
    ...
    }
```

Restarting and canceling sessions after recovery

If you *restart* a session, the session begins where it left off. In this case, Expedite for Windows may resume in the *middle* of a file. The processing starts at the last checkpoint in the first pending order.

If you *cancel* a session, Expedite for Windows marks unprocessed orders as canceled and enables the Information Exchange logon ID.

Checkpoint-level recovery and receipts

Order receipts show several types of status. The StatusCode field is a 5-digit number for an error that occurred. With this field, Expedite for Windows (or your application) associates text describing the message and offers response information.

In the case of send/receive orders, more than one receipt can be created. If you send a file with multiple EDI envelopes, Expedite for Windows separates the envelopes into individual files, resolves the address found in the EDI header, sends each as a separate file, and creates a receipt for each envelope processed.

For a receive order, more than one file in your mailbox may match the specified search criteria, resulting in these multiple files being received with a receipt for each file.

The first receipt created for each order is the OrdrStatus (order status*)* stating if the order is pending, issued, completed, or canceled. Each receipt contains a CmmtCode (commit code*)* field, which shows whether Expedite for Windows has committed the file associated with the receipt.

Depending on the type of session recovery selected, order receipts may show different combinations of OrdrStatus and CmmtCode.

- For session-level recovery, no files are committed until the session is ended successfully.
- For file-level recovery, each file is committed after it has been sent or received.
- For checkpoint-level recovery, data is committed according to the Commitdata and Maxmsgs options specified in the dropoff box (or sessions start request) for the session.

When using checkpoint-level session recovery, even if an order is completed, all the data may not be committed. Each order receipt shows whether the data was committed.

If a send order is marked “Completed,” but not all the order receipts are marked “Committed,” it means that some data was sent but not committed yet, and it will be sent again if the session is resumed and committed at the appropriate time. The recipient will receive all the data and no duplicate data.

Post-session processing for session-level recovery

When you encounter an error while using session-level recovery to transmit data, Expedite for Windows stops transmission and produces a session receipt with an error code. You can check this error code to determine the cause of the error and correct the problem. With session-level recovery, if data transmission stops, you must send or receive all files again. If you were processing a dropoff box, initiate the processing again.

Although it takes time to retransmit a large amount of data, there are advantages to using session-level recovery in the following cases:

- If you want to transmit all or none of the data
- If you transmit small amounts of data

NOTE: Do not start an Information Exchange session using session-level recovery while another Information Exchange session with the same account and user ID is running. If you do, Information Exchange ends the first session and starts the second session. Information Exchange does not deliver data sent in the first session, and does not delete received data from the mailbox. This means that data received in the first session may be received again in error, so the session results may be unpredictable.

Overwrite options

Unlike the Expedite Base for Windows product, the Expedite for Windows product usually does not allow files to be overwritten as a result of session reset or restart errors, because the mailbox ID is disabled for a session in checkpoint recovery state. However, if you specify Overwrite as **Y** for a dropoff box or on a receive order, and you process the dropoff box twice without processing files received in the first processing, the files will be overwritten in the second processing.

Receiving multiple files

When Expedite for Windows processes a receive order, it uses the attributes on the order to determine which files to receive from Information Exchange.

You can specify that Expedite for Windows receives:

- All files in the mailbox
- All files sent from a specific account and user ID
- All files with a specific user class
- Any combination of the above

In the receive order, you must specify the name of the file in which Expedite for Windows is to place the received data. If you have more than one file in your mailbox, you can receive the files from the mailbox into a single file, or receive each file into a separate file. When you receive multiple files from your mailbox into a single file, Expedite for Windows appends the files in the order it receives them. This is the default for receiving multiple files.

NOTE: When you receive multiple files from your mailbox into a single file, specify the value **Y** in the Remove eof field on a receive order to remove all end-of-file (EOF) characters before Expedite for Windows places the files on your PC. Otherwise, when you print a file that contains multiple files with EOF characters, the EOF character is interpreted as the end of the file and the data following that character may not print.

To receive multiple files from your mailbox in separate files, specify the value **Y** in the MultFiles field. Expedite for Windows places the first file in the file you specified and places subsequent files in consecutive files, incrementally numbered, starting with a file extension of 002. For example, if you specify FileID “test.msg” and three files are received, Expedite for Windows names the files:

File 1 = TEST.MSG
File 2 = TEST.MSG.002
File 3 = TEST.MSG.003

If you receive 1000 or more files, files 1001, 1002, and 1003 are named as follows:

File 1001 = TEST.MSG.1001
File 1002 = TEST.MSG.1002
File 1003 = TEST.MSG.1003

If you receive more than 99999 files, the data in the files after file 99999 is appended to a file with the extension of.ovf.

Using Expedite.ini to create file names for multiple files

If you want Expedite for Windows to create file names for received files using the traditional format, you can add FileNameFormat=1 to the ProtocolHandler section of the Expedite.ini file. When receiving multiple files from your mailbox, the second and subsequent files will have a name consisting of the file name specified on the receive command without the extension and with an added sequence number. For example, receiving multiple files using the file name afile.txt results in the following files being received:

afile.txt, afile.002, afile.003, ...

If you do not want the files named in the traditional way, do not specify FileNameFormat=1. An alternate style of file names will be used. For example, receiving multiple files using the file name afile.txt results in the following files being received:

afile.txt, afile.txt.002, afile.txt.003, ...

Receiving specific files

You can specify certain criteria on the Receive order to limit the files that you receive. For example, you can receive all files from a particular user, all files with a particular user class, or files with specific dates and times.

Receiving files from a specific time

You can use the Receive order to specify a date and time range for files you want to receive. Expedite for Windows checks the date and time the files were sent to you and gives you those files that fall within the specified range.

For example, to receive only those files sent to you between 12:00 noon and 6:00 p.m. on June 14, 1999, set up the Receive order as follows:

```
ExpAddField( RvInput.StartDate, “19990614120000”);

ExpAddField( RvInput.EndDate, “19990614180000”);
```

Receiving a single, specific file

Expedite for Windows also allows you to receive a single, specific file even if other files in your mailbox are from the same sender or have the same user class. Each file in your mailbox has a unique message key that distinguishes the file from all others.

To receive a specific file, issue a Receive order using the Msgkey field to specify the unique message key of the file you want to receive.

To determine the Msgkey:

- Issue an **ExpDoQuryOrd**r function call, and then examine the **ExpDoQury-OrdrDoneRspStruct** Msgkey fields

  OR

- Use Information Exchange Administration Services

For example, there may be three files in your mailbox from the same user, with the same user class. The files were sent to your mailbox on three consecutive days. However, you are only interested in receiving the first file, which has a unique message key of 887A9DE0021FA9C236F8.

Your **ExpDoRecvOrdrReqStruct** setup code might look as follows:

