SYBASE®

Reference Guide

# ECRTP™

Version 4.2

DOCUMENT ID: DC36333-01-0420-01

LAST REVISED: November 2004



Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

# Contents

# About This Book

**Audience**

Map developers are targeted as the primary users of this book. Map developers who use this book to manage maps must also be familiar with the contents of the reference guide. Additional conceptual information and examples are provided in the reference guide to assist the users who configure the system.

**How to use this book**

This document describes how to use ECRTP™ in a Windows environment. ECRTP is a data transformation engine. It analyzes, transforms, and routes messages.

The guide is organized into the following chapters:

- Chapter 1, “About ECRTP” describes how the product works.
- Chapter 2, “Running ECRTP” provides code examples on how to run ECRTP as an executable, DLL, or from a Visual Basic script.
- Chapter 3, “User Exit Routines” provides information on how to invoke a proprietary routine from within a map so that you can perform additional functions called by mapping rules.
- Chapter 4, “Using ECRTP as an Adapter” explains how to use ECRTP as a standalone adapter.
- Chapter 5, “Using ECRTP in a Web Environment” explains how to use ECRTP for your website.
- Chapter 6, “ECRTP Performance” explains how to improve the performance of ECRTP using map caching and memory I/O.
- Chapter 7, “Non-ODBC Database and File Formats” describes the format of trading partner files.
- Chapter 8, “ODBC Database Table Formats” explains how ODBC trading partner data is stored.

**Related documents**

This section describes the available documentation.

**Cross-Platform Documentation** The ECRTP documentation set includes:

- *Installation Guide*

- *Reference Guide*
- *Feature Guide*
- *Release Bulletin*

**Related Documentation** Other related documentation is available from New Era of Networks, Sybase, and IBM. Refer to other documentation from each of these companies for more detail about use of applications relevant to this product.

**Other sources of information**

Use the Sybase Getting Started CD, the SyBooks Bookshelf CD, and the Sybase Product Manuals web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks Bookshelf CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks Bookshelf CD is included with your software. It contains product manuals in a platform-independent bookshelf that contains fully searchable, HTML-based documentation.

  Some documentation is provided in PDF format, which you can access through the PDF directory on the SyBooks Bookshelf CD. To view the PDF files, you need Adobe Acrobat Reader.

  Refer to the *README.txt* file on the SyBooks Bookshelf CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is the online version of the SyBooks Bookshelf CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

**Sybase EBFs and software maintenance**

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the "Technical Support Contact" role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

**Sybase EBFs and software maintenance**

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).

3 Select a product.

4 Specify a time frame and click Go.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

**Conventions**

The formatting conventions used in this manual are:

| Formatting example | To indicate |
|---|---|
| command names and method names | When used in descriptive text, this font indicates keywords such as:<br>• Command names used in descriptive text<br>• C++ and Java method or class names used in descriptive text<br>• Java package names used in descriptive text |

| Formatting example | To indicate |
|---|---|
| | Italic font indicates: |
| *myCounter* variable | • Program variables |
| *Server*`.log` | • Parts of input text that must be substituted |
| *myfile.txt* | • Directory and file names. |
| *User Guide* | • Book titles |
| *sybase/bin* | A forward slash ("/") indicates generic directory information. A backslash ("\") applies to Windows users only.<br>Directory names appearing in text display in lowercase unless the system is case sensitive. |
| "About This Book" | References to chapter titles have initial caps and are enclosed within quotation marks. |
| File > Save | Menu names and menu items are displayed in plain text. The angle bracket indicates how to navigate menu selections, such as from the File menu to the Save option. |
| | The vertical bar indicates: |
| `parse\|put\|get` | • Options available within code |
| `Name\|Address` | • Delimiter within message examples |
| | Monospace font indicates: |
| `create table` | • Information that you enter on a command line or as program text. |
| `table created` | • Example output fragments |

**If you need help**

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

# About ECRTP



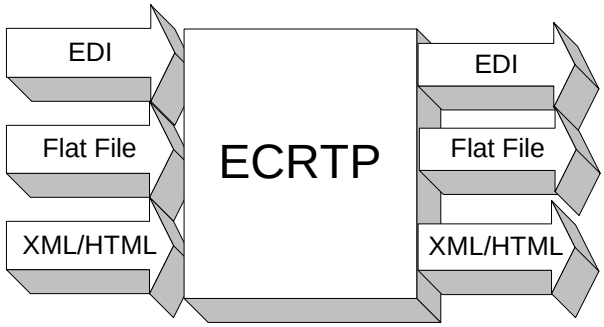
Information included in this chapter includes:

- Map development
- Moving from map development to production
- Executable files and DLLs
- EDI standards
- Generated map files
- Trading partner files
- Application data
- EDI data
- Log files

# About ECRTP

ECRTP is a data transformation engine. It analyzes, transforms, and routes messages. There are many types of message transformations:

***Figure 1-1: ECRTP flow chart"***




1 EDI to flat file – interprets incoming EDI-formatted data and translates it into a user-defined file format. Performed by inbound maps.

2 EDI to EDI – interprets incoming EDI-formatted data and translates it into another EDI format. Performed by any-to-any maps.

3 EDI to XML/HTML – interprets incoming EDI-formatted data and translates it into XML or HTML-formatted data. Performed by web maps (special any-to-any maps).

4 Flat file to EDI – interprets a user-defined file and translates it to an EDI standard or to other standard message formats. Performed by outbound maps.

5 Flat file to flat file – interprets a user-defined file and translates it into another user-defined format. Performed by any-to-any maps.

6 Flat file to XML/HTML – interprets a user-defined file and translates it into XML or HTML-formatted data. Performed by web maps (special any-to-any maps).

7 XML/HTML to EDI – interprets XML or HTML-formatted data and translates it to an EDI standard or to other standard message formats. Performed by web maps (special any-to-any maps).

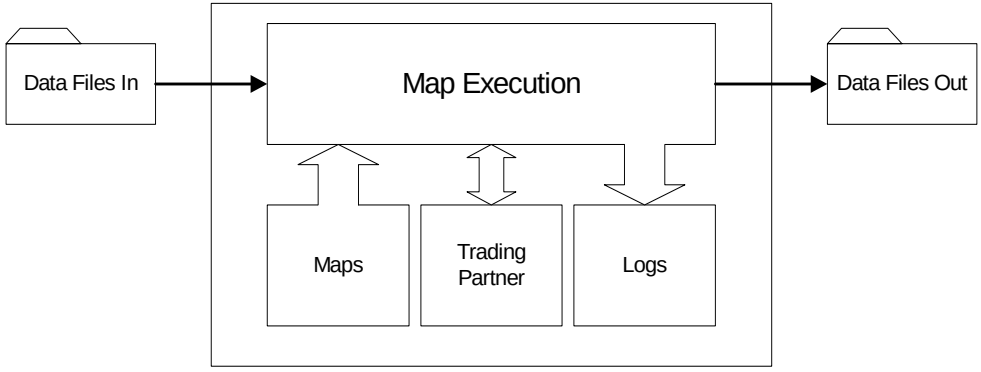
8 XML/HTML to flat file – interprets XML or HTML-formatted data and translates it into a user-defined format. Performed by web maps (special any-to-any maps).

9 XML/HTML to XML/HTML – interprets XML or HTML-formatted data and translates it into XML or HTML-formatted data. Performed by web maps (special any-to-any maps).

ECRTP consists of the following components: map files, trading partner database, and log files.

- The map files contain the business rules and logic that define the relationships between the incoming and outgoing data.
- The trading partner database contains information that is used to route messages between trading partners and to select the specific map that should be run.
- The log files maintain an audit trail of the transaction processing.

***Figure 1-2: Map execution flow chart***




## Map development

The map development program is installed on a client workstation and includes a client ECRTP for execution testing. Maps are developed and tested on the client PC using the map development program and then transferred to the Windows NT server for production. Before transferring map files to the server, it is a good practice to create a backup copy of the files. You can use the Archive or Copy Map functions in ECMap to create a backup.

## Moving from map development to production

The map development program is installed on a client workstation and includes a client ECRTP for execution testing. Maps are developed and tested on the client PC using the map development program and then transferred to the Windows NT server for production. Before transferring map files to the server, it is a good practice to create a backup copy of the files. You can use the Archive or Copy Map functions in ECMap to create a backup.

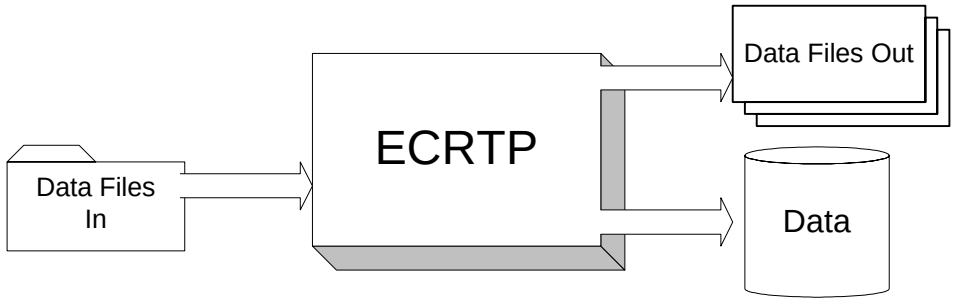
- Map files – copy the ASCII *.*map* files generated by the map development program from the client PC to a "map directory" on the production server. The fully qualified path to this "map directory" (also called the "generated files directory") is used as a switch on the command line of the ECRTP executable or passed in as a parameter to the ECRTP DLL.
- Trading partner database – copy the trading partner database created by the map development program from the client PC to the production server. The location of this database is used as a switch on the command line of the ECRTP executable or passed in as a parameter to the ECRTP DLL. If the trading partner information is stored in a non-ODBC database, the fully qualified path to the "trading partner directory" is used as a switch/parameter. If the trading partner information is stored in an ODBC database, the "trading partner DSN" (that points to the trading partner database) is used as a switch/parameter.

## Production – processing modes

The production ECRTP is installed on an NT server and executes the maps developed with the map development program. ECRTP can be used in a "batch" processing mode or in a "real-time" interactive mode. Illustrations of both modes are shown below:

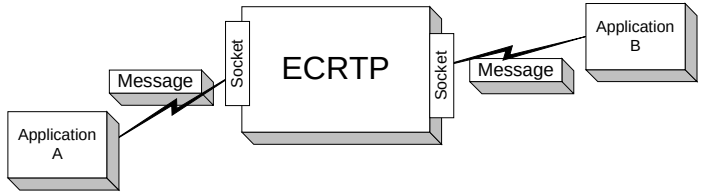
## Batch processing mode

***Figure 1-3: Batch mode processing flow***




## Interactive processing mode

***Figure 1-4: Interactive process mode flow***




ECRTP for Windows can be invoked from a command line as an executable file or called from within a program as a DLL.

There are two executables for the Windows version of ECRTP:

- *wrmi32.exe* for inbound processing
- *wrmo32.exe* for outbound processing

There is one DLL file for the Windows version of ECRTP:

- *owrm32c.dll* for multithreaded processing

This DLL file contains calls to both inbound and outbound runtime functions.

# How ECRTP uses data by location, description, and use

ECRTP uses information stored in files, databases, and memory. The user must know, and provide some of these data locations at runtime. For example, the fully qualified path location of specific data files must be included with certain switches and parameters. Other data locations are known by the program or are not used at runtime. See “Windows runtime parameters/switches” on page 78 for a detailed explanation.

ECMap has a utility available from both the Run Inbound Map and Run Outbound Map screens that allows the user to create a batch file of switches/parameters that correspond to the choices and entries that have been made. When ECRTP is run, this batch file can be used to provide command line parameters or switches, using the -pf switch.

## Executable files and DLLs

The executable files and the DLLs both contain the runtime code. The program uses *Program Files/nnsy/ECRTP* (or */ECMap*) as the default installation location of the executable files and DLLs. The user can install these files in another location during the installation process, but must provide the location of these files at runtime.

## EDI standards

The program uses Program Files/nnsy/Standards as the default installation location of the EDI Standards (X12, EDIFACT, and HL7). The user is not required to know the location of the standards at runtime. This information is required by the map development program during map generation, but not by the runtime program during map execution.

## Generated map files

During map development, the program places map files in a <project name>/<map name> directory structure. When the map files are transferred to the production server, this same directory structure can be kept or the user can specify a different location. The program uses the fully qualified path to the generated map file as part of a required runtime parameter/ switch for inbound and outbound processing. The ECRTP uses this information to dynamically switch maps at runtime, based on the trading partner search option specified by the user at runtime.

- In outbound processing, the map name is the second required switch/parameter and the fully qualified path to the generated map file directory is part of the required -dg (or -dm) switch/parameter. Although a map name is required when the runtime program is invoked, the program does a lookup to see whether a different map should be selected and used. Using the internal trading partner ID information in the incoming application data, ECRTP searches the trading partner database for a record with a matching CUSTNO field. Once it has found a match, ECRTP replaces the current map name with the map name in the trade agreement table (TBCODE field) if it is different from the current map.
- In inbound processing, the fully qualified path to the generated map file directory is part of the required -dg (or -dm) switch/parameter. ECRTP uses either the default trading partner lookup criteria or the lookup specified by a parameter passed in (or switch set) at runtime to dynamically select the correct map at runtime.

Non-ODBC users have two trading partner lookup options:

***Table 1-1: Non-ODBC trading partner search criteria and parameters***

| Trading partner search criteria | Switch/parameter |
|---|---|
| group sender | default |
| reverse - group receiver | -er |

ODBC users have fifteen trading partner lookup options:

***Table 1-2: ODBC trading partner search criteria and parameters***

| Trading partner search criteria | Switch/parameter |
|---|---|
| group sender | default |
| interchange sender | -e5 |
| group and interchange sender | -e2 |
| interchange receiver | -e6 |
| group and interchange receiver | -e4 |
| group sender and receiver | -e1 |
| interchange sender and receiver | -e7 |
| group and interchange sender and receiver | -e3 |
| reverse - group receiver code | -er |
| reverse - interchange receiver | -e10 |
| reverse - group and interchange receiver | -e13 |
| reverse - interchange sender | -e11 |
| reverse - group and interchange sender | -e12 |
| reverse - interchange sender and receiver | -e8 |
| reverse - group and interchange sender and receiver | -e9 |

## Trading partner files

The trading partner files store information about trading partners, the entities that exchange electronic business documents. The trading partner files contain information such as the codes that identify the sender and receiver on EDI envelopes and the details of the trade agreements that link specific maps with specific EDI messages/transaction sets for specific trading partners. Trading partner information is stored in the trading partner file/database and the wixset.dat file.

In both inbound and outbound processing, the fully qualified path to the trading partner directory is part of the required -t switch/parameter for users with non-ODBC trading partner files. For users with an ODBC trading partner database, the DSN (data source name) that points to the trading partner database is part of the required -st <DSN> parameter/switch.

## Application data

User-defined application data is the input for outbound transaction maps, and the output for inbound transaction maps. Application data is both the input and the output for any-to-any maps and webmaps. Application data can be formatted as a flat file, an ODBC database, or HTML or XML data in a text file. Application data can be located in a file/database or in memory; if it is in memory, it can be in a memory location or specified as standardin or standardout.

### Outbound processing (application data is the input of the map)

1 In outbound processing, the location of the output application file is specified in the generated map file. The location of the output application file can be overridden at runtime by entering the fully qualified path to the substitute file name with the optional -eo switch/parameter. (In ECMap, this information is entered as the Substitute Input Filename and the Substitute User File Directory on the Option 2 tab of the Run Outbound Map screen.)

2 Input application data can be read from stdin – instead of from disk – using the -mi switch at runtime. (In ECMap, this information is entered in the Standard Input pane on the I/O Redirect tab of the Run Outbound Map screen.)

3 Input application data can be read from a specified memory address – instead of from disk – using the -mp switch at runtime. (In ECMap, this information is entered in the Internal Memory pane on the I/O Redirect tab of the Run Outbound Map screen.)

### Inbound processing (application data is the output of the map)

1 In inbound processing, the location of the input application file is specified in the generated map file. The location of the input application file can be overridden at runtime by entering a different fully qualified path and file name with the optional -ei switch/parameter. (In ECMap, this information is entered as the Substitute Output Filename and the Substitute User File Directory on the Option 2 tab of the Run Inbound Map screen.)

2 Output application data can be written to stdout – instead of to disk – using the -mo switch at runtime. (In ECMap, this information is entered in the Standard Output pane on the I/O Redirect tab of the Run Inbound Map screen.)

3 Output application data can be written to a specified memory address, instead of to disk, using the -mp switch at runtime. (In ECMap, this information is entered in the Internal Memory pane on the I/O Redirect tab of the Run Inbound Map screen.)