```
ExpAddField( RecvInput.Msgkey, “887A9DE0021FA9C236F8”);
or
memcpy( RecvInput.Msgkey, QueryOutput.Msgkey, 20 );
```

As a result of this command, Expedite for Windows receives only the file with this particular message key.

Chapter 7

Sending and receiving EDI data with Expedite for Windows

You can use Expedite to send and receive data formatted for electronic data interchange (EDI). Expedite provides the options Sendedi and Receiveedi on the **ExpDoSendOrdr** and **ExpDoRecvOrdr** functions to transmit EDI data. Use the value of **Y** in these fields to indicate the data should be treated as EDI data.

Understanding how Expedite sends EDI data

Information Exchange uses a two-part address to deliver mail to a trading partner's mailbox. The address consists of the account ID and user ID.

Information Exchange must have the account ID and user ID of the destination mailbox in order to deliver the file. Expedite allows users to create a nickname to represent the account ID and user ID of trading partners.

However, Expedite handles EDI data differently than does Information Exchange. When you send an EDI file with Expedite, the data in the file contains the destination address:

- In X12 data, the destination is stored in the ISA segment
- In UCS data, the destination is stored in the BG segment
- In EDIFACT data, the destination is stored in the UNB segment
- In UN/TDI data, the destination is stored in the STX segment

Specifying addresses

EDI standards allow you to specify addresses using different conventions. For example, you can:

- Specify a Dun and Bradstreet (DUNS) number for the address.
- Use phone numbers with Expedite and Information Exchange.

You can continue to use these kinds of addressing conventions, but you must provide information so that the various addresses can be converted to the account ID and user ID that Information Exchange needs to deliver the file.

Transmitting EDI envelopes

A group of EDI transactions with a single destination address is an EDI envelope, which consists of:

- The EDI header
- The data in the EDI transactions
- The EDI trailer

The EDI header contains the destination address for the data within the envelope. The format of the headers, data, and trailers differs depending on what type of EDI data is sent. When creating a Send order, you do not need to specify the destination address because Expedite can get the address from the data.

To indicate to Expedite that the data is EDI, use the **ExpDoSendOrdrStruct** or **ExpAddSendOrdrStruct** structures:

1. Specify the Nickname as **EDI**
2. Specify the AddrType field as **E**
3. Specify the Sendedi field as **Y**

Expedite can transmit multiple EDI envelopes with different addresses contained in a single file with a single send order. It can also transmit multiple types of EDI data from a single file. You can combine X12, UCS, EDIFACT, and UN/TDI data in one file and transmit it to multiple Information Exchange destinations with one Send order.

Using the Send order, you can send data to Information Exchange without specifying an Information Exchange destination as part of the order. Expedite can match EDI destinations contained in the EDI data to Information Exchange destinations.

Expedite determines where to send an EDI envelope by examining the contents of the envelope. The envelope definition (the type of EDI data you are transmitting) determines the location of the destination within an EDI envelope. Expedite must read the envelope header to extract the destination address. Expedite converts that address to a valid Information Exchange account ID and user ID mailbox address.

The following sections explain how Expedite converts the EDI address.

Using EDI envelopes

When you send EDI transactions, you can group the EDI transactions for a single destination within a single envelope.

The EDI envelope definitions for each EDI data type are as follows:

This EDI data type:	Uses this EDI envelope definition:
X12	Data between and including the ISA and IEA segments.
EDIFACT	Data between and including the UNA (or UNB) and UNZ segments.
UN/TDI	Data between and including the SCH (or STX) and END segments.
UCS	Data between and including the BG and EG segments.

The type of EDI data you are transmitting determines the location of the destination within an EDI header.

Expedite for Windows locates the EDI destination as described in the following table.

This EDI data type:	Contains the EDI destination in this segment:
X12	ISA. Expedite takes the actual EDI destination from the interchange receiver ID element (ISA08), and the EDI qualifier from the interchange ID qualifier element (ISA07).
EDIFACT	UNB. Expedite takes the destination from data element 0010 in composite data element S003 (Interchange Recipient), and the EDI qualifier from the data element 0007 in composite data element S003 (Interchange recipient).
UN/TDI	STX. Expedite takes the destination from the first subelement of the UNTO element (the recipient code address). If it does not find the recipient code, it uses the second subelement of the UNTO element (the recipient clear address) as the actual Information Exchange address.
UCS	BG. Expedite takes the destination from the application receiver’s code (BG04) element.

Resolving EDI destinations

Before sending EDI data, you should understand how Expedite converts EDI destinations to Information Exchange addresses.

Each Information Exchange mailbox is identified by a unique address. When you send data to Information Exchange, you must provide information so Information Exchange can determine the correct destination mailbox address. Information Exchange and the Expedite products can work with three different forms of addresses:

- Account ID and user ID

  Information Exchange must have the account ID and user ID in order to deliver the data to the proper mailbox.

- Alias table and alias name

  Information Exchange uses tables to convert the alias table and alias name combinations that you provide to the corresponding account ID and user ID.

- List

  When you send to a list, you should have defined a list of account IDs and user IDs, or alias tables and alias names.

The simplest scenario is specifying the destination address in the EDI data in terms of an Information Exchange account ID and user ID. This way, Expedite does not have to convert the address.

Providing destination address information in tables

EDI standards define sets of rules for specifying destination addresses. If you want to send EDI data to Information Exchange using a destination address other than an Information Exchange account ID and user ID, you must provide additional information so that Expedite and Information Exchange can convert an EDI address to an address that Information Exchange understands.

You provide this information using three tables:

- Qualifier
- Destination
- Alias

An overview of these tables is provided in the following flowchart, and a discussion of how the tables work is provided in the following sections. For more details, see “Creating tables for destination resolution” on page 100.

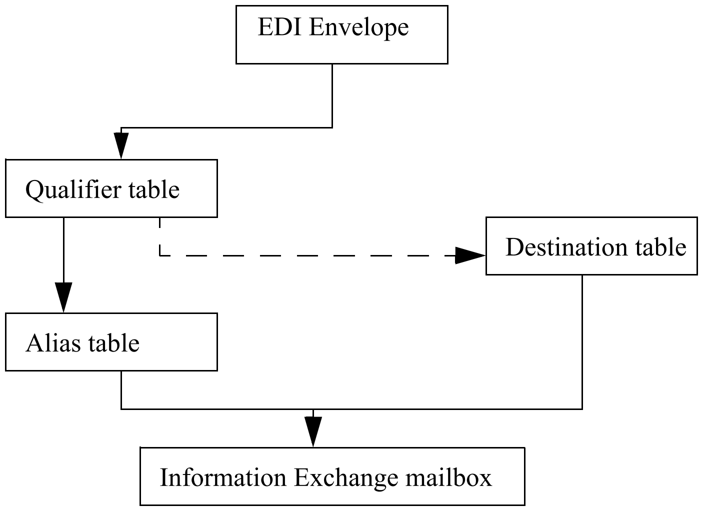


Figure 2. How the Send order locates EDI destinations

Bypassing tables

If you need to send only EDIFACT, X12, or UN/TDI data to an Information Exchange destination, and you specify an Information Exchange destination in the EDI header, you can bypass the tables and send the EDI data directly to an Information Exchange destination.

EDIFACT or X12 data

To send EDIFACT or X12 data to an Information Exchange destination contained within the EDI data:

1. Place ZZ in the receiver ID qualifier of the EDI header.
2. Use the Information Exchange account ID and user ID as the actual EDI destination.
   - For X12 data, separate the account ID and user ID by at least one blank. Otherwise, Expedite will use the first 7 characters as the account ID and the last 8 characters as the user ID.

   - For EDIFACT data, you must separate the account ID and user ID by a period (.), slash (/), or blank.

3. Use the send order to send the file containing your EDI data.

When you use a ZZ qualifier, Expedite tries to resolve the destination by searching the tables. When it does not locate the destination in the tables or the tables do not exist, Expedite sends the data to the specified Information Exchange account ID and user ID.

When you use a blank qualifier in the EDI header for X12 data, Expedite does not refer to the tables first. For EDIFACT data, however, a blank qualifier is treated the same way as any other qualifier and does not result in the tables being bypassed.

UN/TDI data

To send UN/TDI data to an Information Exchange destination contained within EDI data, follow these steps:

1. Do not specify the recipient code (UNTO:1).
2. Specify the Information Exchange account ID and user ID in the recipient clear address (UNTO:2). You must separate the account ID and user ID by a period (.), slash (/), or blank.
3. Use the Send order to send the file containing your EDI data.

Intersystem addressing

You can use intersystem addressing to transmit EDIFACT or UN/TDI data. To do this, you place an Information Exchange address in the EDIFACT or UN/TDI header, specifying the appropriate identifying information in the following order:

- System ID (not required if the sender and receiver are using the same system)
- Account ID
- User ID

All the IDs must be separated by one of the following:

- Period (.)
- Slash (/)
- One or more blank spaces

For EDIFACT data, the receiver code (data element 0010 in composite data element S003, Interchange Recipient) is split into the system ID, account ID, and user ID.

For UN/TDI data, the recipient clear code (UNTO:2) is split into the system ID, account ID, and user ID.

Using tables for UCS data

If you are sending UCS data to Information Exchange, you cannot bypass the use of tables. You must have one of the following in order to send UCS data:

- A ttable01.tbl file that specifies an Information Exchange account ID and user ID for the UCS destination address.
- A qualtbl.tbl file that identifies an Information Exchange alias table. The alias table must have an entry with the UCS destination address and the associated Information Exchange account ID and user ID.
- A qualtbl.tbl file that identifies a destination table to be used to translate the UCS destination address to an Information Exchange account ID and user ID.

How Expedite determines destinations without tables

When Expedite cannot find a destination or qualifier table, it determines the Information Exchange destination for each EDI data type as follows:

If the EDI data is:	Expedite determines Information Exchange destination as follows:
X12, and the qualifier is ZZ or blank	The Expedite order splits the receiver ID into an account ID and a user ID. Expedite looks for a blank character as the separator between account ID and user ID. Otherwise, it uses the first 7 characters as the account ID and the last 8 characters as the user ID.
X12, and the qualifier is not ZZ or blank	This is an error. Expedite does not send the data.
EDIFACT, and the qualifier (0007 in composite data element S003, Interchange Recipient) is ZZ or blank	Expedite splits the receiver code, which is 0010 in composite data element S003 (Interchange Recipient), into the account ID and user ID. The account ID and user ID are separated by a period (.), slash (/), or by one or more blank spaces.

If the EDI data is:	Expedite determines Information Exchange destination as follows:
EDIFACT, and the qualifier (0007 in composite data element S003, Interchange Recipient) is not ZZ or blank	This is an error. Expedite does not send the data.
UN/TDI, and the recipient code (UNTO:1) is not specified	Expedite splits the recipient clear code (UNTO:2) into an account ID and user ID. The account ID and user ID are separated by a period (.), slash (/), or by one or more blank spaces.
UN/TDI, and UNTO:1 was specified	This is an error. Expedite does not send the data.
UCS	This is an error. Expedite does not send the data.

Using EDI destination tables

If you do not specify Information Exchange destinations in the EDI header, you must first create an EDI destination table so that the EDI destination can be converted to an Information Exchange address. It is important that destination tables are stored in the subdirectory named SHARED in the Expedite install directory.

Think of an EDI destination table as a list of EDI destinations paired with Information Exchange destinations. Expedite resolves destinations by searching for an EDI destination and then using the corresponding Information Exchange destination as the actual address for an envelope.

Creating destination tables

You can create these tables using a plain text editor. If you created tables for Expedite 4.x, you can use the same tables for Expedite 5 and higher. The following is an example of the syntax for the file:

```
#comment or description
parameter(value) parameter(value) ... parameter(value);
```

where:

#
: Defines a comment line. Expedite ignores any characters on a line after this symbol.

parameter
: Defines a table entry. Parameters are not case sensitive.

value
: Defines the value associated with a table entry. Values are not case sensitive.

;
: Ends the table entry.

Each set of parameters and values, ended by a semicolon, defines one entry in the table. The entries can span several lines in a table file. However, the following limitations apply:

- Type the entire parameter name (for example, edidest) on a single input line.
- Ensure that a left parenthesis immediately follows each parameter.
- Do not use spaces between parameter names and values, and end each entry with a semicolon.

Naming destination tables

EDI destination tables have the following default naming convention:

This EDI data type:	Defaults to this EDI destination table:
X12	TTABLExx.TBL, where xx is the 2-character ID qualifier
EDIFACT	TTABLExx.TBL, where xx is the first 2 characters of the ID qualifier
UN/TDI	IEUNTDI.TBL
UCS	TTABLE01.TBL

Sending EDI data using a destination table

To send EDI data to a destination you define in an EDI destination table, follow these steps:

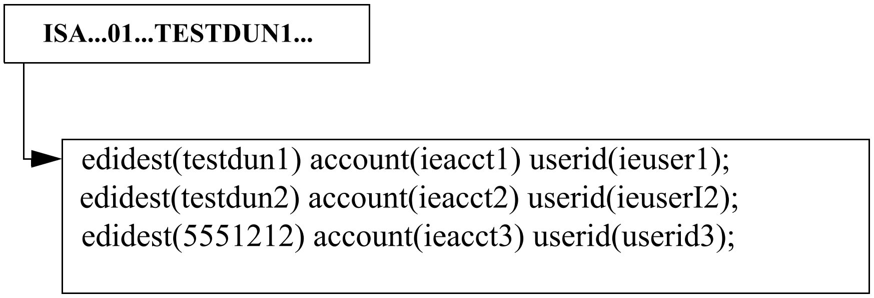
1. Build an EDI destination table containing your EDI destination and the corresponding Information Exchange destination.

   This table has the filename TTABLExx.TBL, where xx is the receiver ID qualifier in the EDI header.

2. Use a Send order to send the file containing your EDI destination.

Example of sending EDI data with destination table

The following example shows the tables Expedite uses to send an X12 file to a trading partner with an EDI destination of testdun1 and a qualifier of 01.



```
ISA...01...TESTDUN1...
```

```
edidest(testdun1) account(ieacct1) userid(ieuser1);
edidest(testdun2) account(ieacct2) userid(ieuserI2);
edidest(5551212) account(ieacct3) userid(userid3);
```

Figure 3. Sending an X12 file with an EDI destination table.

In this example, the receiver ID is testdun1 and the receiver ID qualifier is 01. An EDI qualifier table is not used.

Expedite searches the EDI destination table ttable01.tbl for the receiver ID testdun1 and converts the address to Information Exchange account ID ieacct1 and Information Exchange user ID ieuser1. The table ttable01.tbl is the default destination table for X12 data using an 01 receiver ID qualifier.

Using EDI qualifier tables

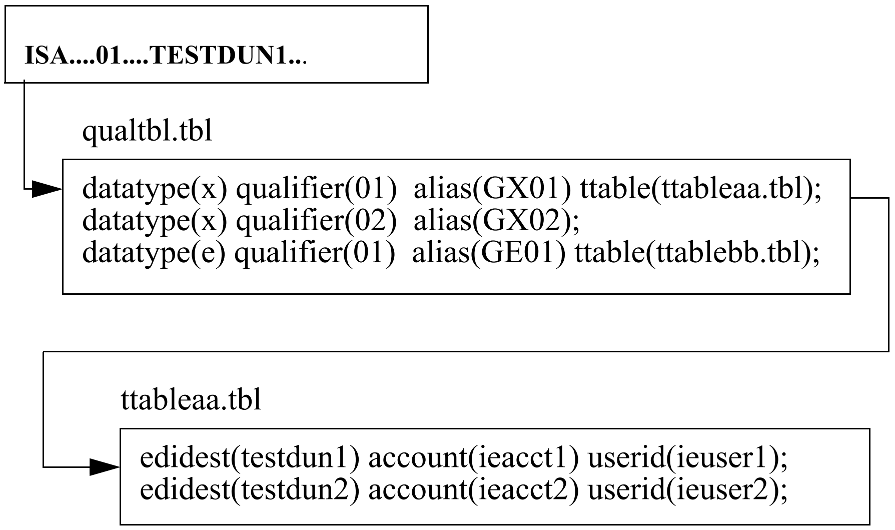
EDI qualifier tables tell Expedite which EDI destination table to use for this conversion. If you do not provide an EDI qualifier table, Expedite uses the default naming convention specified in the table in "Naming destination tables" on page 89.

If you find that the default naming convention is unsatisfactory, you can override the default destination table file names using the EDI qualifier table. The EDI qualifier table specifies a list of EDI destination tables to be used for specific types of EDI data.

Qualifier tables have the same syntax rules as a destination table.

Example of sending EDI data with qualifier table

Following is an example of sending X12 data using a qualifier table.



```
datatype(x) qualifier(01)  alias(GX01) ttable(ttableaa.tbl);
datatype(x) qualifier(02)  alias(GX02);
datatype(e) qualifier(01)  alias(GE01) ttable(ttablebb.tbl);
```