4 By default, output application data is appended to the existing data in the application file at runtime. To overwrite the output data in the application data file instead of appending it, use the -w switch at runtime.

## EDI data

EDI data is the input for inbound transaction maps, the output of outbound transaction maps. EDI data is not used in any-to-any maps.

### Outbound processing (EDI file is the output of the map)

1 In outbound processing, the fully qualified path including the EDI output file name is the first required parameter passed to the ECRTP.

2 The output EDI file can be written to stdout, instead of to disk, using the -mo switch at runtime. (In ECMap, this information is entered in the Standard Output pane on the I/O Redirect tab of the Run Outbound Map screen.)

3 The output EDI file can be written to a specified memory address, instead of to disk, using the -mp switch at runtime. (In ECMap, this information is entered in the Internal Memory pane on the I/O Redirect tab of the Run Outbound Map screen.)

4 Using the trading partner and trade agreement mailbox switches, output EDI data can be routed to different locations:

- If the trading partner mailbox is not ignored (-it switch is not set), outbound EDI data can be routed to the following mailboxes – IN (-ri switch), OUT (-ro switch), GOOD (-rg switch), BAD (-rb switch), or OTHER (-rt switch) – by passing in the appropriate Route EDI Type switch to the ECRTP. (The entry in the Route EDI Type textbox on the Option 1 tab of the Run Outbound Map screen.)

- If the trading partner mailbox is not ignored (-it switch is not set) and either there is no trade agreement mailbox or a trade agreement mailbox exists and is ignored (-o switch is set), the outbound EDI data is automatically routed to the OUT mailbox of the trading partner directory, if no Route EDI Type switch has been passed to the ECRTP. For ODBC users, it is placed in a file with the filename NN.EDI (where NN is the run ID number).
- If the trading partner mailbox is ignored (–it switch is set) and either there is no trade agreement mailbox or a trade agreement mailbox exists and is ignored (–o switch is set), the EDI output is placed in the full path filename specified by the first required parameter passed in to the ECRTP
- If the trade agreement mailbox is not ignored (the–o switch is not set), the EDI output is routed to the trade agreement mailbox and filename. However, for ODBC users, if there is a trade agreement mailbox but no filename, the EDI output is routed to a file with the filename NN.EDI (where NN is the run ID number) in the trade agreement mailbox.

## Inbound processing (EDI file is the input to the map)

1. In inbound processing, the fully qualified path including the EDI input file name is the first required parameter passed to the ECRTP.
2. The input EDI file can be written to stdin, instead of to disk, using the -mi switch at runtime. (In ECMap, this information is entered in the Standard Input pane on the I/O Redirect tab of the Run Inbound Map screen.)
3. The input EDI file can be written to a specified memory address, instead of to disk, using the -mp switch at runtime. (In ECMap, this information is entered in the Internal Memory pane on the I/O Redirect tab of the Run Inbound Map screen.)
4. Using the trading partner and trade agreement mailbox switches, output EDI data can be routed to different locations:
   - If the trading partner mailbox is not ignored (-it switch is not set), inbound EDI data can be routed to the following mailboxes - IN (-ri switch), OUT (-ro switch), GOOD (-rg switch), BAD (-rb switch), or OTHER (-rt switch) - by passing in the appropriate Route EDI Type switch to the ECRTP. (The entry in the Route EDI Type textbox on the Option 1 tab of the Run Inbound Map screen.)

- If the trading partner mailbox is not ignored (-it switch is not set) and either there is no trade agreement mailbox or a trade agreement mailbox exists and is ignored (-o switch is set), the inbound EDI data is routed to the IN mailbox of the trading partner directory when no Route EDI Type switch has been passed in to the ECRTP.
- If the trading partner mailbox is ignored (-it switch is set) and either there is no trade agreement mailbox or a trade agreement mailbox exists and is ignored (-o switch is set), the inbound EDI data is not routed.
- If the trade agreement mailbox is not ignored (the -o switch is not set), the EDI input is routed to the IN mailbox of the trade agreement mailbox if it exists, or to the IN mailbox of the trading partner mailbox if the trade agreement mailbox does not exist.
- If the trading partner mailbox is ignored (-it switch is set) and the trade agreement mailbox is not ignored (-o switches is not set), the inbound EDI data is routed to the IN mailbox of the trade agreement mailbox if it exists, and not routed if the trade agreement mailbox does not exist.

## Log files

Non-ODBC log files include the transaction log and the trace file. The ODBC log database includes the transaction log, the trace file, and the run ID table. If a non-ODBC log is used, ECRTP places the log files in the same directory where the executable is installed. If an ODBC log is used, the user must specify the DSN (data source name) that points to the log database using the required –sl <DSN> parameter/switch.

### Non-ODBC log files

#### Non-ODBC transaction log

The transaction logs for non-ODBC users are stored in fixed-length files in the directory where the executable is installed, in a file named *translog.out* for outbound processing and in a file named translog.in for inbound processing. Each time ECRTP is executed, the program appends new information to the transaction log files. Since these files can become very large, the user must institute a procedure to control their size by periodically deleting information.

Non-ODBC users have three choices for the creation of the transaction log:

- No Log
- Non-Expanded Text Transaction Log
- Expanded Text Transaction Log

If the transaction log is written in the expanded format (-xl switch), additional fields are included and the sizes of three fields are expanded to include eight-digit dates, six-digit times, and the complete text of all error messages. The non-expanded format has fewer fields and includes only six-digit dates, four-digit times, and truncated versions of the error messages. See “Expanded format” on page 110 for the layout of the non-ODBC expanded text transaction log.)

#### Non-ODBC trace files

The trace files for non-ODBC users are stored in fixed-length files in the map/generated files directory, in *outgoing.err* for outbound processing and *incoming.err* for inbound processing.

## ODBC log files

#### ODBC transaction log

For both outbound and inbound processing, the ODBC transaction logs are stored in the TRLOG table in the ODBC log database. The format of TRLOG is essentially the same as the format of the non-ODBC expanded text transaction log file - with the exception of the first field. In TRLOG, this field is an auto-increment field. (Section 11 contains the layout for the ODBC transaction log.)

ODBC users have only one choice for the creation of the transaction log:

- ODBC Database Log Table

#### ODBC RunID table

The RunID table contains unique sequential run number associated with each map run. This number is incremented each time that a map is executed, regardless of whether the processing is inbound or outbound.

### ODBC trace files

The ODBC trace files for both inbound and outbound processing are stored in a fixed-length file called TRNN.dat, where NN is the Run ID number for the map run.

CHAPTER 2

# Running ECRTP

# Running ECRTP as an executable

ECRTP can be invoked from a command line, using one of two executable programs. One of the executable programs is used for outbound and any-to-any processing, and the other executable program is used for inbound processing.

- *wrmo32.exe* – executable program for outbound and any-to-any processing
- *wrmi32.exe* – executable program for inbound processing

When you run ECRTP from the command line, there are required switches that must be used, as well as optional switches that can be used. For a description of all available switches, see "Windows runtime parameters/switches" on page 78. When ECRTP is invoked as an executable, the -mi, -mo, and -mm memory I/O parameters can be used, but not -mp or -mx.) The required switches for executables are described below

## Required switches for an outbound executable

The switches shown below are required for outbound processing. You must use the first three switches in the order shown. The other switches have no fixed position, but they must be preceded by the appropriate "-letters" flag. You use either the -st or -sl switch, depending on whether your trading partner information is stored in a non-ODBC or an ODBC database. You are also allowed to use the -dm switch in place of the –dt and –dg switches when the trading partner directory and the generated files directory are the same.

```
wrmo32 <EDI output file> <mapname> <transaction code>
-t<transaction/message> -dt<fullpath trading partner
directory> -dg<fullpath generated files directory>
```

or

```
wrmo32 <EDI output file> <mapname> <transaction code>
-t<transaction/message> -dg<fullpath generated files
directory> -st <trading partner DSN>
```

When the trading partner directory and the generated files directory are the same, the command line is shortened as follows:

```
wrmo32 <EDI output file> <mapname> <transaction code>
-t<transaction/message> -dm<fullpath trading
partner/generated files directory>
```

***Table 2-1: Switch descriptions***

| Switch | Description |
|---|---|
| <EDI output file> | The name of the EDI output file. |
| <map name without extension> | The map name without the extension. |
| <transaction code> | Two-character transaction code. |
| -t<transaction/message> | Transaction set/message.<br>Example:<br>`-t 837` |
| -dt<full path trading partner directory> | The fully qualified path to the trading partner directory. For users with a non-ODBC trading partner database, these switches are required unless the –dm switch is used.<br>Example:<br>`-dt c:\ecdata\tptner` |
| -dg<full path generated files directory> | The fully qualified path to generated files directory. This directory contains the generated (map and cross-reference tables) files. These switches are required unless the -dm directory is used.<br>Example:<br>`-dg c:\ecdata\rtp` |
| -st<trading partner DSN> | The DSN specifies the data source name for the ODBC trading partner database.<br>Example:<br>`-st "MS Access"` |

## Required switches for an inbound executable

The switches shown below are required for inbound processing. The first switch is required and must always be first in the order. The other switches have no fixed position, but they must be preceded by the appropriate “-letters” flag. You use either the -st or –sl switch, depending on whether your trading partner information is stored in a non-ODBC or an ODBC database. You are also allowed to use the -dm switch in place of the –dt and –dg switches when the trading partner directory and the generated files directory are the same.

```
wrmi32 <EDI input file> -dt<fullpath trading partner
directory>

-dg<fullpath generated files directory>
```

or:

```
wrmi32 <EDI input file> -dg<fullpath generated files
directory> -st <trading partner DSN>
```

When the trading partner directory and the generated files directory are the same, the command line is shortened as follows:

```
wrmi32 <EDI input file> -dm<fullpath trading
partner/generated files directory>
```

***Table 2-2: Switch descriptions***

| Switch | Description |
|---|---|
| <EDI input file> | The name of the EDI input file. |
| -dt<full path trading partner directory> | The fully qualified path to the trading partner directory. For users with a non-ODBC trading partner database, these switches are required unless the –dm switch is used.<br>Example:<br>`-dt c:\ecdata\tptner` |
| -dg<full path generated files directory> | The fully qualified path to generated files directory. This directory contains the generated (map and cross-reference tables) files. These switches are required unless the “-dm” directory is used.<br>Example:<br>`-dg c:\ecdata\rtp` |
| -st<trading partner DSN> | The DSN specifies the data source name for the ODBC trading partner database<br>Example:<br>`-st "MS Access"` |

## Environment variables

In addition to command line switches, you must also set the following environment variables listed in Table 2-3, which affect program performance.

In the current trading partner database, there are two levels for storing information, Interchange Level and Transaction Set Level. As ECRTP processes each transaction, a lookup is done on the trading partner file and if a transaction set is not defined in the trade agreement table, no acknowledgement flags are set for that transaction. Therefore, a bad transaction set id error cannot be reported in the 997 Acknowledgement transaction that is transmitted back to the originator of the interchange. If there was an option in the Trading Partner database to set acknowledgement flags at the Functional Group Level instead of the Transaction Set level, the ACK_EXPECT field in the TRLOG database could be set correctly based on the value of the flag at the Functional Group Level.

***Table 2-3: Environment variable descriptions***

| Environment variable | Description |
|---|---|
| ACKINT | ECRTP checks for the existence of the ACKINT environment variable. If this environment variable exists and has been set to any value, then when the trading partner record is not found during an inbound run, the ACK_EXPECT log value will be a ’1’ instead of '0'. |
| ACKGROUP | ECRTP checks for the existence of an ACKGROUP environment variable. If this environment variable exists and has been set to any value, then when the TRADSTAT record is not found during an inbound run, the ACK_EXPECT log value will be a '1' instead of '0'. |
| ALL_TB_OWNERS | ECRTP checks for the existence of an ALL_TB_OWNERS environment variable. If this environment variable exists and has been set to any value, then a table can be accessed by the code as long as the current user has sufficient access permission. |
| AUTO_INC_FIX | For databases that do not support an auto-increment field, setting the AUTO_INC_FIX environment variable provides a mechanism by which the TPKEY (TP table), TRADKEY (TRADSTAT table), and ALFD (TRLOG table) fields are incremented whenever a record is added to the TP, TRADSTAT, and TRLOG tables.<br>For DB2, the sytax for AUTO_INC_FIX is:<br>`auto_inc_fix = NOT NULL GENERATED ALWAYS AS IDENTITY` |
| PADEDI | Pads numeric fields with leading zeros or alphanumeric field with trailing spaces to ensure that data meets minimal field length requirements.<br>In Windows, set the **Variable Name** to `PADEDI`. Set the **Variable Value** to **Y**.<br>In UNIX, open a shell and run the following commands:<br>`ksh: set PADEDI=YES ; export PADEDI`<br>`sh:  set PADEDI=1 ; export PADEDI`<br>`csh: setenv PADEDI 1` |

| Environment variable | Description |
|---|---|
| WWIXTB=(NUMBER) | The number set by wwixtb is the maximum number of records that a cross reference file can have and still be loaded into memory. The default wwixtb value is 10000. This is equivalent to the command line parameter "-r number".Example: Set wwixtb=200<br>This environment variable is only available on NT and Window Platforms. For UNIX, the default is 10000, and the<br>–r switch must be used to change maximum value. |
| WWIXQUOTE | This optional delimited file environment variable can be set to SPACE or NONE or PIPE or any character. Default delimited file quote is a double quote. |
| WWIXDELIM | This optional delimited file environment variable can be set to SPACE or PIPE (where PIPE is "\|" ) The default delimited file delimiter is a comma. Example: Set WWIQUOTE=' changes delimited file quote to single quote. |
| WWIXNUNG | If set, then no UNG EDIFACT segment is written for outbound maps (same as -u switch). |
| WWIXTRANS | If set to no, no badtrans.nmt is produced for inbound maps (equivalent to the -b command line parameter). |
| WWIXERR | If set, then the inbound control numbers from trading partners are compared with previous inbound control numbers from the same trading partner. If the inbound control numbers are not being incremented by 1 from the previous control number, then an error message is written. (no equivalent switch) |
| WWIXDEBUG | If set to any value, then temporary files created during the processing of "Multiple Files Yes" run will not be deleted for outbound maps. This switch is available to help analyze/debug map problems (same as "-db" switch). |
| WWIXNOCR | If set to any value, then a segment delimiter of newline, will be written as newline only. Normally on the PC, a newline segment delimiter is written as <return, newline>. On UNIX, this is not a valid switch because newline segment delimiters are always written as just a newline (no equivalent switch). |

# Running ECRTP as a DLL

There is one DLL file that can be used to run ECRTP: owrm32c.dll. This file is for single-threaded and multithreaded applications, and contains code for both inbound and outbound maps.

The *owrm32c.dll* file has three inbound run API functions and three outbound run API functions.

There are also several API functions that allow you to:

- Load a map into memory
- Free a specific map in memory
- Free all maps in memory

For the *owrm32c.dll file*, there is a corresponding LIB file that can be used to link implicitly to the DLL from a C program or Java program and a DEF file for information only.

The DLL file and its related files are described below.

***Table 2-4: DLL file descriptions***

| DLL file and related files | Description |
|---|---|
| • *owrm32c.dll*<br>• *owrm32c.lib*<br>• *owrm32c.def* | This is a single-threaded and multithreaded Visual C++ compiled DLL with ODBC rule functions. It can be loaded dynamically via an API during runtime. It can also be linked implicitly from a C++ program via *owrm32c.lib*. *owrm32c.def* provides information only.<br><br>When you run *owrm32c.dll*, the result of the run returns codes 1–5:<br>• 1 – correct<br>• 2 – transaction skipped error, – checking trace file (*incoming.err* or *outgoing.err*)<br>• 3 – transaction user abort error – checking trace file (*incoming.err* or *outgoing.err*)<br>• 4 – transaction user stop error – checking trace file (*incoming.err* or *outgoing.err*)<br>• 5 – transaction fatal error P checking trace file (*incoming.err* or *outgoing.err*).<br>If the return code is 2–5, run the map again with long trace set by adding -c and -l at the end of the mapswitch. Or check the long trace in the Run Map screen in ECMap. |
| • *callrtp.dll*<br>• *callrtp.lib* (For Windows NT only) | This DLL contains JNI functions and acts as a wrapper around *owrm32c.dll*. From Windows NT, it can be linked implicitly from a C++ program via *callrtp.dll*. A *callrtp.def* file is not included. |

| DLL file and related files | Description |
|---|---|
| *toolpak.h* | Contains the prototypes for the following six WIN API calls that are available within each DLL:<br>• OUTBOUNDMAPPER<br>• OUTBOUNDRunCmd<br>• OUTRun<br>• INBOUNDMAPPER<br>• INBOUNDRunCmd<br>• INRun<br>There are also four JAVA API functions provided:<br>• JINBOUNDRunCmd<br>• JOUTBOUNDRunCmd<br>• JINRun<br>• JOUTRun |

When you invoke ECRTP as a DLL, there are required parameters that must be used, as well as optional parameters/switches that can be used. For a description of all available switches, see “Windows runtime parameters/switches” on page 78. When ECRTP is invoked as a DLL, all of the memory I/O parameters can be used.