```
edidest(testdun1) account(ieacct1) userid(ieuser1);
edidest(testdun2) account(ieacct2) userid(ieuser2);
```

Figure 4. Sending an X12 file with an EDI qualifier table

In this example, X12 data is sent to receiver ID testdun1 with receiver ID qualifier 01. Expedite looks for qualtbl.tbl in the subdirectory named SHARED under the Expedite install directory.

Expedite searches the EDI qualifier table for a datatype of **x** and qualifier of **01**. This entry indicates that ttableaa.tbl should be used to convert the EDI destination to a proper Information Exchange address.

Expedite proceeds to search ttableaa.tbl for the receiver ID testdun1 and converts the address in the Information Exchange account ID ieacct1 to Information Exchange user ID ieuser1.

Using centralized Information Exchange alias tables

You may find it time consuming to maintain EDI destination tables in multiple locations. You can use centralized Information Exchange alias tables for more convenience.

These permanent tables reside within Information Exchange and are used to convert EDI destinations into Information Exchange destinations. You can make them available to all Information Exchange users (global alias tables), members of a particular account ID (organization alias tables), or individual users (private alias tables).

The EDI qualifier table and the EDI destination table determine which, if any, of these alias tables Expedite should use.

You can create and maintain alias tables by using Information Exchange Administration Services. For more information, see the *Information Exchange Administration Services User's Guide*.

NOTE: Expedite includes a sample qualifier table that defines standard centralized alias tables for all types of EDI data. In some Information Exchange installations (in the United States), these standard, centralized alias tables already exist, and you can add your EDI destinations to these tables to resolve your destinations.

Sending EDI data with a centralized alias table

To send EDI data to a destination defined in an Information Exchange centralized alias table, follow these steps:

1. Add the target EDI destination to an Information Exchange centralized alias table.
2. Build an EDI qualifier table that contains the name of the Information Exchange centralized alias table and specifies the EDI data type.
3. If you are not using a default EDI destination table file name, build an EDI qualifier table that contains the name of your EDI destination table and the type of EDI data it references.
4. Use the Send order to send the file containing your EDI data.

Example of sending EDI data with centralized alias table

The following example shows the tables Expedite uses to send an X12 file to a trading partner with an EDI destination of testsca1 and a qualifier of 02. This particular example does not include an EDI destination table.

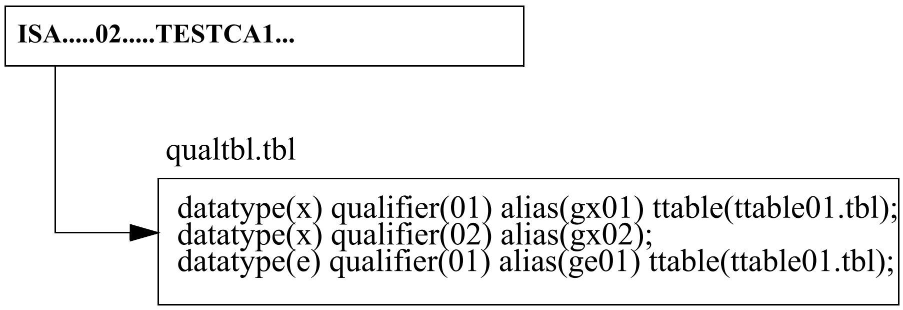

ISA.....02.....TESTCA1...
qualtbl.tbl
datatype(x) qualifier(01) alias(gx01) ttable(ttable01.tbl);
datatype(x) qualifier(02) alias(gx02);
datatype(e) qualifier(01) alias(ge01) ttable(ttable01.tbl);

Figure 5. Sending an X12 file with an Information Exchange centralized alias table

Expedite sends the data in this example to alias name testsca1 in Information Exchange alias table GX02. Information Exchange alias table GX02 is used to resolve the alias to an Information Exchange account ID and user ID.

Using Information Exchange distribution lists

To send EDI data to an Information Exchange distribution list:

1. Define the Information Exchange list.

   You can define a list using Information Exchange Administration Services.

2. Build an EDI destination table containing your EDI destination and the corresponding Information Exchange list name.

3. If you are not using a default EDI destination table file name, build an EDI qualifier table that contains the name of your EDI destination table and the type of EDI data it references.

4. Use the Send order to send the file containing your EDI data.

Example of sending EDI data to a distribution list

The following example shows the tables Expedite uses to send an EDIFACT file to an Information Exchange distribution list named list01.

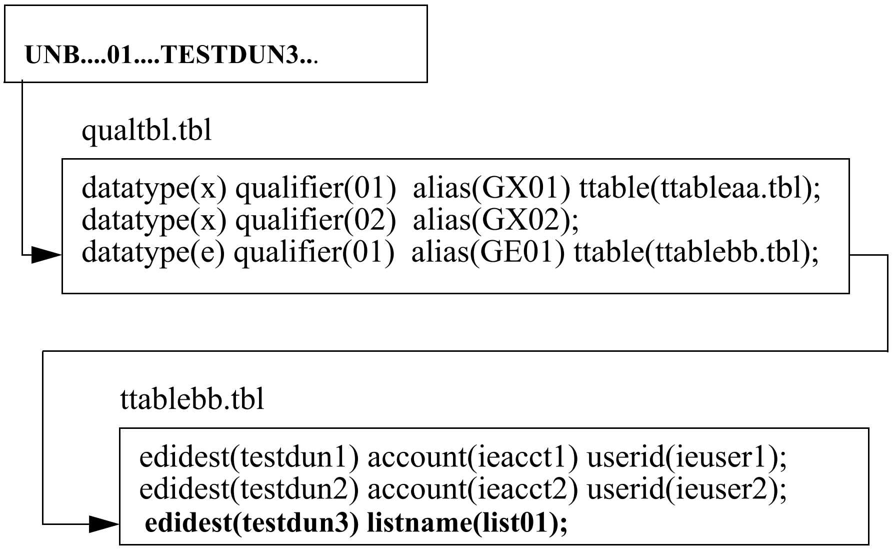

UNB....01....TESTDUN3...
qualtbl.tbl
datatype(x) qualifier(01) alias(GX01) ttable(ttableaa.tbl);
datatype(x) qualifier(02) alias(GX02);
datatype(e) qualifier(01) alias(GE01) ttable(ttablebb.tbl);
ttablebb.tbl
edidest(testdun1) account(ieacct1) userid(ieuser1);
edidest(testdun2) account(ieacct2) userid(ieuser2);
edidest(testdun3) listname(list01);

Figure 6. Sending an EDIFACT file to an Information Exchange distribution list

Expedite sends the data in this example to the Information Exchange list name list01.

Use Information Exchange Administration Services to create a permanent list. For more information, see the *Information Exchange Administration Services User's Guide*.

Specifying Information Exchange control fields

Information Exchange control fields allow you to assign information, which can be used by you and your trading partner, to the files you send. You can specify these control fields by specifying those values on the Send order.

Following are the Information Exchange control fields you can use on the **ExpAddSendOrdrStruct** and **ExpDoSendOrdrStruct** structures:

- MsgName

  An alphanumeric identifier for your file.

- MsgSeqNo

  A numeric identifier for your file.

- MsgClass

  An alphanumeric identifier for your file. The user class can be used by your trading partner when receiving the file.

Providing a message name

When you use the Send order and provide a message name parameter, Expedite uses this value for the Information Exchange message name. If you do not provide a message name parameter, Expedite generates the Information Exchange message name based on the EDI data type. The following table describes how Expedite generates the message name.

This EDI data type:	Generates this message name:
EDIFACT data	Expedite takes the message name from the data element 0020 (Interchange Control Reference) of the EDI data. If the element exceeds 8 bytes in length, Expedite uses the first 8 bytes. If the element is fewer than 8 bytes in length, Expedite places it in the message name field, left-justified and padded with blanks.
UN/TDI data	Expedite takes the message name from the sender’s reference field (SNRF) of the EDI data. If the SNRF exceeds 8 bytes in length, Expedite uses the first 8 bytes. If the SNRF is fewer than 8 bytes in length, Expedite places it in the message name field, left-justified and padded with blanks.

This EDI data type:	Generates this message name:
X12 data	Expedite takes the message name from the last 8 bytes of the interchange control number of the X12 data.
UCS data	Expedite takes the message name from the interchange control number. Because the UCS interchange control number has a maximum length of 5 bytes, Expedite places the interchange control number in the message name field, left-justified and padded with blanks.

Providing a message sequence number

When you use the Send order and provide a message sequence number, Expedite uses this value for the Information Exchange message sequence number. If you do not provide the message sequence number, the Send order generates an Information Exchange message sequence number in the following manner for all EDI data types:

- Expedite counts each EDI envelope it transmits from a single data file.
- Expedite formats the message sequence number as a series of numeric characters ranging from 00001 to 99999.
- Expedite assigns and places the count of each EDI envelope in the message sequence number of the corresponding Information Exchange messages.

  For example, three EDI envelopes Expedite sent from a single file would have the following message sequence number values:

  - 00001 for the first EDI envelope in the file
  - 00002 for the second EDI envelope in the file
  - 00003 for the last EDI envelope in the file

When the message sequence number reaches 99999, it automatically resets to 00001. The message sequence number counter also resets to 00001 each time you use the Send order for a new file of EDI envelopes.

Providing a user class

If you do not provide a user class parameter on the Send order, Expedite generates a parameter value based on the EDI data type.

User class for EDIFACT and UN/TDI data

For EDIFACT and UN/TDI data, Expedite takes the user class from the application reference field (APRF) of the EDI data. If the APRF exceeds 8 bytes in length, Expedite uses the first 8 bytes. If the APRF is fewer than 8 bytes in length, Expedite places the APRF in the user class field, left-justified and padded with blanks. If the APRF is not present, Expedite sets the user class as follows:

This EDI data type:	Defaults to this class:
EDIFACT	#EE
UN/TDI	#EU

User class for X12 and UCS data

For X12 and UCS data, if you do not specify a user class, Expedite sets the class as follows:

This EDI data type:	Defaults to this class:
UCS data	#EC
X12 data	#E2

Inserting blanks following EDI segments

If you insert blanks at the end of EDI segments when preparing EDI information, Expedite does not consider the blanks part of the EDI data. Expedite accepts EDI data with blanks after the segment terminators, but it removes these blanks before transmitting the data to Information Exchange.

Using Expedite order receipts

For every file sent, Expedite records a send receipt. For EDI data, each envelope is considered a file. If there is more than one envelope in a file, Expedite writes one receipt for each envelope, adding a sequence number to differentiate between the receipts produced for the same order.

The send order receipts keep track of the EDI envelopes that Expedite sent. These receipts also include a Verify field that indicates whether or not Expedite sent the EDI envelope.

The field provides a record of the EDI envelopes not sent by the Send order because of a destination verification failure, if you specified that Expedite should continue processing a file with multiple EDI envelopes when a destination verification fails. (Verify or Verify if possible with the On error field set to Continue.)

Verification is specified in the **ExpDoSendOrdrStruct** structure Verify field, with values **Y** or **F** for Always verify or Verify if possible, respectively. The continue processing value is specified in the EdiVerifyProc field as **C** for continue and **S** for stop.

NOTE: You can use the receipts to determine which EDI envelopes have been sent when the Send order does not compete successfully.

Receiving EDI data

You use the Expedite Receive order to receive EDI data from Information Exchange by specifying **Y** in the Receiveedi field in the **ExpDoRecvOrdrStruct** or **ExpAddRecvOrdrStruct** structure to receive as EDI data.

When you indicate the data to be received is EDI, Expedite parses the header and includes that information in the receipt. You can also tell Expedite to do special formatting on the data.

Specifying automatic EDI processing

Expedite provides an automatic EDI processing option. If you specify Receiveedi as **N** or leave it blank, and the data is marked as EDI in the CDH, Expedite processes the data as EDI. This way you can receive both EDI and non-EDI data with the same order and have the EDI data processed appropriately. You can also indicate that Expedite should ignore the CDH and process the data as EDI.

Use the Autoedi field on the **ExpDoRecvOrdrStruct** or **ExpAddRecvOrdrStruct** structure and specify **Y**, **N**, or **F** for process as EDI, receive as-is, or ignore CDH and process as EDI.

Specifying CRLF characters

If you specify EDI processing, Expedite checks the processing option to see where to break the EDI records with carriage-return line-feed (CRLF) characters: after segment delimiters, nowhere, or after a fixed number of characters in the EDI data.

Use the Ediopt field on the **ExpDoRecvOrdrStruc**t or **ExpAddRecvOrdrStruct** structure to specify one of the following:

- **Y** to add CRLF characters after segment delimiters
- **N** to indicate that no CRLF characters should be added
- **B** to indicate CRLF characters should be added after a fixed number of bytes

  If you specify **B,** indicate the number of characters in the Recordsize field.

Specifying only EDI data to be received

You can tell Expedite to receive only data that is marked as EDI data in the CDH. To do this, specify **Y** on the EdiOnly field in the **ExpDoRecvOrdrStruc**t or **ExpAd-dRecvOrdrStruct** structure.

This method is especially useful if you intend to use a translator to translate from EDI standard format to proprietary format. Some translators may have problems trying to translate data that is not really in an EDI format.

If the system sending the EDI data does not support the CDH, there is no way to guarantee that the data you receive is EDI. However, you can make arrangements with your trading partner to help ensure that you receive EDI data. For example, you and your trading partner can agree that all EDI data is sent with a user class of edidata. All non-EDI data must have a different user class. That way, if your EDI receive order receives from user class edidata, the data should be EDI.

This method does not guarantee that the data in the file is EDI; for example, your trading partner may mistakenly send a non-EDI file to your mailbox with a user class of edidata. However, this method does improve the chances that the data is really EDI.

If the trading partner sending you data uses the default user classes of the Send order, you can use the wildcard receive feature of Information Exchange to simplify receipt. For example, by specifying **#E?** as the user class on the Expedite order, you can ask Information Exchange to return only files that have a user class beginning with #E. This includes all files sent with the default EDI user classes.

If there is no CDH, Expedite attempts to process the files as EDI data. If this is not possible, the program writes the data without reformatting.

NOTE: The Send order places a single EDI envelope in an Information Exchange file. The Expedite order expects each EDI envelope to be contained in a separate Information Exchange file. If you put multiple EDI envelopes in a single file and send them without specifying SendEdi as **Y**, the entire file is sent to the Information Exchange account ID and user ID specified on the Send order regardless of the destinations specified in the EDI envelopes within the file.

Creating tables for destination resolution

The following sections provide information on the format of the EDI qualifier table and EDI destination table. When you are sending data, these tables enable Expedite to resolve destinations. Use these formats to create your tables.

Understanding the EDI qualifier table entry format

The file qualtbl.tbl. contains the EDI qualifier table. Each entry in this table indicates an EDI destination table, or a centralized alias table, or both, for a given EDI qualifier or EDI data type.

The format for an EDI qualifier table entry is:

datatype
The EDI data type. Use one of the following values:

- blank - all EDI data types
- **X** - X.12 data
- **C** - UCS data
- **E** - EDIFACT data
- **U** - UN/TDI data

qualifier
The EDI qualifier. If the qualifier parameter is blank, this entry matches any EDI qualifier. Use 1 to 4 alphanumeric characters. The default is blank.

ttable

The name of the EDI destination table. If you specify this parameter, Expedite uses that table to try to resolve the Information Exchange destination. If you do not specify a destination table, Expedite does not use a table to match EDI data and resolve the destination. Use a standard file name, 1 to 12 alphanumeric characters.

alias

The name of the centralized alias table. If you specify this parameter and Expedite does not find the destination in the EDI destination table, Expedite uses this alias table with the EDI receiver ID as the alias name. Use one of the following values:

- blank for a centralized alias table. This is the default.
- **g***xxx* for a global alias table, where *xxx* identifies a 1- to 3-character table name.
- **o***xxx* for an organizational alias table, where *xxx* identifies a 1- to 3-character table name.
- **p***xxx* for a private alias table, where *xxx* identifies a 1- to 3-character table name.

You can include comments in the qualifier table by specifying # as the first character on the line.

Example of a qualifier table entry

The following is an example of a qualifier table entry.

# Destination table for southeast area trading partners

datatype(x) qualifier(01) ttable(ttable01.tbl alias(gx01);

Understanding the EDI destination table entry format

Use the TTABLE parameter in the EDI qualifier to specify the name of your EDI destination table. Each entry in this table indicates the Information Exchange destination associated with a given EDI receiver ID.

The format of an EDI destination table entry is:

```
edidest(EDI destination)
alias(alias) aliasname(alias name);
or
sysid(systemID) account(account) userid(user ID);
or
account(account ID) userid(user ID);
or listname(list name);
```

edidest
The EDI receiver ID. If this parameter matches the receiver ID from the EDI data, Expedite sends the message to the Information Exchange destination using the parameters you select. Use 1 to 35 alphanumeric characters.

alias
The alias table type and table name.

aliasname
An alias name defined in the alias table. Use 1 to 16 alphanumeric characters.

sysid
The system ID of a single-destination user ID. You need this only if the account ID and user ID you specify reside on another Information Exchange system. Use this parameter only with the account and userid parameters. Use 1 to 3 alphanumeric characters.

account
The Information Exchange account ID name of a single-destination user ID. Use 1 to 8 alphanumeric characters.

userid
The Information Exchange destination user ID. Use 1 to 8 alphanumeric characters.

listname

The name of a previously defined list of account IDs and user IDs. Use 1 to 8 alphanumeric characters.

You can include comments in the destination table.

Example of an EDI destination table entry

The following is an example of an EDI destination table entry:

```
# IE address for XYZ company in Pittsburgh
edidest(testdun1) account(ieacct1) userid(ieuser1);
```

Code examples

The following scenarios and code examples illustrate how companies can use Expedite for Windows. These examples may help you implement this application in your company.

Scenario 1

Company A is an insurance agency that sells policies and processes claims. At the end of each business day, Company A runs an application program to collect information about the day's transactions and prepare the information for transmission to the home office.

The program writes all claim information to the file **claims.fil** and writes all policy information to the file **policy.fil**. Every night the company sends these two files to the home office, and the home office sends the agency updated claim information in the file **update.fil**.

Company A needs to do the following:

1. Create a dropoff box with the recovery level set to checkpoint level and automated recovery set to cancel. Process an error if one occurs:

```
struct ExpAddDropBoxReqStruct aReq;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddDropBoxReqStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.DropBoxName,"Dropbox");
ExpAddField(aReq.IeMboxIdName,"account userid");
ExpAddField(aReq.DropRecovery,"C");
ExpAddField(aReq.SessRecovery,"C");

rc = ExpAddDropBox(&aReq,NULL);

/* Process error if one occurred.*/
if (rc != 0)
  {
  struct ErrorTextRspStruct ErrRsp;
  ExpResetMem((char*)&ErrRsp,sizeof(struct ErrTextRspStruct));
  ExpErrorExplanation(&ErrRsp);
  }
```

2. Create a send order to send the claims file:

```
struct ExpAddSendOrdrReqStruct aReq;
struct ExpSendOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddSendOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpSendOrdrAddedRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"Claims");
ExpAddField(aReq.IeMboxId,"account userid");
ExpAddField(aReq.Fileid,"c:\\data\\claims.fil");

rc = ExpAddSendOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

3. Create a send order to send the policies file:

```
struct ExpAddSendOrdrReqStruct aReq;
struct ExpSendOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddSendOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpSendOrdrAddedRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"Policies");
ExpAddField(aReq.IeMboxId,"account userid");
ExpAddField(aReq.Fileid,"c:\\data\\policies.fil");

rc = ExpAddSendOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

4. Create a receive order to receive the updates file:

```
struct ExpAddRecvOrdrReqStruct aReq;
struct ExpRecvOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddRecvOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpRecvOrdrAddedRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"Updates");
ExpAddField(aReq.IeMboxId,"account userid");
ExpAddField(aReq.Fileid,"c:\\data\\update.fil");

rc = ExpAddRecvOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

5. Create a receive order to receive the error messages:

```
struct ExpAddRecvOrdrReqStruct aReq;
struct ExpRecvOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddRecvOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpRecvOrdrAddedRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"Errors");
ExpAddField(aReq.IeMboxId,"*SYSTEM**ERRMSG*");
ExpAddField(aReq.Fileid,"c:\\data\\errors.fil");


rc = ExpAddRecvOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

6. Assign the send orders to the dropoff box:

```
struct ExpAsgSendOrdrToDropBoxReqStruct aReq;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct  ExpAsgSendOrdrToDropBoxReqStruct ) );


ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.DropBoxName,"Dropbox");
ExpAddField(aReq.OrdrName,"Claims");

rc = ExpAsgSendOrdrToDropBox(&aReq);
```

7. Assign the receive orders to the dropoff box:

```
struct ExpAsgRecvOrdrToDropBoxReqStruct aReq;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct  ExpAsgRecvOrdrToDropBoxReqStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.DropBoxName,"Dropbox");
ExpAddField(aReq.OrdrName,"Updates");

rc = ExpAsgRecvOrdrToDropBox(&aReq);
```

8. Process the dropoff box and check the session receipt. Print the error text if there was an error:

```
struct ExpProcDropBoxReqStruct aReq;
struct ExpDropBoxProcRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpProcDropBoxReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpDropBoxProcRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.DropBoxName,"Dropbox");

rc = ExpProcDropBox(&aReq,&aRsp);
/* Process error if one occurred.*/

printf("Session receipt:\n");
printf("Session receipt name:  %20.20s\n",aRsp.SrcptName);
printf("Dropoff box name:      %20.20s\n",aRsp.DropBoxName);
printf("Start date:            %14.14s\n",aRsp.StartDate);
printf("End date:              %14.14s\n",aRsp.EndDate);
printf("Session key:           %8.8s\n",aRsp.Sesskey);
printf("Ieversion:             %2.2s\n",aRsp.Ieversion);
printf("Last session:          %5.5s\n",aRsp.LastSess);
printf("IE response code:      %5.5s\n",aRsp.Ierespcode);
```

```
/* If there was an error, load the error response structure and print
 * the error text.
 */
if (rc != 0)
   {
   struct ErrorTextRspStruct ErrRsp;
   ExpResetMem((char*)&ErrRsp,sizeof(struct ErrTextRspStruct));
   ExpErrorExplanation(&ErrRsp);

   printf("Error # = %s\n",ErrRsp.ReasonCode);
   printf("Error = %s\n",ErrRsp.ErrorMsg);
   printf("Explanation = %s\n",ErrRsp.ErrorTxt);
   printf("Response = %s\n",ErrRsp.Response);
   }
```

Scenario 2

Company B is an engineering firm that sends files containing CAD/CAM drawings, which are in a special data format that must be treated like binary data. Company B enters functions to send the file **cadcam.fil** to account **abcd** and user ID **engin1** with the message class of **drawing**. Using the DataType field, Company B indicates that Expedite for Windows should treat the data in this file as binary data (**B**), which does not get translated to EBCDIC when it is sent to Information Exchange.

In addition to the **cadcam.fil** file, Company B is also sending the e-mail file **email.fil** to the same user. The value **Y** in the Format field tells Expedite for Windows to send the file to Information Exchange with a message class of **FFMSG001**. The program stores the e-mail messages in **email.fil**.

Company B needs to do the following:

1. Create a send order to send CAD/CAM files as binary data:

```
struct ExpAddSendOrdrReqStruct aReq;
struct ExpSendOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddSendOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpSendOrdrAddedRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"CadCam");
ExpAddField(aReq.IeMboxId,"abcd    engin1");
ExpAddField(aReq.Fileid,"c:\\data\\cadcam.fil");
ExpAddField(aReq.Datatype,"B");
ExpAddField(aReq.MsgClass,"drawing");

rc = ExpAddSendOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

2. Create a send order to send an e-mail file:

```
struct ExpAddSendOrdrReqStruct aReq;
struct ExpSendOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddSendOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpSendOrdrAddedRspStruct) );
```

```
ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"Email");
ExpAddField(aReq.IeMboxId,"abcd    engin1");
ExpAddField(aReq.Fileid,"c:\\data\\email.fil");
ExpAddField(aReq.Format,"Y");

rc = ExpAddSendOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

Scenario 3

Company C is a manufacturer that sells parts to hardware stores. Each day Company C receives orders from hardware stores electronically. All parts orders in the company mailbox have a user class of **parts**. It receives these order files into one file so a clerk can enter the information in the order entry system.

Company C also receives company profiles from new hardware stores. All of these files have the user class **profile**. When Company C receives more than one profile, it receives each profile into separate files.

Company C needs to do the following:

1. Create a receive order to receive all **parts** orders to one file:

```
struct ExpAddRecvOrdrReqStruct aReq;
struct ExpRecvOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddRecvOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpRecvOrdrAddedRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"Parts");
ExpAddField(aReq.IeMboxId,"account userid");
ExpAddField(aReq.Fileid,"c:\\data\\parts.fil");
ExpAddField(aReq.MsgClass,"parts");
ExpAddField(aReq.Multfiles,"N");

rc = ExpAddRecvOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

2. Create a receive order to receive and store in the Information Exchange archive file:

```
struct ExpAddRecvOrdrReqStruct aReq;
struct ExpRecvOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddRecvOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpRecvOrdrAddedRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"Archive");
ExpAddField(aReq.Fileid,"c:\\data\\archive.fil");
ExpAddField(aReq.Archiveid,"archivenum");

rc = ExpAddRecvOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

3. Create a receive order to receive company profiles:

```
struct ExpAddRecvOrdrReqStruct aReq;
struct ExpRecvOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddRecvOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpRecvOrdrAddedRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"Profiles");
ExpAddField(aReq.Fileid,"c:\\data\\profile.fil");
ExpAddField(aReq.MsgClass,""profile");
ExpAddField(aReq.Multfiles,"Y");

rc = ExpAddRecvOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

Scenario 4

Company D is a parts manufacturer that receives its software from a vendor. The vendor is responsible for setting up the company's PCs, installing all software, and providing code updates.

The vendor sends the code update file **updates.fil** to the company's Information Exchange mailbox, and Company D receives the file using the OrigFile field. This field tells Expedite for Windows to receive the updates in a file using the file name from the sending system.

The original file name from the sending system is in the CDH. Therefore, both the vendor and Company D must be using versions of the Expedite for Windows products that support the CDH.

NOTE: When Company D uses the OrigFile field, it must use the FileId field in case the file does not have a CDH.

Company D needs to do the following to create a Receive order to receive updates with their original filename:

```
struct ExpAddRecvOrdrReqStruct aReq;
struct ExpRecvOrdrAddedRspStruct aRsp;
int rc = 0;

ExpResetMem( (char*)&aReq,sizeof(struct ExpAddRecvOrdrReqStruct) );
ExpResetMem( (char*)&aRsp,sizeof(struct ExpRecvOrdrAddedRspStruct) );

ExpAddField(aReq.ProjName,"Project");
ExpAddField(aReq.OrdrName,"Updates");
ExpAddField(aReq.Fileid,"c:\\data\\updates.fil");
ExpAddField(aReq.Origfile,"Y");

rc = ExpAddRecvOrdr(&aReq,&aRsp);
/* Process error if one occurred.*/
```

Scenario 5

Company E needs to list all the addresses and nicknames in the address book. Because there are many addresses in the database, the addresses must be retrieved a few at a time.

Company E needs to do the following to list multiple addresses:

```
  #define MAX_LIST 15;

  struct ExpListReqStruct aListReq;
  struct ExpAddrListedRspStruct aListRsp [MAX_LIST];
  unsigned short NumRead = 0;
  int rc = 0;
  int index = 1;
  char indexStr [10];
  char numEntStr [10];

  sprintf(indexStr,"%d",index);
  sprintf(numEntStr,"%d",MAX_LIST);

  ExpResetMem((char*)&aListReq,sizeof(struct ExpListReqStruct) );
  ExpResetMem((char*)&aListRsp,(MAX_LIST *
                    sizeof(struct ExpAddrListedRspStruct)) );
  ExpAddField(aListReq.ProjName,"Project");
  ExpAddField(aListReq.NumEntries,numEntStr);
  ExpAddField(aListReq.StartIndex,indexStr);

  do
    {
    rc = ExpListAddr(&aListReq,&NumRead,aListRsp);
    /* Process error if one occurred.*/
    if (rc == 0)
      {
      int i;
      for(i=0;i<NumRead;i++)
        {
        char nick›EXP_MAXLEN_AD_NICKNAME+1|;
        memset(nick,'\0',(EXP_MAXLEN_AD_NICKNAME+1));
        memcpy(nick,aListRsp[i].Nickname,EXP_MAXLEN_AD_NICKNAME);
        printf("Address nickname #%d:  %s\n",(index + i),nick);
        }
      }
    index += NumRead;
    sprintf(indexStr,"%d",index);
ExpReplaceField(aListReq.StartIndex,EXP_MAXLEN_START_INDEX,indexStr);
    }
  while ((NumRead >= MAX_LIST) && (rc == 0));
```

Scenario 6

Company F needs to list all the orders on the order shelf. Because the order shelf contains a variety of order types, the type must be defined before processing.

Company F needs to do the following to list multiple types of objects:

```
/* List all orders on the order shelf.*/
struct ExpListReqStruct aListReq;
struct ExpSendOrdrListedRspStruct *aSendOrdr;
struct ExpRecvOrdrListedRspStruct *aRecvOrdr;
char ordrname [130+1];
char type[EXP_MAXLEN_DR_ORDR_TYPE + 1];
char *aListRsp;
unsigned short NumRead = 0;
int rc = 0;
int ordptr = 0;

aListRsp = (char *)malloc(MAX_LIST *
             sizeof(struct ExpRecvOrdrListedRspStruct));

ExpResetMem((char*)&aListReq,sizeof(struct ExpListReqStruct) );
ExpResetMem(aListRsp,(MAX_LIST *
             sizeof(struct ExpRecvOrdrListedRspStruct)));

ExpAddField(aListReq.ProjName,"Project");

printf("ExListOrdersOnOrdrShlf = \n");
rc = ExpListOrdrsOnOrdrShlf(&aListReq,&NumRead,aListRsp);
printf("Number of Orders on the order shelf is: %d\n",NumRead);
/* Process error if one occurred.*/
if (rc == 0)
  {
  int i;
  if (NumRead != 0)
    printf("Here are the order names and types: \n");
  for(i=0;i<NumRead;i++)
    {
    memset(ordrname,'\0',(130+1));
    memset(type,'\0',(EXP_MAXLEN_DR_ORDR_TYPE+1));

    switch(aListRsp[ordptr])
      {
      case EXP_SEND_ORDR_TYPE:
        {
          aSendOrdr = (struct ExpSendOrdrListedRspStruct *)
                   &aListRsp[ordptr];
          memcpy(ordrname,aSendOrdr->OrdrName,EXP_MAXLEN_SN_ORDR_NAME) ;
          memcpy(type,aSendOrdr->Type,EXP_MAXLEN_DR_ORDR_TYPE);
          printf("Order #%d Type :%s: Order Name :%s:\n",(i+1),
                   type,ordrname);
          ordptr += sizeof(struct ExpSendOrdrListedRspStruct);
          }
          break;
        case EXP_RECV_ORDR_TYPE:
          {
          aRecvOrdr = (struct ExpRecvOrdrListedRspStruct *)
               &aListRsp[ordptr];
          memcpy(ordrname,aRecvOrdr->OrdrName,EXP_MAXLEN_RV_ORDR_NAME);
          memcpy(type,aRecvOrdr->Type,EXP_MAXLEN_DR_ORDR_TYPE);
          printf("Order #%d Type :%s: Order Name :%s:\n",(i+1),
               type,ordrname);
          ordptr += sizeof(struct ExpRecvOrdrListedRspStruct);
          }
          break;
        default:
          exit(0);
        }
      }
    }
```

Scenario 7

Company G needs to send a file using Express priority to their CICS-based host doing continuous receiving with Information Exchange. The sender requires an immediate response. If a response is not returned within six minutes and the classification is “urgent,” the request is forwarded to another mailbox to be picked up and processed later.