The prototypes for the WINAPI calls (available in toolpak.h) are shown below:

```
typedef struct {
   char *filename;   /* Pointer to name of
                     Directory\Filename-in MAP File */
   char **paddr;     /* Double pointer to memory address
                        of data */
   long *pbytes;     /* Pointer to Number of bytes of data
                        in memory */
   long *pbuflen;   /* Pointer to Number of bytes allocated
                        in memory */} MEMIOSTRUCT;

   extern "C" {
   __declspec( dllimport) int WINAPI  OUTBOUNDRunCmd(char
   *cmd);
   __declspec( dllimport) int WINAPI  INBOUNDRunCmd(char
   *cmd);__declspec( dllimport) int WINAPI
   OUTBOUNDMAPPER(int argc, char **argv);
   __declspec( dllimport) int WINAPI  INBOUNDMAPPER(int
   argc, char **argv);
```

```
__declspec( dllimport) int WINAPI OUTRun(LPSTR,
MEMIOSTRUCT **);
__declspec( dllimport) int WINAPI INRun(LPSTR,
MEMIOSTRUCT **);

/* load map into memory ahead of map running*/
/* note: LOADMAP can be called multiple times to load
many maps*/
/* -map_dir:  0 for inbound , 1 for outbound */
/* -MulMaps: how many maps to save in memory, 1 to N */
/* upper limit N is dependent on memory available */
__declspec( dllimport) int WINAPI LOADMAP(char
*dir_path, char * name, int map_dir, int MulMaps);

/* free specific map in memory*/

__declspec( dllimport) int WINAPI FREEMAPNAME(char
*map_name);

/* clear all maps in memory*/
/* it's calling program's responsibility to call
FREEALLMAP to free all maps in memory if  calling
program does not pass -xf switch to rtp engine. If -xf
switch passes into rtp engine  rtp will  free all maps
in memory at the end of this run */

__declspec( dllimport) void WINAPI FREEALLMAP();}
```

The prototypes for the Java API calls are shown below:

- int JINBOUNDRUNCmd(String Str);
- int JOUTBOUNDRunCmd(String Str);
- int JINRun(String[] starr);
- int JOUTRun(String[] strarr);

## WIN API function calls for outbound processing

The outbound ECRTP functions convert data from a flat file, ODBC database table, or XML/ HTML data to a standard EDI message format. Three API function calls are available for running outbound transaction maps:

```
extern "C" _declspec(dllimport) int WINAPI
OUTBOUNDMAPPER(int argc, char **argv

extern "C" _declspec(dllimport) int WINAPI
```

```
OUTBOUNDRunCmd(char *cmd);

extern "C" _declspec(dllimport) int WINAPI
OUTRun(LPSTR, MEMIOSTRUCT **)
```

The parameters passed to the DLL for outbound processing have the same values as the switches used on the command line of the executable program for outbound maps—*wrmo32.exe*.

- For OUTBOUNDMAPPER, the runtime parameters are passed in using an array. “argc” is the number of cells in the array “argv”. The first “argv[0]” cell contains the function name (OUTBOUNDMAPPER) and the remaining cells contain the runtime parameters in the required order shown below.
- For OUTBOUNDRunCmd, the runtime parameters are passed in as a string, in the required order shown below. “cmd” is the string containing the runtime parameters.
- For OUTRun, the runtime parameters are passed in as a string, in the required order shown below. “LPSTR” is the string containing the runtime parameters. “MEMIOSTRUCT” is a pointer to an array of MEMIOSTRUCT structures that are used to redirect memory I/O files.

## Parameters required for outbound API function calls

The following parameters are required for outbound processing via a DLL:

***Table 2-5: Switch descriptions***

| Switch | Description |
|---|---|
| <EDI output file> | The name of the EDI output file. |
| map name without file extension | Map name without file extension. |
| -t<transaction/message> | Transaction set/message |
| -dt | Required if -st is not used. |
| -dg<full path generated files directory> | The fully qualified path to generated files directory. This directory contains the generated (map and cross-reference tables) files. These switches are required unless the “-dm” directory is used.<br>Example:<br>`-dg c:\ecdata\rtp` |

| Switch | Description |
|---|---|
| -st<trading partner DSN> | The -st option is required if the -dt option is used. The DSN specifies the data source name for the ODBC trading partner database<br>Example:<br>`-st "MS Access"` |

## Syntax for outbound API function calls

The three following syntax options are available for outbound processing via a DLL:

```
ret = OUTBOUNDMAPPER(int argc, char **argv);
```

The required parameters are passed as an array for OUTBOUNDMAPPER.

```
ret = OUTBOUNDRunCmd(char *cmd);
```

The parameters are passed as one command string for OUTBOUNDRunCmd. This string argument can be generated automatically in ECMap by filling in the appropriate the textboxes on the Run Outbound Map screen and clicking the Create Batch button.

```
ret = OUTRun(char *argv[1], MEMIOSTRUCT **ppmystruct);
```

The parameters are passed as one command string for OUTRun. This string argument can be generated automatically in ECMap by filling in the appropriate the textboxes on the Run Outbound Map screen and clicking the Create Batch button.

## Sample programs for using outbound API function calls

A sample setup for the OUTBOUNDMAPPER API call is shown below:

```
int ret, iarg;
char **argpp;
char *argp[12];
argp[0] = "OUTBOUNDMAPPER";
argp[1] = "EDIFILE"; /* full EDI output file name */
argp[2] = "837MAP"; /* file name of initial map */
argp[3] = "PO"; /* transaction code */
argp[4] = "-t"; /* message/transaction set parameter */
argp[5] = "837"; /* message/transaction  set */
argp[6] = "-dt"; /* trading partner directory parameter
*/
```

```
argp[7] = "../tptner"; /* trading partner directory */
argp[8] = "-dg"; /* generated files directory
                    parameter */
argp[9] = "../rtp"; /* generated files (ie. map)
                    directory */
argp[10] = "-st"; /* trading partner DSN parameter */
argp[11] = "MS Access";   /* trading partner DSN */
iarg = 12 ;
argpp = &argp[0];ret  = OUTBOUNDMAPPER(iarg, argpp);
```

A sample setup for the OUTBOUNDRunCmd API call is shown below:

```
 int ret;ret = OUTBOUNDRunCmd("EDIFILE 837MAP PO -t 837
–dt ../tptner –dg ../rtp”);
```

A sample setup for the OUTRun API call is shown below:

```
// memiodemo.cpp
// This is a memory I/O demo program. This is an example
// of an OUTRun call where the input file and output
// file are both memory files.  Thisprogram illustrates
// how to call a map and have that map read a file
// from memory instead of disk.  It also illustrates how
// to call a map so that the map will write a file to
// memory instead of disk. // The construction of second
// parameter, the MEMIOSTRUCT, is the key to passing
// memory files.  The run map switches are passed in as
// the first parameter.
#include <windows.h>
#include <stdio.h>
/* Memory file structure */
typedef struct {
    char *filename;   /* Pointer to name of
Directory\Filename - in MAP File */
    char **paddr;    /* Double pointer to memory address
                        of data */
    long *pbytes;     /* Pointer to Number of bytes of
                        data in memory */
    long *pbuflen;    /* Pointer to Number of bytes
                        allocated in memory */
} MEMIOSTRUCT

extern "C" {
__declspec (dllimport) int WINAPI OUTRun(LPSTR,
MEMIOSTRUCT **);
}
int main(int argc, char* argv[])
{
```

```
   int ret;
char myfile1[100], myfile2[100];
   char *membuf1, membuf2;
   long membytes1, membytes2;
 long membuflen1, membuflen2;
   FILE *fp;
MEMIOSTRUCT mystruct, mystruct2;
MEMIOSTRUCT *ppmystruct[3];
ppmystruct[0] = &mystruct;
ppmystruct[[1] = &mystruct2;
ppmystruct[2] = (MEMIOSTRUCT *)NULL; /* Must be
terminated with a NULL pointer */

   /* Allocate memory for the map input file */
    if((membuf1 = (char *)malloc(100000 * sizeof(char)))
                  == NULL
)      exit( 1 );
   /* Load data into the memory */
   if( (fp = fopen("d:\\tmp\\850out\\data\\
       850_udf.txt", "rb" )) == NULL
)
   {
    exit(1);
   }
    else
   {
    int i = 0;
    char ch;
      while((ch = fgetc(fp)) != EOF)
    membuf1[i++] = ch;
      if(ch == EOF) membuf1[i + 1] = '\0';
      fclose( fp );
   }
   /* Find the number of bytes of valid data */
   membytes1 = strlen(membuf1);
/* The directory and file name are exactly the same as
in the map(case sensitive) */
   strcpy(myfile1, "c:\\test\\data\\850_udf.txt");
    ppmystruct[0]->filename = myfile1;
    ppmystruct[0]->paddr = &membuf1;
    ppmystruct[0]->pbytes = &membytes1;
    ppmystruct[0]->pbuflen= &membuflen1;

/* Set up the MEMIOSTRUCT parameters for the Output
memory File */
/* The directory and file name are exactly the same as
in the map(case sensitive) */
```

```
   strcpy(myfile2, "c:\\test\\data\\850_out.txt");
    ppmystruct[1]->filename = myfile2;
/* For files that are being written to, one can set the
 base char pointerto null, and ECRTP will allocate space
for any output data and place the address of the
allocated space in membuf2. Similarly, if the output
memory was preallocated and the ECRTP needed more space,
the ECRTP would reallocate the space and place any
modified memory address back in membuf2. */

membuf2 = (char *) NULL;

ppmystruct[1]->paddr = &membuf2;

/* set number bytes to output memory written to 0 */
/* Note, if one is starting with allocated space, the
new output would be appended at the value of membytes2
*/

membytes2 = 0L;

ppmystruct[1]->pbytes = &membytes2;

/* Set the number of bytes already allocated for the
output file to zero */

ppmystruct[1]->pbuflen= &membuflen2;

 /* The run map switches are passed in as one parameter
argv[1] */

ret = OUTRun(argv[1], ppmystruct);

 free membuf1;  /* free input file buffer */

 /* Note at this point if the map wrote any data to the
output file,  then membuf2 will point to the memory that
was allocated, membytes2 will contain the number of
bytes written.  */

/* it is the user's responsibility to free the output
file */

if(membuf2 != (char *) NULL)
    free(membuf2);return ret;
}
```

## WIN API function calls for inbound processing

The inbound ECRTP functions convert data from a standard EDI message format to a flat file, ODBC database table, or XML/HTML data. Three API function calls are available for running inbound transaction maps:

```
extern "C" -declspec(dllimport) int WINAPI
INBOUNDMAPPER(int argc, char

extern "C" -declspec(dllimport) int WINAPI
INBOUNDRunCmd(char *cmd);

extern "C" -declspec(dllimport) int WINAPI INRun(LPSTR,
MEMIOSTRUCT**)
```

The parameters passed to the DLL for inbound processing have the same values as the switches used on the command line of the executable program for inbound maps—*wrmi32.exe*.

- For INBOUNDMAPPER, the runtime parameters are passed in using an array. “argc” is the number of cells in the array “argv”. The first “argv[0]” cell is the function name (INBOUNDMAPPER) and the remaining cells contain the runtime parameters in the required order shown below.
- For INBOUNDRunCmd, the runtime parameters are passed in as a string, in the required order shown below. “cmd” is the string containing the runtime parameters.
- For INRun, the runtime parameters are passed in as a string, in the required order shown below. “LPSTR” is the string containing the runtime parameters. “MEMIOSTRUCT” is a pointer to an array of MEMIOSTRUCT structures that are used to redirect memory I/O files.

## Parameters required for inbound API function calls

The following parameters are required for inbound processing via a DLL:

***Table 2-6: Switch descriptions***

| Switch | Description |
|---|---|
| <EDI input file> | The name of the EDI input file. |
| -dt | Required if -st is not used. |
| -dg<full path generated files directory> | The fully qualified path to generated files directory. This directory contains the generated (map and cross-reference tables) files. These switches are required unless the "-dm" directory is used.<br>Example:<br>`-dg c:\ecdata\rtp` |
| -st<trading partner DSN> | The -st option is required if the -dt option is used. The DSN specifies the data source name for the ODBC trading partner database<br>Example:<br>`-st "MS Access"` |

## Syntax for inbound API function calls

The three following syntax options are available for inbound processing via a DLL:

```
ret = INBOUNDMAPPER(int argc, char **argv);
```

The required parameters are passed as an array for INBOUNDMAPPER.

```
ret = INBOUNDRunCmd(char *cmd);
```

The parameters are passed as one command string for INBOUNDRunCmd. This string argument can be generated automatically in ECMap by filling in the appropriate the textboxes on the Run Inbound Map screen and clicking the Create Batch button.

```
ret = INRun(char *argv[1], MEMIOSTRUCT **ppmystruct);
```

The parameters are passed as one command string for InRun. This string argument can be generated automatically in ECMap by filling in the appropriate the textboxes on the Run Inbound Map screen and clicking the Create Batch button.

## Sample programs for using inbound API function calls

A sample setup for the INBOUNDMAPPER API call is shown below:

```
int ret, iarg;char **argpp;char *argp[8];
```

```
argp[0] = "INBOUNDMAPPER";
argp[1] = "../data/EDIFILE"; /* full EDI input file
                        name */
argp[2] = "-dt";        /* trading partner directory
                         parameter */
argp[3] = "../tptner"; /* trading partner directory */
argp[4] = "-dg";        /* generated files directory
                         parameter */
argp[5] = "../rtp"; /* generated files directory */
argp[6] = “-st”;    /* trading partner DSN parameter */
argp[7] = “MS Access”; /* trading partner DSN */
iarg = 8;
argpp = &argp[0];
    ret = INBOUNDMAPPER(iarg, argpp);
```

A sample setup for the INBOUNDRunCmd API call is shown below:

```
int ret;
ret = INBOUNDRunCmd( "../data/EDIFILE -dt ../tptner -dg
../rtp “);
```

A sample setup for the INRun API call is shown below:

```
int ret;
MEMIOSTRUCT mystruct[2];
MEMIOSTRUCT *pmystruct=&mystruct[0];
MEMIOSTRUCT *ppmystruct[2];
ppmystruct[0]=pmystruct;
ppmystruct[1]=(MEMIOSTRUCT*)NULL;
    ret = INRun( "../data/EDIFILE -dt ../tptner -dg
../rtp “;ppmystruct);
```

## Using Java to execute ECRTP

You can use Java in two ways to execute ECRTP:

- Using Java API calls (requires C++ programming and is very flexible)
- Using a Java package (does not require C++ programming but is less flexible)

First, “Using Java API calls to execute ECRTP” on page 34 is described below followed by “Using a Java package to execute ECRTP” on page 38.

## Using Java API calls to execute ECRTP

This section describes how to use a Java program to call the Java-callable version of the ECRTP. CALLRTP.DLL was created using jni (Jave Native Interface) and OWRM32C.DLL contains ECRTP C/C++ API calls. Both are compiled as fully reentrant, multi-threaded .dlls using VC++. The Java classes MYAPP and RTP call CALLRTP.DLL. CALLRTP.DLL is a wrapper around OWRM32C.DLL and were compiled using JDK 1.3, as follows:

- *rtp.class* was created by javac *RTP.java*
- *MYAPP.class* was created by javac *MYAPP.java*

The Java program calls *RTP.java* and passes in two parameters:

- 1, 2, 3 or 4
- Command string (that contains the parameters outlined earlier under either Parameters Required for Outbound API Function Calls or Parameters Required for Inbound API Function Calls) or a string array for memory I/O API functions.)

Based on the first parameter passed in, the program calls one of four Java functions—JINBOUNDRunCmd, JOUTBOUNDRunCmd, JINRun, or JOUTRun.