Company G needs to do the following using interactive sessions:

```
...
struct ExpStartSessReqStruct aSSReq;
struct ExpSessStartedRspStruct aSSRsp;
struct ExpDoSendOrdrReqStruct aDSOReq;
struct ExpSendOrdrDoneRspStruct aSODRsp;
struct ExpDoRecvOrdrReqStruct aDROReq;
struct ExpRecvOrdrDoneRspStruct aRODRsp;
struct ExpEndSessReqStruct aESReq;
struct ExpSessEndedRspStruct aSERsp;
int rc = 0;

/* Start session call establishes an interactive session */
ExpResetMem((char*)&aSSReq,sizeof(struct ExpStartSessReqStruct) );
ExpResetMem((char*)&aSSRsp,sizeof(struct ExpSessStartedRspStruct) );
ExpAddField(aSSReq.ProjName,theProjectName);
rc = ExpStartSess(&aSSReq, &aSSRsp);
if (rc)
  /* Error Handling */

/* Send order with Express priority */
ExpResetMem((char*)&aDSOReq,sizeof(struct ExpDoSendOrdrReqStruct) );
ExpResetMem((char*)&aSODRsp,sizeof(struct ExpSendOrdrDoneRspStruct) );
ExpAddField(aDSOReq.ProjName,theProjectName);
ExpAddField(aDSOReq.IeMboxId,"ATAP    IETESTB");
ExpAddField(aDSOReq.Fileid,"d:\\temp\\testfile.snd");
ExpAddField(aDSOReq.Verify,"N");
ExpAddField(aDSOReq.Charge,"3");
ExpAddField(aDSOReq.Ack,"N");
ExpAddField(aDSOReq.Retention,"0");
ExpAddField(aDSOReq.Priority,"X");
ExpAddField(aDSOReq.Datatype,"T");
ExpAddField(aDSOReq.Format,"N");
rc = ExpDoSendOrdr(&aDSOReq, &aSODRsp);
if (rc)
  /* Error Handling */
/* Issue Receive order to get immediate response with maximum */
/* wait time for 5 minutes                                      */
ExpResetMem((char*)&aDROReq,sizeof(struct ExpDoRecvOrdrReqStruct) );
ExpResetMem((char*)&aRODRsp,sizeof(struct ExpRecvOrdrDoneRspStruct) );
ExpAddField(aDROReq.ProjName,theProjectName);
ExpAddField(aDROReq.IeMboxId,"ATAP    IETESTB");
ExpAddField(aDROReq.Fileid,"d:\\temp\\testfile.rcv");
ExpAddField(aDROReq.Allfiles,"N");
ExpAddField(aDROReq.Multfiles,"Y");
ExpAddField(aDROReq.Origfile,"N");
ExpAddField(aDROReq.Waittime,"5");
rc = ExpDoRecvOrdr(&aDROReq, &aRODRsp);
if (rc)
  /* Error Handling */

/* After 5 minutes, if the receive receipt shows 0 files received, */
/* send order to another mailbox.                                   */
if ( atoi(aRODRsp.FilesRcvd) == 0 )
  {
  ...
  ExpReplaceField(aDSOReq.IeMboxId,EXP_MAXLEN_DS_IE_MBOX_ID,
             "ATAP    IETESTA");
  ...
  ExpReplaceField(aDSOReq.Priority,EXP_MAXLEN_DS_PRIORITY,"N");
  rc = ExpDoSendOrdr(&aDSOReq, &aSODRsp);
  if (rc)
    /* Error Handling */
  }

/* End the intereactive session */
ExpResetMem((char*)&aESReq,sizeof(struct ExpEndSessReqStruct) );
ExpResetMem((char*)&aSERsp,sizeof(struct ExpSessEndedRspStruct) );
ExpAddField(aESReq.ProjName,theProjectName);
rc = ExpEndSess(&aESReq, &aSERsp);
if (rc)
  /* Error Handling */
```

Scenario 8

Company H needs to send files to a trading partner and receive files from headquarters. If the headquarters’ files are received, they need to be forwarded to another application for processing. When that application completes processing, the files need to be forwarded to another Information Exchange mailbox for a different company’s use.

Company H needs to do the following, using interactive sessions:

```
...
struct ExpStartSessReqStruct aSSReq;
struct ExpSessStartedRspStruct aSSRsp;
struct ExpDoSendOrdrReqStruct aDSOReq;
struct ExpSendOrdrDoneRspStruct aSODRsp;
struct ExpDoRecvOrdrReqStruct aDROReq;
struct ExpRecvOrdrDoneRspStruct aRODRsp;
struct ExpEndSessReqStruct aESReq;
struct ExpSessEndedRspStruct aSERsp;
int rc = 0;

/* Start session call establishes an interactive session */
ExpResetMem((char*)&aSSReq,sizeof(struct ExpStartSessReqStruct) );
ExpResetMem((char*)&aSSRsp,sizeof(struct ExpSessStartedRspStruct) );
ExpAddField(aSSReq.ProjName,theProjectName);
rc = ExpStartSess(&aSSReq, &aSSRsp);
if (rc)
  /* Error Handling */

/* Issue send order for testfile.snd*/
ExpResetMem((char*)&aDSOReq,sizeof(struct ExpDoSendOrdrReqStruct) );
ExpResetMem((char*)&aSODRsp,sizeof(struct ExpSendOrdrDoneRspStruct) );
ExpAddField(aDSOReq.ProjName,theProjectName);
ExpAddField(aDSOReq.IeMboxId,"ATAP    IETESTB");
ExpAddField(aDSOReq.Fileid,"d:\\temp\\testfile.snd");
ExpAddField(aDSOReq.Verify,"N");
ExpAddField(aDSOReq.Charge,"3");
ExpAddField(aDSOReq.Ack,"N");
ExpAddField(aDSOReq.Retention,"0");
ExpAddField(aDSOReq.Priority,"X");
ExpAddField(aDSOReq.Datatype,"T");
ExpAddField(aDSOReq.Format,"N");
rc = ExpDoSendOrdr(&aDSOReq, &aSODRsp);
if (rc)
  /* Error Handling */
...
/* Issue send order for next file */
ExpReplaceField(aDSOReq.Fileid,"d:\\temp\\nextfile.snd");
rc = ExpDoSendOrdr(&aDSOReq, &aSODRsp);
if (rc)
  /* Error Handling */
...
```

```
/* Issue receive order to receive file from headquarter */
ExpResetMem((char*)&aDROReq,sizeof(struct ExpDoRecvOrdrReqStruct) );
ExpResetMem((char*)&aRODRsp,sizeof(struct ExpRecvOrdrDoneRspStruct) );
ExpAddField(aDROReq.ProjName,theProjectName);
ExpAddField(aDROReq.IeMboxId,"ATAP    IETESTB");
ExpAddField(aDROReq.Fileid,"d:\\temp\\testfile.rcv");
ExpAddField(aDROReq.Allfiles,"N");
ExpAddField(aDROReq.Multfiles,"Y");
ExpAddField(aDROReq.Origfile,"N");
rc = ExpDoRecvOrdr(&aDROReq, &aRODRsp);
if (rc)
  /* Error Handling */
...

/* Issue receive order for next file */
ExpReplaceField(aDROReq.Fileid,"d:\\temp\\nextfile.rcv");
rc = ExpDoRecvOrdr(&aDROReq, &aRODRsp);
if (rc)
  /* Error Handling */
...

/* Forward file to another application for processing. */
...

/* When processing is complete, forward to Information Exchange */
/* mailbox for another company to use                           */
ExpResetMem((char*)&aDSOReq,sizeof(struct ExpDoSendOrdrReqStruct) );
ExpResetMem((char*)&aSODRsp,sizeof(struct ExpSendOrdrDoneRspStruct) );
ExpAddField(aDSOReq.ProjName,theProjectName);
ExpAddField(aDSOReq.IeMboxId,"ATAP    IETESTA");
ExpAddField(aDSOReq.Fileid,"d:\\temp\\testfile.snd");
ExpAddField(aDSOReq.Verify,"N");
ExpAddField(aDSOReq.Charge,"3");
ExpAddField(aDSOReq.Ack,"N");
ExpAddField(aDSOReq.Retention,"0");
ExpAddField(aDSOReq.Priority,"N");
ExpAddField(aDSOReq.Datatype,"T");
ExpAddField(aDSOReq.Format,"N");
rc = ExpDoSendOrdr(&aDSOReq, &aSODRsp);
if (rc)
  /* Error Handling */

/* End the interactive session */
ExpResetMem((char*)&aESReq,sizeof(struct ExpEndSessReqStruct) );
ExpResetMem((char*)&aSERsp,sizeof(struct ExpSessEndedRspStruct) );
ExpAddField(aESReq.ProjName,theProjectName);
rc = ExpEndSess(&aESReq, &aSERsp);
if (rc)
  /* Error Handling */
```

Scenario 9

Company I wants to display the contents of the Information Exchange mailbox and allow the user to select items to receive. This example shows how to use the **ExpDoQuryOrdr** function and process the results to list the contents of the Information Exchange mailbox.

Company I needs to do the following:

```
struct ExpListItemsReqStruct aListReq;
struct ExpQuryOrdrDoneRspStruct aListRsp[MAX_LIST];

char indexStr[10];
char numEntStr[10];
unsigned short NumRead = 0;
int rc = 0;
int index = 1;
int i;
struct ExpQuryOrdrDoneRspStruct* msg;

sprintf(indexStr, “%d”, index);
sprintf(numEntStr, “%d”, MAX_LIST);

ExpResetMem((char*)&aListReq,
            sizeof(struct ExpListItemsReqStruct) );
ExpResetMem((char*)&aListRsp,
             MAX_LIST *
             sizeof(struct ExpQuryOrdrDoneRspStruct)) );

ExpAddField(aListReq.ProjName, theProjectName);
ExpAddField(aListReq.NumEntries, numEntStr);
ExpAddField(aListReq.StartIndex, indexStr);

do
  {
  ExpResetMem((char*)&aListRsp,
              (MAX_LIST *
               sizeof(struct ExpQuryOrdrDoneRspStruct)) );
  rc = ExpDoQuryOrdr(&aListReq, &NumRead, aListRsp);
  GetError(“ExpDoQuryOrdr”, rc);
  if (rc == 0)
    {
    for(i=0;i<NumRead;i++)
      {
      msg = &(aListRsp[i]);
      /* display the data */
      ...
      }
    }
```

```
     index += NumRead;
     sprintf(indexStr, "%d", index);



ExpReplaceField(aListReq.
                    StartIndex,
                    EXP_MAXLEN_CF_START_INDEX,
                    indexStr);
     }
  while ((NumRead >= MAX_LIST) && (rc == 0));
```

Glossary

A

account. A unique identifier assigned to a group of users who work for the same company.

account ID. A name that identifies an account to a program, device, or system.

account profile. Data that describes the characteristics of a user or group of users.

acknowledgment. A response from Information Exchange that confirms whether files were delivered, received, purged, or any combination of these actions.

address. (1) A unique code assigned to a user connected to a network. (2) The location in the storage of a computer where data is stored.

address book. A repository for trading-partner information. The address book can contain nicknames (for Information Exchange addresses) and trading profiles.

alphanumeric. Pertaining to data that consists of letters, digits, and usually other characters, such as punctuation marks.

alias name. An alternate name used in place of an account and user ID. This is used with an alias table and alias table type.

alias table. A permanent file list of alternate names that resides in Information Exchange.

alias table type. A single character access level indicator used with an alias table name to identify an alias table. Values are G for global access, P for owner access, and O for organizational (account) access.