- If the first parameter is a “1” for an inbound map, the program calls the JAVA function JINBOUNDRunCmd and passes to it the second parameter to run the inbound map. JINBOUNDRunCmd then calls the API INBOUNDRunCmd and again passes the command string.
- If the first parameter is a “2” for an outbound map, the program calls the JAVA function JOUTBOUNDRunCmd and passes the second parameter to run the outbound map. JOUTBOUNDRunCMD then calls the API OUTBOUNDRunCmd and again passes the command string.
- If the first parameter is a “3” for an inbound map using memory I/O variables, the program calls the Java function JINRun and passes the second parameter to run the inbound map. The second parameter contains the command string, input/output memory file names, and the data of the memory files to run the specified inbound map. JINRun then calls the API INRun and again passes the command string.

- If the first parameter is a “4” for an outbound map using memory I/O variables, the program calls the Java function JOUTRun and passes the second parameter to run the outbound map. The second parameter contains the command string, input/output memory file names, and the data of the memory files to run the specified outbound map. JOUTRun then calls the API OUTRun and again passes the command string.

These Java calls automatically interface with *CALLRTP.DLL* that in turn calls *OWRM32C.DLL*. *OWRM32C.DLL* executes the map and returns a value that indicates the success of the map run. The *CALLRTP.DLL* contains a JNI (Java Native Interface) which calls an API function in *OWRM3C.DLL*. *OWRM3C.DLL* returns an integer from 0 to 5 indicating the degree of success of the map run. The value is returned to CALLRTP.DLL.

**Sample program for using a Java API call**

```
public class MYAPP
{
  public static void main(String[] args)
  {
    int ret;
    // Declare and initialize the RTP JAVA Class.
    RTP myrtp;
    myrtp = new RTP();

    // Convert the first argument as the case number.
    Integer num = Integer.valueOf(args[0]);
    switch (num.intValue()) {
      case 1:
        System.out.println("one");
        System.out.println("args[1] = " + args[1]);
       // Run an inbound map, pass the command switches
           as the parameter.
        ret = myrtp.JINBOUNDRunCmd(args[1]);
        System.out.println("ret = " + ret);
        break;

      case 2:
        System.out.println("two");
        System.out.println("args[1] = " + args[1]);
       // Run an outbound map, pass the command switches
as the parameter.
        ret = myrtp.JOUTBOUNDRunCmd(args[1]);
        System.out.println("ret = " + ret);
        break;
```

```
      case 3:
        System.out.println("three");
   // Prepare the string array as the JINRun() parameter
to run 271in map.
        String[] strarr_inrun =
        {
          // Command line switches for 271in map.
          "d:\\maps\\271in\\271gdtst.txt -dg
d:\\maps\\271in -dt d:\\maps\\271in -du d:\\maps\\271in
-xl -l -it -o -wx 1 -n -w -b -m 271IN -mx 2",
          // Input file name to be substituted with
memory buffer.
          "d:\\maps\\271in\\271gdtst.txt",
          // Input data of 271in map.
        "ISA*00*        *00*        *01*9012345720000
*01*9088877320000
*001030*1030*U*00401*000000001*0*T*:!\n" +

"GS*HB*901234572000*908887732000*20001030*1615*1*X*004
010X092!\n" +
          "ST*271*0001!\n" +
         "BHT*0022*11*3920394930203*20001030*1615!\n" +
          "HL*1**20*1!\n" +
          "NM1*PR*2*BLUE CROSS BLUE
SHIELD*****PI*9012345918341!\n" +
          "PER*IC*ARTHUR
JONES*TE*6145551212*FX*6145551214!\n" +
          "HL*2*1*21*1!\n" +

"NM1*1P*1*JOHNSON*BARBARA****SV*223447582752!\n" +
          "REF*1J*500!\n" +
          "HL*3*2*22*1!\n" +

"TRN*2*12345678900987654321768958473*1311234567*500!\n
" +
          "NM1*IL*1*DAVIS*SAM*T***MI*223344!\n" +
          "REF*18*223453424!\n" +
          "N3*PO BOX 123!\n" +
          "N4*CINCINNATI*OH*43017*US!\n" +
          "PER*IC**HP*6147562231*WP*6145221212!\n" +
          "DMG*D8*19720513*F!\n" +
          "INS*Y*18******F*N!\n" +

          "DTP*102*D8*001030!\n" +
          "EB*1*CHD*1*GP**6*100.00****N*N!\n" +
          "HSD*DY*100*DA*163395*6*100*1*A!\n" +
          "REF*18*4654746868565!\n" +
```

```
            "DTP*193*D8*001030!\n" +
            "III*BF*11!\n" +
            "LS*2120!\n" +
            "NM1*13*1*SMITH*MUFFY****24*111222333!\n" +
            "N3*157 WEST 57TH STREET!\n" +
            "N4*COLUMBUS*OH*43017*US!\n" +
          "PER*IC*MAGGIE MCGILLICUTTY*TE*6145551245!\n" +
            "PRV*AT*9K*3920394930203!\n" +
            "LE*2120!\n" +
            "SE*31*0001!\n" +
            "GE*1*1!\n" +
            "IEA*1*000000001!\n",
            // Output file name.
            "d:\\maps\\271in\\dbo_EL_ENVOY",
           // An empty string that indicates the previous
           // file name is substituted with output memory
file.
            ""
          };
          // Run an inbound map with Memory I/O, pass a
             string array as the parameter.
          ret = myrtp.JINRun(strarr_inrun);
          System.out.println("ret = " + ret);
          System.out.println("strarr_inrun[4] = " +
             strarr_inrun[4]);
          break;

        case 4:
          System.out.println("four");
          // Prepare the string array as the JOUTRun()
parameter to run 270out map.
          String[] strarr_outrun =
          {
            // Command line switches for 270out map.
           "d:\\maps\\270out\\270out.x12 270OUT HS -t 270
-xl -dg d:\\maps\\270out -dt d:\\maps\\270out -du
d:\\maps\\270out -xl -l -it -o -wx 1 -mx 2",
            // Input file name to be substituted with
memory buffer.
            "D:\\maps\\270out\\dbo_EL_ENVOY",
            // Input data of 270out map.
            "0394930203            SAM
 DAVIS                          PO BOX 123
 CINCINNATI                     OH
 43017                        19720513      F223344
 1        100BLUE CROSS BLUE SHIELD 9012345918341
```

```
JOHNSON                                223447582752
 AT SMITH
 \n",
       // Output file name.
       "d:\\maps\\270out\\270out.x12",
       // An empty string that indicates the previous
       // file name is substituted with output memory
file.
          ""
        };
          // Run an outbound map with Memory I/O, pass
          // a string array as the parameter.
        ret = myrtp.JOUTRun(strarr_outrun);

        System.out.println("ret = " + ret);

        System.out.println("strarr_outrun[4] = " +
strarr_outrun[4]);

        break;

      default:
        System.out.println("Invalid option!");
        break;    }  }}
```

## Using a Java package to execute ECRTP

For the ECRTP Java API on a PC, we are now providing the JAVA class (*RTP.class)* and the wrapper DLL (*callrtp.dll*) rather than the source code. Before using your Java API with your map, you need to follow these steps.

1 At your classpath, extract the *javartp.jar* (i.e. *jar xf javartp.jar*).

2 Copy *callrtp.dll* and *owrm32c.dll* from *%classpath%\com\sybase\vn\lib* directory to your PATH directory. You can add a PATH directory (*c:\lib*, for example) through your System Properties, Advanced Tab.

3 Check and modify the drive letter in the demo files to point to the hard drive where your map files are located.

4 Use the provided demo files (*demo1.bat, demo2.bat*, etc.) and their referenced sample maps to test with your ECRTP Java API before you try to run your own maps.

5 Modify DEMOAPP.java or build your own application with ECRTP Java API and reference the maps you want to run.

Following is a table of the error codes for the wrapper DLL:

***Table 2-7: Error code description***

| Error code | Description |
|---|---|
| 100 | An exception occurred when calling the API for ECRTP. This indicates that something may be wrong in your *OWRM32C.DLL*. For example, the version may be incorrect. |
| 101 | An exception occurred before the API for ECRTP was called. Exceptions other than 103-109 will be caught by this. |
| 102 | An exception occurred after the API for ECRTP was called. Exceptions other than 103-109 will be caught by this. |
| 103 | The incorrect number of elements were entered in a string array. |
| 104 | An exception occurred when the UTF character strings were released. |
| 105 | An exception occurred when getting the string array elements. System resource problem. |
| 106 | An exception occurred when a memory I/O structure pointer array was allocated. System resources problem. |
| 107 | An exception occurred when the memory I/O structure pointer for memory files was initialized. System resources problem. |
| 108 | An exception occurred when the output was set. System resource problem. |
| 109 | An exception occurred when deleting memiostruct elements. |

### Sample code for a Java package

Below is the sample code for DEMOAPP.java.

```
/*
   DEMOAPP.java
   This Java Program demonstrates how to use RTP Java
Class(RTP.class).

   There are 4 API functions provided by RTP Java Class.
   1. int JINBOUNDRunCmd(String Str);
```

```
   2. int JOUTBOUNDRunCmd(String Str);
   3. int JINRun(String[] strarr);
   4. int JOUTRun(String[] strarr);
```

The 3rd and 4th functions call Memory I/O functions. The parameter of these two functions is a string array. The very first element of the string array is the command line switches for running the map. The following elements are grouped by two elements. the first one is the file name that will be substituted with a memory file. The second one will be the data of the memory file. If the memory file is an input file, put the input data in this string. If the memory file is an output file, use an empty string to indicate it. Multiple input/output memory files can be specified in this string array.

DEMOAPP Class contains 4 cases that use each of the 4 RTP Java API functions. Each case is called by a separate batch file that is below after this listing.

Case 1: Run an inbound map, pass the command switches as the parameter.

Case 2: Run an outbound map, pass the command switches as the parameter.

Case 3: Run an inbound map with Memory I/O, pass a string array as the parameter.

Case 4: Run an outbound map with Memory I/O, pass a string array as the parameter.

```
   */

package com.sybase.vn.demo;
import com.sybase.vn.javartp.RTP;

public class DEMOAPP
{
  public static void main(String[] args)
  {
    int ret;
    // Declare and initialize the RTP Java Class.
    RTP myrtp;
    try
    {
      myrtp = new RTP();
    }
    catch(Throwable t)
    {
      System.err.println("Exception caught: " +
```

```
t.getMessage());
      return;
    }
    // Convert the first argument as the case number.
    Integer num = Integer.valueOf(args[0]);
    switch (num.intValue()) {
      case 1:
        System.out.println("one");
        System.out.println("args[1] = " + args[1]);
       // Run an inbound map, pass the command switches
as the parameter.
        ret = myrtp.JINBOUNDRunCmd(args[1]);
        System.out.println("ret = " + ret);
        break;

      case 2:
        System.out.println("two");
        System.out.println("args[1] = " + args[1]);
       // Run an outbound map, pass the command switches
as the parameter.
        ret = myrtp.JOUTBOUNDRunCmd(args[1]);
        System.out.println("ret = " + ret);
        break;

      case 3:
        System.out.println("three");
        // Prepare the string array as the JINRun()
parameter to run 271in map.
        String[] strarr_inrun =
        {
          // Command line switches for 271in map.
"d:\\classes\\com\\sybase\\vn\\maps\\271in\\271gdtst.t
xt -dg d:\\classes\\com\\sybase\\vn\\maps\\271in -dt
d:\\classes\\com\\sybase\\vn\\maps\\271in -du
d:\\classes\\com\\sybase\\vn\\maps\\271in -xl -l -it
-o -wx 1 -n -w -b -m 271IN -mx 2",
          // Input file name to be substituted with
memory buffer.
          "d:\\classes\\com\\sybase\\vn\\maps\\
271in\\271gdtst.txt",
          // Input data of 271in map.
         "ISA*00*         *00*         *01*9012345720000
 *01*9088877320000
*001030*1030*U*00401*000000001*0*T*:!\n" +

"GS*HB*901234572000*908887732000*20001030*1615*1*X*004
```

```
010X092!\n" +
          "ST*271*0001!\n" +
         "BHT*0022*11*3920394930203*20001030*1615!\n" +
          "HL*1**20*1!\n" +
          "NM1*PR*2*BLUE CROSS BLUE
SHIELD*****PI*9012345918341!\n" +
          "PER*IC*ARTHUR
JONES*TE*6145551212*FX*6145551214!\n" +
          "HL*2*1*21*1!\n" +
"NM1*1P*1*JOHNSON*BARBARA****SV*223447582752!\n" +
          "REF*1J*500!\n" +
          "HL*3*2*22*1!\n" +

"TRN*2*12345678900987654321768958473*1311234567*500!\n
" +
          "NM1*IL*1*DAVIS*SAM*T***MI*223344!\n" +
          "REF*18*223453424!\n" +
          "N3*PO BOX 123!\n" +
          "N4*CINCINNATI*OH*43017*US!\n" +
          "PER*IC**HP*6147562231*WP*6145221212!\n" +
          "DMG*D8*19720513*F!\n" +
          "INS*Y*18*******F*N!\n" +
          "DTP*102*D8*001030!\n" +
          "EB*1*CHD*1*GP**6*100.00****N*N!\n" +
          "HSD*DY*100*DA*163395*6*100*1*A!\n" +
          "REF*18*4654746868565!\n" +
          "DTP*193*D8*001030!\n" +
          "III*BF*11!\n" +
          "LS*2120!\n" +
          "NM1*13*1*SMITH*MUFFY****24*111222333!\n" +
          "N3*157 WEST 57TH STREET!\n" +
          "N4*COLUMBUS*OH*43017*US!\n" +
        "PER*IC*MAGGIE MCGILLICUTTY*TE*6145551245!\n" +
          "PRV*AT*9K*3920394930203!\n" +
          "LE*2120!\n" +
          "SE*31*0001!\n" +
          "GE*1*1!\n" +
          "IEA*1*000000001!\n",
          // Output file name.
          "d:\\classes\\com\\sybase\\vn\\maps\\
271in\\dbo_EL_ENVOY",
         // An empty string that indicates the previous
         // file name is substituted with output memory
file.
          ""
        };
```

```
        // Run an inbound map with Memory I/O, pass a
string array as the parameter.
        ret = myrtp.JINRun(strarr_inrun);
        System.out.println("ret = " + ret);
        System.out.println("strarr_inrun[4] = " +
strarr_inrun[4]);
        break;

      case 4:
        System.out.println("four");
        // Prepare the string array as the JOUTRun()
parameter to run 270out map.
        String[] strarr_outrun =
        {
           // Command line switches for 270out map.

"d:\\classes\\com\\sybase\\vn\\maps\\270out\\270out.x1
2 270OUT HS -t 270 -xl -dg
 d:\\classes\\com\\sybase\\vn\\maps\\270out -dt
 d:\\classes\\com\\sybase\\vn\\maps\\270out -du
 d:\\classes\\com\\sybase\\vn\\maps\\270out -xl -l -it
-o -wx 1 -mx 2",
          // Input file name to be substituted with
memory buffer.
          "d:\\classes\\com\\sybase\\vn\\maps
\\270out\\dbo_EL_ENVOY",
          // Input data of 270out map.
          "0394930203               SAM
          DAVIS                     PO BOX 123
          CINCINNATI                OH
          43017 19720513      F223344

          1 100BLUE CROSS BLUE SHIELD 9012345918341
          JOHNSON 223447582752
          AT SMITH
 \n",
       // Output file name.

"d:\\classes\\com\\sybase\\vn\\maps\\270out\\270out.x1
2",
    // An empty string that indicates the previous file.
    // name is substituted with output memory file.
    ""
    };
         // Run an outbound map with Memory I/O, pass a
         // string array as the parameter.
```

```
            ret = myrtp.JOUTRun(strarr_outrun);
            System.out.println("ret = " + ret);
            System.out.println("strarr_outrun[4] = " +
strarr_outrun[4]);
            break;
          default:
            System.out.println("Invalid option!");
            break;
        }
      }
}
```

Below is the contents of *demo1.bat*, which runs an inbound map and passes the command switches as the parameters.

```
REM demo1.batjava com.sybase.vn.demo.DEMOAPP 1

"d:\\classes\\com\\sybase\\vn\\maps\\271in\\271gdtst.t
xt -dg
d:\\classes\\com\\sybase\\vn\\maps\\271in -dt
 d:\\classes\\com\\sybase\\vn\\maps\\271in -du
d:\\classes\\com\\sybase\\vn\\maps\\271in -xl -l -it -
o -wx 1 -n -w -b -m
271IN -mx 2"
pause
```

Below is the contents of *demo2.bat*, which runs an outbound map and passes the command switches as the parameters.

```
REM demo2.batjava com.sybase.vn.demo.DEMOAPP 2
"d:\\classes\\com\\sybase\\vn\\maps\\270out\\270out.x1
2 270OUT HS -t 270 -xl -dg
d:\\classes\\com\\sybase\\vn\\maps\\270out -dt
d:\\classes\\com\\sybase\\vn\\maps\\270out -du
d:\\classes\\com\\sybase\\vn\\maps\\270out -xl -l -it -
o -wx 1 -mx 2"
pause
```

Below is the contents of *demo3.bat*, which runs an inbound map with Memory I/O and passes a string array as a parameter.

```
REM demo3.batjava com.sybase.vn.demo.DEMOAPP 3pause
```

Below is the contents of *demo4.bat*, which runs an outbound map with Memory I/O and passes a string array as a parameter.

```
REM demo4.bat
java com.sybase.vn.demo.DEMOAPP 4
pause
```

# Running ECRTP from a Visual Basic script

You can execute ECRTP from a Visual Basic script. For Visual Basic, there are four API functions to call an EDI server directly. The commands are:

- INBOUNDRUNCmd
- OUTBOUNDRUNCmd
- INRUN
- OUTRUN

Use INBOUNDRUNCmd and OUTBOUNDRUNCmd for inbound and outbound mapping. Use INRUN and OUTRUN for Memory I/O inbound and outbound mapping.

Visual Basic developers can design and build their own forms and module to call either of those API functions. Here is a code example for a module and a form.

## Source code for a module

**Note** As a VB developer, you need to built a similar module code to declare function and user type. It will be very helpful if you know how to call a function inside a dynamic link library. You need to know some concepts about Microsoft automatic data types and how to pass a string through VB and C/C++ code to do Memory I/O with Visual Basic.

```
'call INBOUNDRunCmd() of owrm32c.dll
Declare Function inBound _
    Lib "owrm32c" _
    Alias "INBOUNDRunCmd" _
    (ByVal commandline As String) As Integer

 Declare Function outBound _
    Lib "owrm32c" _
    Alias "OUTBOUNDRunCmd" _
    (ByVal commandline As String) As Integer
 ''user defined type to match struct memio in rtp engine
Type MEMIOSTRUCT
    filename As String 'file name for memory redirection
    paddr As String 'data string to in or out
    pbytes As Long 'size of data string in or out
```

```
    pbuflen As Long 'set to vbNullString
End Type
'call vb_OUTRUN () in rtpdll.dll which will call
OUTRun() of owrm32c.dll
Declare Function vb_OUTRUN _
    Lib "vbrtpdll" _
    Alias "OUTRun_vb" _
    (ByVal commandline As String, ByRef ioStruct_in As
MEMIOSTRUCT, ByRef ioStruct_out As MEMIOSTRUCT) As
Integer
'call vb_INRUN () in rtpdll.dll which will call INRun
() of owrm32c.dllDeclare Function vb_INRUN _
    Lib "vbrtpdll" _
    Alias "INRun_vb" _
    (ByVal commandline As String, ByRef ioStruct_in As
MEMIOSTRUCT, ByRef ioStruct_out As MEMIOSTRUCT) As
Integer
```

## Source code for Visual Basic form

Visual Basic developers can design their own forms to call the above alias functions. Here is an example of code:

```
Option Explicit
    Private Sub ExitBut_Click()
    End
End Sub

Private Sub INBOUNDRunCmdBut_Click(
)'call INBOUNDRunCmd () in owrm32c.dll
    Dim ret As Integer
    Dim inString As String
   'user's mapping commandline
    inString = "c:\maps\MapTestings\850in\s-850.x12 -dt
 c:\maps\MapTestings\850in -dg
c:\maps\MapTestings\850in -du
c:\maps\MapTestings\850in -eo
c:\maps\MapTestings\850in\850udf.txt -b -z -l -it -c"
    ret = inBound(inString)
    MsgBox " The return value is " & ret
End Sub

Private Sub OUTBOUNDRunCmdBut_Click()
    Dim ret As Integer
    Dim outString As String
```

```
   'user's mapping commandline
    outString = "c:\maps\MapTestings\850out\850o.x12
850OUT XX -t 850 -dt c:\maps\MapTestings\850out -dg
c:\maps\MapTestings\850out -du
c:\maps\MapTestings\850out -ei
c:\maps\MapTestings\850out\s-850.udf -z -it -l"
    ret = outBound(outString)
    MsgBox "The return value is " & ret
End Sub

Private Sub OUTRUNBut_Click()
    Dim ret As Integer 'return value from INRUN/OUTRUN
    Dim memBuf_in As String 'input buffer
    Dim memBytes_in As Long 'size of input buffer
    Dim memBuflen_in As Long 'length of input buffer

   Dim memBuf_out As String        'output buffer
    Dim memBytes_out As Long        '0
    Dim memBuflen_out As Long       'size of buffer

   Dim myStruct_in As MEMIOSTRUCT  'input memiostruct
    Dim myStruct_out As MEMIOSTRUCT 'output memiostruct
    Dim map_switches As String

   'open a file and load data into input buffer (memory)
    Dim myfile_in As String         'for test purpose
    Dim myfile_out As String        'for test purpose
    memBuf_in = String(1000, vbNullChar)
   'input disk file
    myfile_in =
"L:\WINDOWS\dbase\t850out\ver221\normal.x12"
    Open myfile_in For Input As #1
        memBuf_in = Input(LOF(1), #1)
        memBytes_in = Len(memBuf_in)
        memBuflen_in = 1000
    Close #1
    myStruct_in.filename = myfile_in
    myStruct_in.paddr = memBuf_in
    myStruct_in.pbuflen = memBuflen_in
    myStruct_in.pbytes = memBytes_in

'define output buffer and enough buffer size
'rtp engine will reallocate memory space and it is
illegal in VB
   'out put disk file
    myfile_out = "I:\TEST\850BP\DATA\PO_HEADI.SEQ"
```

```
    memBuflen_out = 2000              'define 2000 here
   memBuf_out = String(2000, vbNullChar)  'clear string
    memBytes_out = 0                      'no appending
    myStruct_out.filename = myfile_out
    myStruct_out.paddr = memBuf_out
    myStruct_out.pbuflen = memBuflen_out
    myStruct_out.pbytes = memBytes_out

 'set map switches
    map_switches = String(255, vbNullChar)
   ‘user’s mapping commandline
    map_switches =
"L:\WINDOWS\dbase\t850in\ver221\normal.x12 -dg
L:\WINDOWS\dbase\t850in\ver221 -dt
L:\WINDOWS\dbase\t850in\ver221 -n -it  -o -w -l -b -wx
0 -du L:\WINDOWS\dbase\t850in\ver221"
    'call vb_OUTRUN() in rtpdll.dll
    ret = vb_OUTRUN(map_switches, myStruct_in,
myStruct_out)
    MsgBox "The return value is " & ret

 Dim final_out As String
    'output
    final_out =
"L:\WINDOWS\dbase\t850in\ver221\PO_HEADI.SEQ"
    Open final_out For Binary As #2
        Put #2, , myStruct_out.paddr
    Close #2
    End Sub

Private Sub INRUN1_Click()
    Dim ret As Integer              'return value from
INRUN/OUTRUN
    Dim memBuf_in As String         'input buffer
   Dim memBytes_in As Long        'size of input buffer
    Dim memBuflen_in As Long        'length of input
buffer
    Dim memBuf_out As String        'output buffer
   Dim memBytes_out As Long       'size of data already
in buffer
   Dim memBuflen_out As Long     'size of output buffer
    Dim myStruct_in As MEMIOSTRUCT  'input struct
    Dim myStruct_out As MEMIOSTRUCT 'output struct
    Dim map_switches As String

   Dim myfile_in As String         'test purpose
```

```
    Dim myfile_out As String        'test purpose
    Dim final_out As String         'test purpose

 'open a file and load all data into memory
    memBuf_in = String(1000, vbNullChar)
    myfile_in =
"L:\WINDOWS\dbase\t850in\ver221\normal.x12"
    Open myfile_in For Input As #1
        memBuf_in = Input(LOF(1), #1)
        memBytes_in = Len(memBuf_in)
        memBuflen_in = 1000
    Close #1
    myStruct_in.filename = myfile_in
    myStruct_in.paddr = memBuf_in
    myStruct_in.pbuflen = memBuflen_in
    myStruct_in.pbytes = memBytes_in
    'output buffer    myfile_out =
"I:\TEST\850BP\DATA\PO_HEADI.SEQ"
    memBuflen_out = 2000
    memBuf_out = String(2000, vbNullChar)
    memBytes_out = 0
    myStruct_out.filename = myfile_out
    myStruct_out.paddr = memBuf_out
    myStruct_out.pbuflen = memBuflen_out
    myStruct_out.pbytes = memBytes_out

map_switches = String(255, vbNullChar)
 ‘user’s mapping commandline
    map_switches =
"L:\WINDOWS\dbase\t850in\ver221\normal.x12 -dg
L:\WINDOWS\dbase\t850in\ver221 -dt
L:\WINDOWS\dbase\t850in\ver221 -du
L:\WINDOWS\dbase\t850in\ver221 -n -it  -o -w -l -b -wx
0 "

 'call vb_OUTRUN() in rtpdll.dll
 ret = vb_INRUN(map_switches, myStruct_in,
myStruct_out)
    MsgBox "The return value is " & ret

 final_out =
"L:\WINDOWS\dbase\t850in\ver221\PO_HEADI.SEQ"
    Open final_out For Binary As #2
        Put #2, , myStruct_out.paddr
    Close #2
End Sub
```

CHAPTER 3

# User Exit Routines

# About user exit routines

User exit routines allow a user to invoke a proprietary routine from within a map, providing a way for a user to perform additional functions called by mapping rules. The user exit routine behaves like a call-back mechanism in event-driven programs. The following files are provided to help the user create user exit routines.

- *userex32.dll*
- *userexit.cpp*
- *fcnv.cpp*
- *userex32.lib*
- *userex32.def*

The *USEREX32.DLL* file provided with the program contains one User Exit Function FCNV. The FCNV function will read a file that has lines terminated by CRLF or LF and write them out to another file with fixed line length specified by the cpRegBuf input parameter. To call the FCNV function, the User Exit CpuExitName must be equal to “FCNV”, and the cpRegBuf parameter should contain the new line length, the old file name and the new file name. These values should be separated by one space in the cpRegBuf input parameter. If the parameter CpuExitName does not equal “FCNV” then the supplied program will simply ring a bell and return.

To have the USEREX32.DLL file perform other functions, the user must write his or her own “user exit” routine and insert the code in the appropriate place in the *USEREXIT.C* file, which is supplied with ECRTP. The user then compiles the modified *USEREXIT.C* file to create a new *USEREX32.DLL*.

The *USEREX32.LIB* file allows the user to implicitly link to the DLL from a C program. The DEF file provides information. The C program is recompiled to create the DLL file after the user’s code has been inserted.

The *USEREX32.DEF* file is furnished for informational purposes only.

The annotated example provided with the installation is shown below. This example represents a very simple routine, which the user is expected to modify. The user’s code should be inserted where the code says “ring bell – user will add code here”.

```
/*-------------------------------------------------
* UserExit - Windows DLL and Unix shared library.
* This file and userdll.c and userexit.def are used for
* userexit.DLL.
```

```
* This file alone is used for UNIX shared library
* userexit.sl
*-----------------------------------------------------
* USEREXIT function.
* Parameters:
*  char cpuExitName - pointer to dynamically allocated
*         storage which contains the routine name.
*         This pointer should not be written to.
*         This value can be used to determine what
*         action should be done by USEREXIT() function.
*  short sLanguage - will contain a 1 for Cobol or a 2
*         for 'C' language.
*  char *cpRegBuf - pointer to input buffer which
*         has been loaded with the value of a Memvar,
*         Record Field or Record Buffer.
*         For 16 bit program the maximum record buffer
*         length is 3200 characters. For 32 bit program
*         the maximum record buffer length is 10000
*         characters.
*  char cpRetBuf - pointer to output buffer, where the
*         routine output of Memvar, Record Field or
*         Record Buffer should be placed.
*
*         Note cpRegBuf and cpRetBuf have been set to
*         point to the same large buffer. Empty input
*         cpRegBuf before writing to output cpRetBuf.
*
*
*  char *cpStatus - pointer to dynamically allocated
*         storage which has been space filled and null
*         terminated to actual length of memvar of
*         status memvar. The user exit routine should
*         be careful not to store more information
*         in the field than it can hold.
*
* parameter returns: cpRetbuf, and cpStatus.
* return value: none
*-----------------------------------------------------*/
# ifndef NO_PROTO
  void fcnv(char *, char *);
#else
   void fcnv();
# endif

#ifndef
 UNIXextern "C" __declspec (dllexport)
```

```
void
WINAPI
USEREXIT(char *cpUExitName, short sLanguage, char
*cpRegBuf,
   char *cpRetBuf, char *cpStatus)
#else
#if defined(__cplusplus)
extern "C"
#endif
void
# ifndef NO_PROT
OUSEREXIT(char *cpUExitName, short sLanguage, char
*cpRegBuf,
   char *cpRetBuf, char *cpStatus)
# else
USEREXIT(cpUExitName, sLanguage, cpRegBuf,cpRetBuf,
cpStatus)
char *cpUExitName;
short sLanguage;
char *cpRegBuf, *cpRetBuf, *cpStatus;
# endif
#endif
{
    if(stricmp(cpUExitName, "FCNV") == 0)
   {
      fcnv(cpRegBuf, cpStatus);
   }
   else
   {
#ifndef UNIX
     MessageBeep(-1);   /* ring bell - user will add
code here */
#else
    write(0,"\007", 1);   /* ring bell - user will add
code here */
#endif
   }
}
/*********** File Conversion Program *****************
 ** UNWRAP.CPP - 11/11/98                  **
**-------------------------------------------------**
** THIS PROGRAM CONVERTS A SOURCE FILE OF VARIOUS LENGTH
**
** RECORDS INTO THE USER'S DESIRED SIZE RECORDS. **
** PARAMETER 1 - USER'S DESIRED RECORDS SIZE.   **
** PARAMETER 2 - SOURCE FILE NAME          **
```

```
** PARAMETER 3 - DESTINATION FILE NAME (CREATED BY
PROGRAM)   **
**-------------------------------------------------**/
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <ctype.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <io.h>
#include <process.h>
#include <windows.h>
#include <winbase.h>
#include <time.h>
#include <stddef.h>
#include <memory.h>

/*   ErrCondTYP  */
#define NoErrCOND      1  /* normal completion, no
                             errors detected */
#define OpenErrCOND    7  /* attempt opening an already
                             open file, */
#define FileMissingCOND   10 /*  operating system can't
                                find file */
#define OtherCOND      11 /*  other unspecified error
                             conditions    */
#ifndef NO_PROTO
lastchr(char *strng, int len);
#else
lastchr();
#endif

#ifndef NO_PROTO
void fcnv(char *xcmdline, char *cpStatus);
#else
void fcnv();
#endif
#define IMAX_BUF 1024
#ifndef NO_PROTO
void fcnv(char *xcmdline, char *cpStatus)
#else
void fcnv(xcmdline, cpStatus)
char *xcmdline, *cpStatus
;#endif
{
```

```
    char *argvp[3];
    unsigned char rdBfr[(IMAX_BUF + 2)];
   char arg1[302], arg2[302], arg3[302], buf[302];
   long rsize, lcnt, jcnt;

   FILE *srcfp, *dstfp;
   int at_eof, ich, k = 0, len, j, indx = -1;

   argvp[0] = arg1;
   argvp[1] = arg2;
   argvp[2] = arg3;
   memset(arg1, 302, '\0');
   memset(arg2, 302, '\0');
   memset(arg3, 302, '\0');
   len = lastchr(xcmdline, (int)strlen(xcmdline));
   if(len == 0)
   {
      sprintf(cpStatus, "%ld", OtherCOND);
      return;
   }
   while(k < len && indx < 2)
   {
      while (xcmdline[k] == ' ' && k < len)
         k += 1;
      j = 0;
      if(k < len)
      {
         indx += 1;
       /* Group single-quoted parameter as one argvp[]
entry. */
         if (xcmdline[k] == '\'')
         {
            k += 1; /* skip first ' */
           while (xcmdline[k] != '\'' && j < 300 && k
             < len) buf[j++] = xcmdline[k++];
            k += 1; /* skip last ' */
         }
      /* Group double-quoted parameter as one argvp[]
entry. */
         else if (xcmdline[k] == '\"')
         {
            k += 1; /* skip first " */
           while (xcmdline[k] != '\"' && j < 300 && k
< len)
               buf[j++] = xcmdline[k++];
            k += 1; /* skip last  " */
```

```
         }
         /* Normal argvp[] entry processing. */
        else while (xcmdline[k] != ' ' && j < 300 && k
< len)
         {
            buf[j++] = xcmdline[k++];
         }
         buf[j] = '\0';
                  strcpy(argvp[indx], buf);
      }
   }
  /**-----------------------------**
   ** Verify input parameter count **
   **-----------------------------**/
   if(indx != 2)
   {
#ifdef UNIX
     printf("\nTHE FCNV UserExit REQUIRES THREE
             PARAMETERS:\n");
     printf("PARAMETER 1 = NEW RECORD SIZE\n");
     printf("PARAMETER 2 = SOURCE FILE NAME\n");
     printf("PARAMETER 3 = DESTINATION FILE NAME\n");
#endif
     sprintf(cpStatus, "%ld", OtherCOND);
     return;
 } // END IF on number of parameters
  /**--------------------------------------**
   ** Convert the new record size to integer **
   **--------------------------------------**/
   rsize = atol(arg1);

 /**-----------------------**
   ** Open source file name **
   **-----------------------**/
   if ( (srcfp = fopen(arg2, "rb")) == (FILE *) NULL)
   {
#ifdef UNIX
       printf("ERROR OPENING FILE %s - Program
Terminated\n", argv[2]);
#endif
       sprintf(cpStatus, "%ld", FileMissingCOND);
       return;
   }
  /**-------------------------**
   ** Create destination file **
   **-------------------------**/
```

```
   if( (dstfp = fopen(arg3, "wb")) == (FILE *) NULL)
    {
#ifdef UNIX
      printf("ERROR CREATING DESTINATION FILE %s -
             Program Terminated\n", arg3);
#endif
      sprintf(cpStatus, "%ld", OtherCOND);
      return;
   }
   at_eof = 0;
   /* lcnt is index into rdBfr */
   /* jcnt is index up to size of wrap and inserted CR
LF */
   jcnt = lcnt = 0L;
    memset(rdBfr, 0, ( IMAX_BUF + 2) );

   while(at_eof == 0)
   {
    /**------------------**
    ** Read source file **
    **------------------**/
    if( (ich = fgetc(srcfp)) == EOF && feof(srcfp))
      {
         at_eof = 1;
         continue;
      }
      if(ich == 10 || ich == 13)
      {
         continue;
      }
      rdBfr[lcnt++] = ich;
      jcnt += 1;
      if(jcnt >= rsize)
      {
#ifndef UNIX
         rdBfr[lcnt++] = 13;
#endif
         rdBfr[lcnt++] = 10;
         jcnt = 0;
      }

      if(lcnt >= IMAX_BUF)
      {
         if((fwrite(rdBfr, (size_t) lcnt, (size_t) 1,
dstfp)) != 1)
         {
```

```
                fclose(srcfp);
                fclose(dstfp);
                unlink(arg3);
                sprintf(cpStatus, "%ld", OtherCOND);
                return;
             }
             lcnt = 0;
          }
       }
       if(jcnt != 0)  /* Need to pad out line */
       {
          while(jcnt < rsize)
          {
             rdBfr[lcnt++] = '\0';
             jcnt++;
             if(lcnt >= IMAX_BUF)
             {
                if((fwrite(rdBfr, (size_t) lcnt, (size_t)
1, dstfp)) != 1)
                {
                   fclose(srcfp);
                   fclose(dstfp);
                   unlink(arg3);
                   sprintf(cpStatus, "%ld", OtherCOND);
                   return;
                }
                lcnt = 0;
             }
          }

#ifndef UNIX
          rdBfr[lcnt++] = 13;
#endif
          rdBfr[lcnt++] = 10;
       }
       if(lcnt > 0)
       {
         if((fwrite(rdBfr, (size_t) lcnt, (size_t) 1,
dstfp)) != 1)
          {
             fclose(srcfp);
             fclose(dstfp);
             unlink(arg3);
             sprintf(cpStatus, "%ld", OtherCOND);
             return;
          }
```

```
   }
   fclose(srcfp);
   fclose(dstfp);
   sprintf(cpStatus, "%ld", NoErrCOND);
   return;
}
 /* END main */
/*-------------------------------------------------*/
int
#ifndef NO_PROTO
lastchr(char *strng, int len)
#else
lastchr(strng, len)
char *strng;
int len;
#endif
{
   int i;
   char *ptr;

   if(len)
   {
      i = len -1;
      for(ptr = strng+i; ptr >= strng; ptr--)
      {
         if(*ptr != ' ' && *ptr != '\0') break;
      }
      i = (int) (ptr - strng + 1);
      return(i);
   }
   else
      return(0);
}
```