American National Standard Code for Information Interchange (ASCII). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters.

archive. A place to store messages on a database for future reference.

ASCII. American National Standard Code for Information Interchange.

attribute. A property or characteristic of one or more entities; for example, length, value, color, or intensity.

audit trail. Data, in the form of a logical path, linking a sequence of events used to trace and verify the transactions that have affected the contents of a record.

authorization level. The ability to do certain restricted functions.

API. See application programming interface.

application link. A set of files containing information that Request Manager uses to interact with different applications.

application programming interface. A software interface that enables applications to communicate with each other. An API is the set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by an underlying operating system or service program.

B

binary. Machine instructions that a person cannot read or enter from a computer keyboard.

C

carriage-return and line-feed characters (CRLF). A word processing formatting control that moves the printing or display point to the first position of the next line.

CDH. Common data header.

centralized alias table. Permanent tables that reside in Information Exchange and contain a centralized list of addresses. You can put a list of your trading partners' addresses in this table rather than maintain destination tables in multiple locations.

A centralized alias table enables Information Exchange to resolve destinations because it contains a list of EDI destinations paired with Information Exchange destinations. Expedite for Windows searches this table for an EDI destination and then uses the corresponding Information Exchange destination as the actual address.

character. A letter, digit, or other symbol used as part of the organization, control, or representation of data.

checkpoint-level recovery. A method of restart and recovery within Expedite for Windows. A point where information about the status of a data transmission can be recovered so it can be restarted later.

class. In object-oriented design or programming, a model or template that can be instantiated to create objects with a common definition and, therefore, common properties, operations, and behavior. An object is an instance of a class.

command. A request from a terminal for the performance of an operation or the execution of a particular program.

command line. On a display screen, a display line on which only instructions to the operating system can be entered.

commit. The point in a session at which Expedite and Information Exchange record checkpoint information, such as number of characters and files transmitted so far, and other information needed to recover a session. A commit can occur during or after transmitting a file, depending on the checkpoint-level selected. Once a file is completely committed, Information Exchange will deliver the file to the recipient if you are sending, or it will purge the file from your mailbox if you are receiving. If a session fails, all uncommitted files are discarded, and the session can be resumed at the last commit checkpoint.

common data header (CDH). A set of control information about a file, which is sent to Information Exchange by some sending interfaces. When the file is received by the trading partner, the receiving interface can use the information in the CDH.

CRLF. Carriage-return and line-feed characters.

D

default value. A value assumed when no value has been specified.

delivery acknowledgment. A confirmation that Information Exchange generates when a destination user receives a file from an Information Exchange mailbox.

delivery class. Specifies how messages and files are delivered; senders can choose from high-priority, normal-priority, and express delivery.

distribution list. A list of the addresses of users with whom a certain user communicates. It is used to send messages to several people without having to type their addresses.

dropoff box. The dropoff box represents an Information Exchange session. The dropoff box definition contains information about how Expedite should conduct the session with Information Exchange, such as recovery level, account and user ID for the mailbox, and your time zone. Once orders are created, they are assigned to a dropoff box for processing. The dropoff box is similar to the input file and the TRANSMIT and SESSION profile commands in Expedite Base.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

EDI. Electronic data interchange.

electronic data interchange (EDI). The process of sending specially formatted business documents directly from one computer to another electronically.

electronic mail (e-mail). Free formatted messages and formatted file correspondence sent from one computer to another.

EOF (end of file). A coded character recorded in a file to indicated the end of the file.

express-priority messages. Messages that are delivered immediately after they are received by Information Exchange; the recipient must be receiving messages to receive an express-priority message. If the receiver is not receiving messages, the file is discarded.

extended security option (ESO). An option that extended security users can specify in their profiles for stricter password security.

extended security users. Users with stricter security requirements, such as levels of password protection.

F

field. An area of a panel reserved for data of a certain type or length.

file. A named set of records stored or processed as a unit.

file-level recovery. A method of restart and recovery within Expedite for Windows; checkpoints are taken for each file sent and received.

G

global alias. An alias name that can be used by any Information Exchange user.

global alias table. (1) A system-wide alias table. (2) An alternative name table setup within a system.

H

high-priority message. Messages that move to the front of the queue when they are received; normal-priority messages enter the queue in the order.

I

Information Exchange. A communication service that allows users to send and receive information electronically.

Information Exchange Administration Services. An online, panel-driven product that the Information Exchange Service Administrator uses to perform administrative tasks for Information Exchange.

Information Exchange Service Administrator. The person who coordinates the use of Information Exchange in a company.

J

Java. An object-oriented programming language for portable interpretive code that supports interaction among remote objects. Java was developed and specified by Sun Microsystems, Incorporated.

Java Development Kit (JDK). A software package that can be used to write, compile, debug, and run Java applets and applications.

L

library. A place to store information for an extended period of time. A library consists of a collection of files called library members.

library member. A named collection of records or statements in a library.

M

mailbox. (1) A database that contains records that represent orders and receipts for processed orders.

(1) Any piece of data that users send or receive. (2) The smallest subdivision of information that can be sent from one user to another. (3) An

instruction or explanation on the screen that describes what the system is doing or warns that the system has detected an error.

message acknowledgment. A response from Information Exchange that lets users know whether their messages were delivered, received, purged, or various combinations of the three.

message class. A category, agreed upon by trading partners, that is used to group mail.

message group. A collection of messages that is treated as a single entity. A file is an example of a message group.

message header. The leading part of a message that contains information, such as the source or destination code of the message.

method. In object-oriented design or programming, the software that implements the behavior specified by an operation.

N

National Institute of Standards and Technology (NIST). In the United States, this was formerly the National Bureau of Standards.

non EDI data. Rules defined by NIST to enable X.400 users to exchange binary files through the 1984 X.400 InterPersonal Messaging Services.

O

object. In object-oriented design or programming, a concrete realization of a class that consists of data and the operations associated with that data.

order. An Information Exchange request. The order object represents the Expedite Base commands in Expedite for Windows.

order shelf. When you create an order, Expedite stores the order as part of a set in the mailbox database, known as the order shelf.

order receipt. Expedite for Windows provides a confirmation of the results of order processing that corresponds to a record in the output file of Expedite.

organizational alias table. An alias table setup within an account.

P

parameter. (1) A variable that is given a constant value for a specified application and that may denote the application. (2) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted. (3) Data passed between programs or procedures.

password. A combination of confidential characters that users enter when they log on, to prevent unauthorized access of their systems and data.

permanent distribution list. A distribution list stored permanently in Information Exchange.

private alias. An alias that can be used only by the user who created it.

private alias table. An alias table setup for an individual user.

projects. A naming mechanism that allows multiple users or applications (one at a time) access to the same copy of Expedite for Windows.

R

receipt acknowledgment. A confirmation generated by Information Exchange when a file reaches the receiver's mailbox after a successful session.

receipt shelf. When Expedite creates a receipt for order processing, it stores the receipt as part of a set in the mailbox database, known as the receipt shelf.

receiver. The user or users to whose mailboxes you are sending or retrieving information.

receive-side charges. The charges that users incur when they receive messages through Information Exchange

Request Manager. The component of Expedite for Windows that manages the databases and Information Exchange sessions.

reset. (1) A type of session recovery that indicates Expedite should mark unprocessed orders as pending and enable the mailbox ID. When the session is resumed, Expedite starts processing at the beginning of the first order marked pending. Note that if you were sending multiple EDI envelopes from a single file, and the session is interrupted, if you reset the session, it resumes sending envelopes from the beginning of the file. (2)A session recovery state that indicates the error that interrupted the session prevents the session from being resumed.

restart. (1) A session recovery state that indicates Expedite has recorded commit information during the session and can resume processing at the last checkpoint when the session is resumed. (2) To resume a session at the last checkpoint.

S

send-side charges. The charges that users incur when they send messages through Information Exchange.

session. The period of time during which a user of a terminal can communicate with an interactive system, usually, elapsed time between logon and logoff.

session-level recovery. A method of restart and recovery within Expedite for Windows; no files are committed until the session ends normally.

session receipt. An acknowledgment provided by Information Exchange that corresponds to session start and session end records in Expedite. The session receipt includes a set of order receipts, which provide information about the processing of a single order.

synchronous. A process that is completed within a regular or predictable time frame.

T

temporary distribution list. A distribution list that lasts only for the duration of an Information Exchange session.

trading partner. The business associates with whom users exchange information electronically.

trading partner list. A list of business associates that users can send information to and receive information from using Information Exchange.

trading partner profile. A list that defines which trading partner pays to send or receive messages as part of a set of default information for a trading partner to use to configure send orders.

U

UCS. Uniform Communication Standard.

Uniform Communication Standard (UCS). A standard EDI format used in the grocery industry.

United Nations/Trade Data Interchange (UN/TDI. An EDI standard for administration, commerce, and transportation fields developed by the United Nations Economic Commission for Europe.

UN/TDI. United Nations/Trade Data Interchange.

user ID. A name that identifies a user to Information Exchange within an account.

user message class. A category used to group mail. This category is agreed upon by trading partners.

user profile. A user description that includes account ID, user ID, and password information. The characteristics that designate how a user works with Information Exchange.

W

wildcard. A special character, such as a question mark, that can be used to represent one or more characters for pattern matching.

X

X12. An electronic data interchange standard that defines a specially-formatted EDI data stream, approved by the American National Standards Institute (ANSI).

Index