CHAPTER 4

# Using ECRTP as an Adapter



Information included in using ECRTP is:

- Configuration file for the Acquire Mode
- Configuration file for the Deliver Mode
- Configuration file for the Process Mode

# Using ECRTP as an adapter

ECRTP can be used as a plug-in adapter (called EDIadapter) with these core integration products:

- e-Biz 2000
- e-Biz Integrator
- MQSeries Integrator

To use ECRTP as an adapter, you must perform the following additional actions that are not required when ECRTP is used as a stand-alone product.

- Install ARE (Adapter Runtime Environment) from the New Era of Networks EDI Products CD onto the same server that has ECRTP. See the *ECRTP* Installation Guide for instructions on how to install the ARE.
- Export a schema to the core integration product. The schema can be created in ECMap and exported to the other product's schema repository, or the format of a schema that already exists in the other product's schema repository can be manually entered in ECMap. The finished schema tells the integration product which adapter to run (in this case, the EDIadapter), provides specific queue-related information, and supplies information required by the EDIadapter. (Refer to the *ECMap* Reference Guide for a detailed explanation of the utility that creates and exports a schema.)
- Create a configuration file. In ECMap, click on the Create Adapter Configuration File button in the Run Inbound Map, Run Outbound Map, or Run Any-to-Any Map Option 2 screens. When this button is clicked, ECMap creates a partial configuration file, which must then be modified based on a variety of factors. The configuration file will either be in Acquire, Deliver, or Process Mode. The type of configuration file depends on the map that is run. After the button is clicked, use Windows Explorer to locate the configuration file, open the file, and modify it.

Example configuration files for the Acquire Mode, the Deliver Mode, and the Process Mode are shown below. Note that the keys and values in the configuration files are case sensitive. For complete details of how to use the EDIadapter, see the *Adapter Runtime Environment User Guide*.

## Configuration file for the Acquire Mode

After you click on Run Map, remove all of the "#" comment symbol in column one for every line except the two sentences below ("# The following is a sample configuration file." and "# You must modify the settings to work for your instance."). After you remove the comment symbols, modify the lines that have instructions listed below.

```
Adapter
    clash.avoid=TRUE
    continue.format.exists=TRUE
    adapter=EDIAdapter
    mode=ACQUIRE
    data=NDO

# The following is a sample configuration file.
# You must modify the settings to work for your
# instance.
    #maximum_num_retries=2
    #transport.out.name=OUTQ
    #failurequeue.name=FAIL
#output.serializer.factory=XMLSerializer_Factory
#output.serializer.library=adk33xmlsd
#output.serializer.factory=NCFSerializer_Factory
#output.serializer.library=adk33ncfsd
#prefix=<prefix>    # Obtain prefix name from the
                    # configuration file generated
                    # from Export Schema to MQSI, e-Biz
                    # Integrator, or e-Biz 2000.
#msg.type=<prefix>.IC.<schema name>  # Obtain the
                    # prefix name from the file
                    # configuration file generated from
                    # Export Schema to MQSI, e-Biz
                    # Integrator, or e-Biz 2000.)

#transport.context.name=ADKContext

#OTContext.ADKContext
    #NNOT_CTX_DEFAULT_TIL_ID=FAIL
    #NNOT_CTX_TMID=MQSeriesTM
    #NNOT_CTX_ENFORCE_TX=TRUE

#TransactionManager.MQSeriesTM
    #NNOT_SHARED_LIBRARY=oti21mqstm
    #NNOT_FACTORY_FUNCTION=NNOTMQSeriesTXManagerFactory
    #NN_TM_MQS_QMGR=TEST_QMGR  # Obtain the Queue
                               # Manager name from MQSeries
```

```
#Session.ADKSession
    #NNOT_SHARED_LIBRARY=dbt21mqs
    #NNOT_FACTORY_FUNCTION=NNMQSSessionFactory
    #NNMQS_SES_OPEN_QMGR=TEST_QMGR # Enter the Queue
                                   # Manager name from
                                   # MQSeries.

#TRANSPORT.OUTQ
    #NNOT_SHARED_LIBRARY=dbt21mqs
    #NNOT_FACTORY_FUNCTION=NNMQSQueueFactory
    #NNOT_TIL_OPEN_SESSION_ID=ADKSession
    #NNOT_TIL_OPEN_TSI=TEST_OUT   # Enter the local
                              # queue to which the message
                              # is being put in
                              # MQSeries.

#Transport.FAIL
    #NNOT_SHARED_LIBRARY=dbt21mqs
    #NNOT_FACTORY_FUNCTION=NNMQSQueueFactory
    #NNOT_TIL_OPEN_SESSION_ID=ADKSession
    #NNOT_TIL_OPEN_TSI=TEST_FAIL  # Enter the local
                                  # queue that serves as
                                  # the failure queue in
                                  # MQSeries.

#Session.FileSession
    #NNOT_SHARED_LIBRARY=nnfile
    #NNOT_FACTORY_FUNCTION=NNSesFileFactory
    #NN_SES_MSG_SIZE=1000
    #NN_SES_SERVER=bsmith1

EDIAdapter # The lines below are unique to
# each map generated by the Run
# Map screen.
rtp.mapswitches="C:\Program
Files\nnsy\ECMap\maps\NDO2\EDI_TO_NDO\training.x12"
-xl -dg "C:\Program
Files\nnsy\ECMap\maps\NDO2\EDI_TO_NDO" -dt "C:\Program
Files\nnsy\ECMap\maps\NDO2\EDI_TO_NDO" -n -it  -o -l
-m EDI_TO_NDO -b -du "C:\Program
Files\nnsy\ECMap\maps\NDO2\EDI_TO_NDO" -wx 1

rtp.acquire_dir=I
```

In some cases, an additional line must be added to the AcquireNDO configuration file:

```
rtp.acquire_dir="1" (or "0")
```

When the EDIadapter is used to read XML data, a change must be made to the configuration file. ECRTP uses outbound processing to read XML data, and the EDIadapter uses the Deliver Mode for outbound processing. However, when EDI data is being converted to an NDO data tree, the EDIadapter normally uses the Acquire Mode. In order for the EDIadapter to read and process the XML input data, a line (that was added only for the EDIadapter Acquire Mode) must be changed. The line rtp.acquire_dir=”1” must be changed to rtp.acquire_dir=”0”.

## Configuration file for the Deliver Mode

Deliver Mode is used to get information from the transport and deliver it to the application. Outbound maps use the Deliver Mode. In Deliver NDO Mode, the adapter receives a deserialized data tree. After you click on Run Map, remove all of the “#” comment symbol in column one for every line except the two sentences below (“# The following is a sample configuration file.” and “# You must modify the settings to work for your instance.”). After you remove the comment symbols, modify the lines that have instructions listed below.

```
Adapter
    clash.avoid=TRUE
    continue.format.exists=TRUE
    adapter=EDIAdapter
    mode=DELIVER
    data=NDO

# The following is a sample configuration file.
# You must modify the settings to work for your
# instance.

    #transport.in.name=INQ
    #transport.failure_store_name=FAIL
    #maximum.transport.retries=2
    #transport.exit_if_empty=true
    #input.serializer.factory=NCFSerializer_Factory
    #input.serializer.library=adk33ncfsd
    #input.serializer.factory=XMLSerializer_Factory
    #input.serializer.library=adk33xmlsd
    #prefix=<prefix> >  # Obtain the prefix name from
                        # the configuration file
                        # generated from Export Schema
                        # to MQSI, e-Biz Integrator,
                        # or e-Biz 2000.
```

```
    #msg.type=<prefix>.IC.<schema name> # Obtain the
                        # prefix name from the
                        # configuration file generated
                        # from Export Schema to MQSI,
                        # e-Biz Integrator, or e-Biz
                        # 2000.
    #transport.context.name=ADKContext

#OTContext.ADKContext
    #NNOT_CTX_DEFAULT_TIL_ID=FAIL
    #NNOT_CTX_TMID=MQSeriesTM
    #NNOT_CTX_ENFORCE_TX=TRUE

#TransactionManager.MQSeriesTM
    #NNOT_SHARED_LIBRARY=oti21mqstm
    #NNOT_FACTORY_FUNCTION=NNOTMQSeriesTXManagerFactory
    #NN_TM_MQS_QMGR=TEST_QMGR # Obtain the Queue Manager
                        # name from
                        # MQSeries.#Session.ADKSession
    #NNOT_SHARED_LIBRARY=dbt21mqs
    #NNOT_FACTORY_FUNCTION=NNMQSSessionFactory
    #NNMQS_SES_OPEN_QMGR=TEST_QMGR     #Enter the Queue
                               # Manager name from
                               # MQSeries.#TRANSPORT.INQ
    #NNOT_SHARED_LIBRARY=dbt21mqs
    #NNOT_FACTORY_FUNCTION=NNMQSQueueFactory
    #NNOT_TIL_OPEN_SESSION_ID=ADKSession
    #NNOT_TIL_OPEN_TSI=TEST_OUT # Enter the local queue
                               # to which the message
                               # is being put in
                               # MQSeries.

#Transport.FAIL
    #NNOT_SHARED_LIBRARY=dbt21mqs
    #NNOT_FACTORY_FUNCTION=NNMQSQueueFactory
    #NNOT_TIL_OPEN_SESSION_ID=ADKSession
    #NNOT_TIL_OPEN_TSI=TEST_FAIL    # Enter the local
                               # queue that serves as
                               # the failure queue in
                               # MQSeries.EDIAdapter
                  # The lines below are unique to each map
                   # generated by the Run Map screen.

    rtp.mapswitches=NDO_TO_EDI.x12 NDO_TO_EDI PO -t 850
-dg "C:\Program Files\nnsy\ECMap\maps\NDO2\NDO_TO_EDI"
-dt "C:\Program Files\nnsy\ECMap\maps\NDO2\NDO_TO_EDI"
```

```
-n -it  -o -l -wx 1

    rtp.acquire_dir=O
```

## Configuration file for the Process Mode

In addition to converting data to and from an EDI format, the EDIadapter can also be used with an NDO data tree as both input and output. The EDIAdapter normally uses the Deliver Mode when NDO is the input. Process Mode is used to get data from a transport, enrich the data or submit a request and get a response, and put the data to another transport. Process Mode is most appropriate when interfacing with applications that operate in a synchronous manner such as a CORBA call, an RFC request reply, or a database SELECT.

The Process Mode file already has comments removed. Copy the file below to a configuration file and then modify the lines that have instructions listed below.

```
Adapter
    adapter=EDIAdapter
    mode=PROCESS
    data=NDO
    prefix=<prefix>  # Obtain the prefix name from the
                     # Schema configuration
                     # file generated from Export Schema
                     # to MQSI, e-BizIntegrator,
                     # or e-Biz 2000.
    msg.type=<prefix>.IC.<schema name>  > # Obtain the
                        # prefix name from the
                        # configuration generated from
                        # Export Schema to MQSI, e-Biz
                        # Integrator, or e-Biz 2000.
    maximum_num_retries=2
    transport.context.name=ADKContext
    transport.out.name=OUT
    failurequeue.name=FAIL
    transport.in.name=IN
    transport.out.name=OUT
    transport.failure_store_name=FAIL
    maximum.transport.retries=2
    transport.exit_if_empty=true
    acknowledge.put=true

#NCF Serializer
    Input.Serializer.Factory=NCFSerializer_Factory
```

```
    Input.Serializer.Library=adk33ncfsd
    Output.Serializer.Factory=NCFSerializer_Factory
    Output.Serializer.Library=adk33ncfsd

#XML Serializer
    #Input.Serializer.Factory=XMLSerializer_Factory
    #Input.Serializer.Library=adk33xmlsd
    #Output.Serializer.Factory=XMLSerializer_Factory
    #Output.Serializer.Library=adk33xmlsd

OTContext.ADKContext
    NNOT_CTX_DEFAULT_TIL_ID    = FAIL
    NNOT_CTX_TMID   = MQSeriesTM
    NNOT_CTX_ENFORCE_TX   = TRUE

TransactionManager.MQSeriesTM
    NNOT_SHARED_LIBRARY    = oti21mqstm
    NNOT_FACTORY_FUNCTION =
                          NNOTMQSeriesTXManagerFactory
    NN_TM_MQS_QMGR = TEST_MQ1   # Obtain the Queue
                         # Manager name from
                         # MQSeries.

Session.ADKSession
    NNOT_SHARED_LIBRARY    = dbt21mqs
    NNOT_FACTORY_FUNCTION  = NNMQSSessionFactory
    NNMQS_SES_OPEN_QMGR   = TEST_MQ1   # Enter the Queue
                                # Manager name from
                                # MQSeries.Transport.IN
     NNOT_SHARED_LIBRARY        = dbt21mqs
     NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
     NNOT_TIL_OPEN_SESSION_ID   = ADKSession
     NNOT_TIL_OPEN_TSI      = TESTQ    # Enter the local
                                  #   queue where
                                  # the message(s)
                                  # originally reside
                                  # on the queue.

Transport.OUT
     NNOT_SHARED_LIBRARY        = dbt21mqs
     NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
     NNOT_TIL_OPEN_SESSION_ID   = ADKSession
     NNOT_TIL_OPEN_TSI          = OUTQ    # Enter the
                                  # local queue where
                                  # the message(s) is
                                  # being moved.
```

```
Transport.FAIL
    NNOT_SHARED_LIBRARY        = dbt21mqs
    NNOT_FACTORY_FUNCTION      = NNMQSQueueFactory
    NNOT_TIL_OPEN_SESSION_ID   = ADKSession
    NNOT_TIL_OPEN_TSI          = FAILQ      # Enter the
                                  # local queue that
                                  # serves as the
                                  # failure queue in
                                  # MQSeries.

    EDIAdapter          # The lines below are unique
                        # to each map and must
                        # be carefully edited.

rtp.mapSwitches="C:\Program Files\Nnsy\ECMap\Maps\
NDO2\temp.x12" NDO_TO_NDO PO -t ANY -dg "C:\Program
Files\Nnsy\ECMap\Maps\NDO2\NDO_TO_NDO" -dt "C:\Program
Files\Nnsy\ECMap\Maps\NDO2\NDO_TO_NDO" -nt -ne -c -l -
wx 0
    rtp.acquire_dir=O
```

After the EDIadapter configuration file has been created and edited, you can click on Run Map. ECMap will call the *NNSYadapter33.exe* with the command line parameter: `"-file=<configuration file>"`.

CHAPTER 5

# Using ECRTP in a Web Environment

# Using ECRTP in a Web environment

ECRTP can be used to map data to and from HTML or XML data. There are a number of features in ECMap (the map development program) that support the use of ECRTP in a Web environment.

On the Web Script tab of the Run Inbound Map and Run Outbound Map, ECMap has a utility that creates an ASP (Active Server Page) that contains the runtime script or a CGI (Common Gateway Interface) that contains the script. You have the choice of creating uncompiled C code or compiled executable code. If you create uncompiled code, you can make changes to it. The map switches are in the code itself, and you must compile it before you use it. If you create compiled code, you cannot make changes to the code. The map switches are in a separate file outside the code.

CHAPTER 6

# ECRTP Performance

# Factors affecting performance

A variety of factors can influence ECRTP's speed and performance, including the design of the maps being run, the database technology being used, and the use of optional product features designed to enhance performance.

Map design is the responsibility of the individual mappers. Maps can impact runtime performance because they reflect complicated business logic that is required, but sometimes maps have not been written in the most efficient manner or have not taken advantage of performance-enhancing features. Two important performance-enhancing features are:

- Map caching
- Memory I/O

## Map caching

Map caching is designed to improve the interactive character of ECRTP by eliminating the processing time associated with opening and closing maps. The user is able to specify that maps will be cached in memory and to specify the maximum number of maps that will be cached. The user is also able to load specific maps into memory before running the ECRTP.

The user is able to specify that a map will stay open, using the –mx <maximum number of cached maps> switch. A separate switch is required for each map that will remain open. This functionality is available through both the ECRTP .dll and the ECRTP executable. For the ECRTP executable, maps will stay open only until the executable is finished. For the ECRTP *.dll*, the maps will stay open until the user explicitly closes them. The open maps are maintained in a structure in memory, with the mapname as the index key. An API allows a pointer (to the array of maps stored in memory) to be passed back to the calling program.

The user specifies the maximum number of maps that will remain open. If no maximum is specified, the default is 0 maps. As a result, if the user does not explicitly set a maximum number of maps, the ECRTP will not cache maps in memory. If the maximum number of maps is reached and the user specifies that a new map is to remain open, the program uses an algorithm to determine the least frequently used map of the maps that are currently open and replace that map with the new map. There is no upper limit to the maximum number of maps except that imposed by memory.

- FREEALLMAP() closes all open maps. (-xf switch)

- FREEMAPNAME(char*) closes a specific map. There is no switch because you cannot close a specific map from the command line.

The user passes in the map name, and the function returns a "-1" if the map is not found and a "1" if the map is found and removed. This functionality is not applicable for the ECRTP executable because it automatically frees all maps in memory when it is finished, and consequently, ignores any -xf switch.

The user may use UTILCONTMAP and UTILMAPNAME to decide which map to free from memory with the FREEMAPNAME() API.

```
extern "C" DLLEXPORT int WINAPI UTILCOUNTMAP(int
*mapMaxLoad,

int *mapLoaded);extern "C" DLLEXPORT int WINAPI
UTILMAPNAME(int mapPosition, char

 *mapName, int mapNameBufSize);
```

There are two return parameters for UTILCOUNTMAP. The first return parameter is int *mapMaxLoad. This is the maximum number of maps that can be loaded into memory. This value is set by the initial -mx N parameter on the command line API (for example, OUTBOUNDRUNCmd, OUTRun, INBOUNDRUNCmd, or INRun). The second return parameter is int *mapLoaded. This is the number of maps that have actually been loaded into memory.

The API function returns zero (success) as long as the original -mx N switch is used to allow multiple map processing. The API function returns -1 if there is no -mx switch or there was a -mx 0 switch.

There are two input parameters for UTILMAPNAME. The first input parameter is int mapPosition. The valid values are 0 to maximum number of maps. The first map is loaded at index zero and the second map is loaded at index position 1, and so forth. The second input parameter is int mapNameBufSize. This input parameter provides the maximum length of the parameter String mapName.

The output parameter for UTILMAPNAME is char * mapName which specifies the name of the map loaded at mapPosition index.

UTLIMAPNAME returns a zero (success) and the name of the map at position mapPosition in the mapName return parameter. If no map was loaded at position, mapPosition or the map name size was greater than mapNameBufSize then a -1 for failure is returned.

Using the LOADMAP API, the user is able to load map(s) into memory before the ECRTP is run. The user passes in the map directory, map name, and map direction, as well as the maximum number of maps that can be cached at one time. (The maximum number passed to this DLL must be the same maximum number that is specified in the -mx <maximum number of maps> switch on the ECRTP command line.

The arguments for the LOADMAP API are shown below:

```
LOADMAP(LPSTR dir_path, LPSTR name, int map_dir, int
MulMaps)

dir_path -- directory where map file store in disk.
             such as "c:\\temp\\850IN"

name -- name of map file without extension file name
          such as "T850IN" not "T850IN.MAP"

map_dir --    0 means INBOUND,
              1 means OUTBOUND

MulMaps-- how many maps are allowed to save in memory.
          It should have same value as -mx switch when
          the run ECRTP
```

The LOADMAP API returns one of the following four error/success codes:

- 0 – success
- -1 – input directory or map name does not exist
- -2 – number of maps is memory cannot be < 0
- -3 – only MAP_IN and MAP_OUT are legal input for map_dir

Neither -xf nor -mx is case-sensitive.

## Memory I/O

Memory I/O is designed to speed up performance by allowing the user to read or write from memory rather than the physical disk. The user can read from stdin or a specified memory address, or write to stdout or a temporary memory address.

- To read from stdin instead of from disk, the user can enter information on the Standard Input pane on the I/O Redirect tab of the Run Outbound Map screen. If the data being read is application data, the user can enter this information using the –mi switch/parameter at runtime. If the data being read is EDI data, the user can enter this information using the –xmi switch/parameter at runtime.
- To write to stdout instead of to disk, the user can enter information on the Standard Output pane on the I/O Redirect tab of the Run Inbound Map screen. If the data being written is application data, the user can enter this information using the –mo switch/parameter at runtime. If the data being written is EDI data, the user can enter this information using the –xmo switch/parameter at runtime.
- To read from or write to a temporary internal memory address, the user can enter information on the Internal Memory pane on the I/O Redirect tab of the Run Inbound Map or Run Outbound Map screen. The user can also enter this information using the –mm switch/parameter at runtime.

There is an additional memory I/O option, which allows the user to read or write from a specific memory address, but this option is available only when ECRTP is invoked from a DLL.

- To read application data from or write application data to a specific memory address, the user must use the –mp parameter at runtime; this information cannot be entered in ECMap. The format of the –mp switch is -mp <full-path file name> <pointer to memory address> <pointer to # of bytes> <pointer to size of buffer>.
- To read EDIdata from or write EDI data to a specific memory address, the user must use the –xmp parameter at runtime; this information cannot be entered in ECMap. The format of the –xmp switch is -xmp <pointer to memory address> <pointer to # of bytes> <pointer to size of buffer>.

## Database technology

Maps that use ODBC databases for either trading partner information or logging will take longer to execute than maps that use non-ODBC databases.

The performance of maps that use ODBC databases for trading partner information improves when dBase is used as the ODBC database. ECMap has a utility that allows ECRTP to run against their trading partner information in dBase. Refer to the Run Map chapter of the ECMap Reference Guide for more information about using this feature.

# Windows runtime parameters/switches

The following tables contain a listing of all the Windows parameters/switches that can be used with the runtime program (ECRTP)—both when it is run from a script and when it is invoked with a function call. For each parameter/switch, there is a brief description, an indication of whether the parameter/switch is used for inbound or outbound messages, and an indication of whether it is required or optional. The first table contains parameters/ switches that are used with both non-ODBC and ODBC trading partner files, while the second table contains parameters that are used only when the trading partner information is in an ODBC database. Each parameter is described in greater detail following the table.

***Table 6-1: Parameters/switches for trading partner files***

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -a | Updates the ISA Out control count field only in the ALL TradePartner record when the interchange envelopes are built. | Outbound | Optional |
| -ab | Specifies a new full path and file name to be used in place of the BAD EDI file. | I/O | Optional |
| -af | Specifies a new full-path file name to be used in place of a file name embedded in the map file. | I/O | Optional |
| -ag | Specifies a new full path and file name to be used in place of the GOOD EDI file. | I/O | Optional |
| -as | Checks that each ST Transaction Control Number in a GS to GE is greater than the previous ST Transaction Control Number. Validation assumes control numbers appear in ascending order. | Inbound | Optional |
| -b | Does not save the rejected EDI transactions/messages into the *badtrans.nmt* file. | Inbound | Optional |
| -c | Closes the trace file after every write statement. | I/O | Optional |

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -clz | Flags leading zeros in numeric X12 fields as an error on HIPAA compliance maps.<br>An error is flagged if leading zeros are not used to meet the minimum length requirement of that particular element.<br>If the leading zeros are necessary to meet the minimum length of the element, no error is generated. This option does not check leading zeros on non-compliance maps. | Inbound | Optional |
| -cu | Checks for unique control numbers within a transaction. Control numbers can now occur in any sequence, as long as they are unique.<br>If duplicate control numbers are found, ECRTP logs a 6054 error. Use only one of the switches (-as or -cu) for any map run. If both switches are present, ECRTP defaults to the last switch encountered. | Inbound | Optional |
| -db | Does not delete outbound temporary files that are created when processing multiple files. | Outbound | Optional |
| -dg | Specifies the directory in which the map files are located. | I/O | Required |
| -dm | Specifies the directory in which the trading partner, map, and log files are located. | I/O | Optionally<br>Required |
| -dt | Specifies the directory in which the trading partner files are located. | I/O | Required for Non-ODBC |
| -du | Specifies the directory to be used in place of the application directories embedded in the map. | I/O | Optional |
| -dw | Specifies the directory in which the company (*wixset.dat*) file is located. | Outbound | Optional |
| -ec | Does not create the transaction log file (*translog.in*, *translog.out*, or *trlog*) or the status file (*status.in* or *status.out*). | I/O | Optional |

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -ed | Specifies the directory in which the transaction log file (*translog.in*, *translog.out*, or *trlog*), trace file (*incoming.err*, *outgoing.err*, or *trnn.dat*), and status file (*status.in* or *status.out*) are located. | I/O | Optional/<br>Required for Tandem and Stratus |
| -ee | Ends processing of the EDI file after it processes the specified number of characters. | Inbound | Optional |
| -ef | Does not create the status file (*status.in* or *status.out*) | I/O | Optional |
| -ei | Specifies the full-path file name to be used in place of the input file name embedded in the map. | Outbound | Optional |
| -el | Specifies the full-path file name to be used for the transaction log. | I/O | Optional<br>Required for Tandem and Stratus |
| -eo | Specifies the full-path file name to be used in place of the output application filename embedded in the map. | Inbound | Optional |
| -er | Performs the trading partner lookup based on the group receiver. (basic reverse lookup) | Inbound | Optional |
| -es | Starts processing the EDI file after it has read a specified number of characters. | Inbound | Optional |
| -et | Specifies the directory in which the trace file is located. (*incoming.err* or *outgoing.err*) | I/O | Optional |
| -eu | Specifies a string variable to be used in place of all but the first character in the application file name (not including the file extension) embedded in the map. | I/O | Optional |
| -ev | Specifies a string variable to be placed in front of the application file name in the map. | I/O | Optional/ |
| -id | Specifies a Run ID number to be used instead of having the program look it up. If there is a log database, the program normally looks up the Run ID number in the Run ID table in the log database. | Inbound | Optional |
| -it | Ignores the trading partner mailbox. | I/O | Optional |

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -k | Sets compliance checking. | Inbound | Optional |
| -kf | Splits files into multiple files only once. | Outbound | Optional |
| -l | Writes long trace messages to an error file. | I/O | Optional |
| -m | Runs a specific map (identified by the mapname, which is the file name of the map with no extension) without referencing company or trading partner files. | Inbound | Optional |
| -mi | Uses stdin in place of a file embedded in the map. | Inbound | Optional |
| -mm | Uses a temporary memory location in place of a file embedded in the map. | I/O | Optional |
| -mn | Passes the map name extension as part of a command line argument.<br>The Map Name Extension works with other map lookup fields to find a correct map. For map functions with multiple parameters, such as LOADMAP, the correct map name should be found by using current map lookup fields with the Map Name Extension before calling the functions. | I/O | Optional |
| -mo | Uses stdout in place of a file embedded in the map. | Outbound | Optional |
| -mp | Uses a specific memory address in place of a file embedded in the map. | I/O | Optional |
| -mx | Keeps a specified number of maps open in memory. | I/O | Optional |
| -n | Uses the ALL TradePartner record if no trading partner match is found in the trading partner file. | I/O | Optional |
| -ncb | Indicates that the EDI file to be processed is an NCPDP batch file. Required for all inbound NCPDP files unless -nct is active. | Inbound | Required for inbound NCPDP batch files |
| -nct | Indicates that the EDI file to be processed is an NCPDP telecommunications file. Required for all inbound NCPDP files unless -ncb is active. | Inbound | Required for inbound NCPDP telecommunications files |
| -ne | Does not produce an outbound EDI file. | Outbound | Optional |

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -nret | Adds new return codes that provide more information to the calling program. Based on the return value, the user can determine the next step in the process. The return codes reflect the following information:<br>• At least one interchange or group is in error implying a TA1 map should be run.<br>• At least one group or transaction is in error, implying a 997 map should be run.<br>• At least one good transaction is present, implying a translation map should be run. | I/O | Optional |
| -nt | Does not perform a trading partner lookup. Uses the map specified in the command line. | Outbound | Optional |
| -nz | Maps numeric data literally (including blank fields) as it appears on the map. Previous versions of ECRTP pad numeric values with leading zeros based on field length. | I/O | Optional |
| -o | Does not use the trade agreement mailbox directory and file name. | I/O | Optional |
| -ol | Triggers a series of look ups against the Trading Partner database when the ECMap/EC Gateway Log is used as input.<br>If a trading partner match is found, ECRTP uses the entry to populate the EDI envelope. | Outbound | Optional |
| -pe | Pads alphanumeric fields with trailing spaces if those spaces are required to meet the minimum length of the element. Numeric fields will be padded with leading 0s if those 0s are required to meet the minimum length of the element.<br>This switch is an optional replacement for `PADEDI` environmental variable. | Outbound | Optional |

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -pf | Uses contents of the file as the command line parameters | I/O | Optional |
| -r | Specifies the maximum number of cross-reference table entries that will be loaded into memory. Anything over the maximum must be accessed from the database. | I/O | Optional |
| -rb | Ignores the trade agreement mailbox and places routed EDI data in the trading partner BAD mailbox. | I/O | Optional |
| -re | Ignores the trade agreement mailbox and places routed EDI data in the trading partner IN mailbox. | I/O | Optional |
| -rg | Ignores the trade agreement mailbox and places routed EDI data in the trading partner GOOD mailbox. | I/O | Optional |
| -ro | Ignores the trade agreement mailbox and places routed EDI data in the trading partner OUT mailbox. | I/O | Optional |
| -rt | Ignores the trade agreement mailbox and places routed EDI data in the trading partner OTHER mailbox. | I/O | Optional |
| -s | Does not produce a trace file. | I/O | Optional |
| -s3 | Processes the X12 ST03 element. | I/O | Optional |
| -sc | Validates the sequence of the Interchange (ISA) and Group (GS) control numbers. If you use this command, RTP checks the current control number against the Trading Partner database to validate the entry. If the entry in the file is not the next sequential entry, an error will be reported.<br>If you run an inbound compliance map, choose the “Validate Control Number Sequence" option to add an -sc switch to the command line options. | I/O | Optional |
| -sdb | Specifies the maximum number of cached ODBC connections. The default value '0' indicates no ODBC connection caching. | I/O | Optional |
| -t | Specifies the message/transaction set being mapped. | Outbound | Required |

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -td | Specifies the directory in which the bin files are located if the –du switch is not set.<br>Specifies the directory in which the temporary split files are located. | Inbound<br>Outbound | Optional<br>Optional |
| -tm | Writes the elapsed run time to the trace file. | I/O | Optional<br>Optional |
| -u | Does not write EDIFACT UNB and UNG segments. | Outbound | Optional |
| -w | Overwrites all application output files. (The default is to append the application output files.) | Inbound | Optional |
| -xf | Closes map(s) that have been left open in memory. | I/O | Optional |
| -xl | Writes the text transaction log file in expanded field length format. | I/O | Optional |
| -xmi | Uses stdin in place of the EDI file in the map. | Inbound | Optional |
| -xmo | Uses stdout in place of the EDI file in the map. | Outbound | Optional |
| -xmp | Uses a specific memory address in place of the EDI file in the map. | I/O | Optional |
| -z | Zero-fills numeric fields that contain data. | I/O | Optional |

Table 6-2 contains UNIX Run Time parameters that are only for ODBC users.

***Table 6-2: ODBC trading partner parameters/switches***

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -ad | Specifies a DSN connect string to be used in place of a specified DSN connect string embedded in the map file. | I/O | Optional |
| -e1 | Performs the trading partner lookup based on the group sender and receiver. | Inbound | Optional |
| -e2 | Performs the trading partner lookup based on the group and interchange sender. | Inbound | Optional |
| -e3 | Performs the trading partner lookup based on the group and interchange sender and receiver. | Inbound | Optional |

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -e4 | Performs the trading partner lookup based on the group and interchange receiver. | Inbound | Optional |
| -e5 | Performs the trading partner lookup based on interchange sender. | Inbound | Optional |
| -e6 | Performs the trading partner lookup based on the interchange receiver. | Inbound | Optional |
| -e7 | Performs the trading partner lookup based on the interchange sender and receiver. | Inbound | Optional |
| -e8 | Performs a reverse trading partner lookup based on the interchange sender and receiver. | Inbound | Optional |
| -e9 | Performs a reverse trading partner lookup based on the group and interchange sender and receiver. | Inbound | Optional |
| -e10 | Performs a reverse trading partner lookup based on the interchange receiver. | Inbound | Optional |
| -e11 | Performs a reverse trading partner lookup based on the interchange sender. | Inbound | Optional |
| -e12 | Performs a reverse trading partner lookup based on the group and interchange sender. | Inbound | Optional |
| -e13 | Performs a reverse trading partner lookup based on the group and interchange receiver. | Inbound | Optional |
| -e14 | Performs the trading partner lookup based on Batch Sender.<br>Validates interchange and group data against the Trading Partner database during inbound processing, and, if any information does not match, ECRTP writes an error to the log and returns an error in the TA1 Acknowledgment. | Inbound | Optional |
| -e15 | Performs the trading partner lookup based on Batch Receiver. | Inbound | Optional |
| -e16 | Performs the trading partner lookup based on Batch Sender and Receiver. | Inbound | Optional |
| -e17 | Performs the trading partner lookup based on Bin Number. | Inbound | Optional |

| Parameter | Description | Inbound/outbound | Required/optional |
|---|---|---|---|
| -e18 | Performs the trading partner lookup based on Processor Control Number. | Inbound | Optional |
| -e19 | Performs the trading partner lookup based on Batch Sender and Bin Number. | Inbound | Optional |
| -e20 | Performs the trading partner lookup based on Batch Sender and Processor Control Number. | Inbound | Optional |
| -e21 | Performs the trading partner lookup based on Batch Receiver and Bin Number. | Inbound | Optional |
| -e22 | Performs the trading partner lookup based on Batch Receiver and Processor Control Number. | Inbound | Optional |
| -et | Specifies the trace file directory. (*trnn.dat*) | I/O | Optional |
| -sl | Specifies a DSN connect string to be used in place of the log database DSN connect string embedded in the map. | I/O | Required for ODBC Log |
| -st | Specifies a DSN connect string to be used in place of the trading partner database DSN connect string embedded in the map. | I/O | Required for ODBC TP |
| -td | Specifies directory used to make sure information is correctly backed out with Backout and Checkpoint commands. | I/O | Optional |
| -wx | Uses the record in the WIXSET company ID table where the RECORD_NO field is the same as the specified number (in other words, selects a specific company profile). | I/O | Required for ODBC TP |

## Required parameters

Some parameters are required for both inbound and outbound runs, while others are required only for outbound runs or only for inbound runs.

### Outbound required parameters

The normal outbound run has the parameters illustrated below.

```
rmapout <full-path EDI output filename> <map>
```

```
<transaction/message code> -t <message/ transaction>
-dt <full-path trading partner directory> -dg <full
-path map directory>
```

When the trading partner and the map directory are the same, the command line can be shortened by using the –dm directory in their place, as shown below:

```
rmapout <full-path EDI output filename> <map>
<transaction/message code> -t <message/ transaction/>
-dm <full-path trading partner and map directory>
```

The first three parameters are always required and must be in the following order:

- <full-path EDI output filename>
- <map>
- <transaction/message>

The three other required parameters do not have to be in any specific order, but they must be preceded by the appropriate "-letters" flag.

- -t <transaction> – specifies the code of the EDI transaction/message being mapped.
- -dt <directory> – specifies the directory that contains trading partner information.
- -dg <directory> – specifies the directory that contains the generated files – map files, cross reference tables, and log files.

When the trading partner directory and the map directory are the same, the –dm switch can be used in place of both the –dt and –dg switches.

- -dm <directory> – specifies the directory that contains trading partner information and generated files; map files, cross reference tables, and log files.

## Outbound required parameters for ODBC database users

When an ODBC trading partner database is used, –dt <trading partner directory> is replaced by –st <"DSN=<data source name>;uid=<user id>;pwd=<password>">.

When an ODBC log database is used, –sl <"DSN=<data source name>;uid=<user id>; pwd=<password>"> switch must be included. (The DSN pointing to the log database is the only required parameter; uid and pwd are included only if they are required by the database.)

```
 rmapout <full-path EDI output filename> <map>
<transaction/message code>
-t <transaction/message>
-st <trading partner DSN connect string>
-dg <full-path map directory>
-sl <log DSN connect string>
```

- -sl <"DSN Connect String"> – specifies the data source name (DSN) connect string for the ODBC log database that contains the log tables. In addition to the DSN, this connect string must include a uid and pwd if they are required.
- -st <"DSN connect string"> – specifies the data source name (DSN) connect string for the ODBC trading partner database that contains the trading partner and trade agreement tables. In addition to the DSN, this connect string must include a uid and pwd if they are required.

## Inbound required parameters

The normal inbound run has the parameters illustrated below.

```
mapinrun <full-path EDI input filename>
-dt <full-path trading partner Directory>
-dg <full-path map directory>
```

When the trading partner and the map directory are the same, the command line can be shortened by using the –dm directory in their place, as shown below:

```
mapinrun <full-path EDI input filename> -dm <full-path
trading partner/map directory>
```

The first parameter is always required for an inbound run and must be first in order.

```
<full-path EDI input filename>
```

The three other required parameters do not have to be in any specific order, but they must be preceded by the appropriate "-letters" flag.

- -dt <directory> – specifies the directory that contains trading partner information.
- -dg <directory> – specifies the directory that contains the generated files – map files, cross reference tables, and log files.

  When the trading partner directory and the map directory are the same, the –dm switch can be used in place of both the –dt and –dg switches.

- -dm <directory> – specifies the directory that contains trading partner information and generated files; map files, cross reference tables, and log files.

## Inbound required parameters for ODBC database users

When an ODBC trading partner database is used, –dt <trading partner directory> is replaced by –st <"DSN=<data source name>;uid=<user id>;pwd=<password>">.

When an ODBC log database is used, –sl <"DSN=<data source name>;uid=<user id>; pwd=<password>"> switch must be included. (The DSN pointing to the log database is the only required parameter; uid and pwd are included only if they are required by the database.)

```
rmapinrun <full-path EDI input filename> -st <trading
partner DSN connect string> -dg <full-path map
directory> -sl <log DSN connect string>
```

- -st <"DSN connect string"> – specifies the data source name (DSN) connect string for the ODBC trading partner database that contains the trading partner and trade agreement tables. In addition to the DSN, this connect string must include a uid and pwd if they are required.
- -sl <"DSN Connect String"> – specifies the data source name (DSN) connect string for the ODBC log database that contains the log tables. In addition to the DSN, this connect string must include a uid and pwd if they are required.

## Optional parameters

Some optional parameters can be used with both inbound and outbound runs, while other optional parameters can be used for only inbound or only outbound runs.

### Inbound/outbound optional parameters

***Table 6-3: Inbound/outbound optional parameters***

| Parameter | Description |
|---|---|
| -af <full-path map filename> <full-path new filename> | Uses the named "new filename" in place of the named "map filename" imbedded in the map. |

| Parameter | Description |
|---|---|
| -c | Closes the trace file after every write statement. (This is used to ensure that the last trace message is written to disk. This flag impedes processing and should not be used unless a serious problem is encountered and must be debugged.) |
| -du <directory> | Uses the named directory in place of the application directories embedded in the map. |
| -ec | Does not create the transaction log file (*translog.in*, *translog.out*, or *trlog*) or the status file (*status.in* or *status.out*). |
| -ed <directory> | Writes the transaction log file (*translog.in* or *translog.out*), status file (*status.in* or *status.out*), or trace file (*incoming.err*, *outgoing.err*, or *trnn.dat*), to the named directory. (Required for Tandem and Stratus; optional for all other versions.) |
| -ef | Does not create the status file (*status.in* or *status.out*). |
| -el | Uses the named full path file name as the transaction log file name. (Required for Tandem and Stratus; optional for all other versions.) |
| -el <full-path filename> | Uses the named directory for the trace file (*incoming.err* or *outgoing.err*). If this switch is not used, the trace file is placed in the map directory. |
| -et <directory> | Replaces all except the first character in the application file name in the map (excluding the file extension) with the named string variable. This switch is required when multiple copies of the executable are run simultaneously. |
| -eu <string variable> (up to 7 characters) | Replaces all except the first character in the application file name in the map (excluding the file extension) with the named string variable. This switch is required when multiple copies of the executable are run simultaneously. |
| -ev <string variable> (up to 8 characters) | Places the named string variable in front of all application file names in the map. |

| Parameter | Description |
|---|---|
| -id <run ID number> | Uses the specified run ID number instead of looking it up. (Normally, the program looks up the next run ID number in the Run ID table in the log database.) The run ID number can have from one to eight digits. |
| -it | Ignores the trading partner mailbox. If this switch is not set for inbound maps, a copy of the inbound EDI file is placed in the trading partner IN mailbox and no rules are processed unless the trade agreement records have the EDI_OUT flag set. If this switch is not set for outbound maps, the outbound EDI file is placed in the trading partner OUT mailbox. (If a trade agreement mailbox exists and this switch is not set, the trade agreement mailbox will override the trading partner mailbox.) |
| -l | Turns the long trace on, causing trace messages to be written to an error file. (This switch is used for debugging.) |
| -mm <full-path filename> | Uses a temporary memory location in place of the named file imbedded in the map. |
| -mn | Passes the map name extension as part of a command line argument. The Map Name Extension works with other map lookup fields to find a correct map.<br>For map functions with multiple parameters, such as LOADMAP, the correct map name should be found by using current map lookup fields with the newly added Map Name Extension before calling the functions. |
| -mp <full-path filename> <pointer to memory address> <pointer to # of bytes> <pointer to size of memory buffer> | Uses a specified memory location in place of a specified file during map execution. |
| -mx <number> | Keeps a specified number of maps open in memory. |
| -n | Uses the ALL TradePartner if no trading partner match is found in the trading partner file. |

| Parameter | Description |
|---|---|
| -nret | Adds new return codes that provide more information to the calling program. Based on the return value, the user can determine the next step in the process. The return codes reflect the following information:<br>• At least one interchange or group is in error implying a TA1 map should be run.<br>• At least one group or transaction is in error, implying a 997 map should be run.<br>• At least one good transaction is present, implying a translation map should be run. |
| -nz | Maps numeric data literally (including blank fields) as it appears on the map. Previous versions of ECRTP pad numeric values with leading zeros based on field length. |
| -o | Ignores the trade agreement mailbox file name that was set up for routing EDI data. |
| -pf | Uses the map switches in the named batch file on the command line. |
| -r | Specifies the maximum number of cross reference table entries that will be loaded into memory. Anything over the maximum must be accessed from the database. |
| -rb | Ignores the trade agreement mailbox and places the EDI data in the trading partner BAD mailbox on inbound maps and on outbound maps with “Route Bad” selected. |
| -rg | Ignores the trade agreement mailbox and places the EDI data in the trading partner GOOD mailbox on inbound maps and on outbound maps with “Route Good” selected. |

| Parameter | Description |
|---|---|
| -ri | Ignores the trade agreement mailbox and places the EDI data in the trading partner IN mailbox on inbound maps and on outbound maps with “Route In” selected. |
| -ro | Ignores the trade agreement mailbox and places the EDI output in the trading partner OUT mailbox on outbound maps and on inbound maps with “Route Out” selected. |
| -rt | Ignores the trade agreement mailbox and places the EDI data in the trading partner OTHER mailbox on inbound maps and on outbound maps with “Route Other” selected. |
| -s | Does not produce a trace file. |
| -sdb | Specifies the maximum number of cached ODBC connections. The default value '0' indicates no ODBC connection caching. |
| -tm | Writes the elapsed time of execution for the entire run of this command line to the trace file (*incoming.err* file for inbound maps or *outgoing.err* file for outbound maps). |
| -xf | Closes all open maps. |
| -xf <mapname> | Closes the specified map. |
| -xl | Writes the text transaction log file in the expanded field-length format. (See Section 10 for the format of the non-ODBC expanded text transaction log file.) |
| -xmp <pointer to memory address> <pointer to # of bytes> <pointer to size of memory buffer> | Indicates that a specified memory location should be used in place of the EDI file during map execution. |
| -z | Zero-fills numeric fields that contain data. (For inbound maps, numeric fields are zero-filled if the EDI element contains data, and left blank if there is no data. For outbound maps, a zero is produced if a zero is contained in the data field.) |

### Inbound/outbound optional parameters for ODBC database users

***Table 6-4: Inbound/outbound optional parameters for ODBC database users***

| Parameter | Description |
|---|---|
| -ad <“map DSN connect string”> <“new DSN connect string”> | Specifies the named “new DSN connect string” to be used in place of the named “map DSN connect string” in the map file. (Connect strings include the data source name and any other required connection information, such as uid and pwd.) |
| -et <directory> | Uses the named directory for the trace file (trnn.dat where nn is a non-zero run ID number). If this switch is not used, the trace file is placed in the current directory. The run ID number can have from one to eight digits. |
| -sdb <number> | Specifies the maximum number of cached ODBC connections. The default value '0' indicates no ODBC connection caching. |
| -td <directory> | Specifies the named directory to be used with the Backout and Checkpoint commands to make sure that information is correctly backed out. (The -td switch has two other uses; one only for inbound and one only for outbound.) |
| -wx <number> | Uses the record in the wixset company table where the RECORD_NO field is the same as the specified number. (allows a specific company profile to be selected) |

### Outbound-only optional parameters

***Table 6-5: Outbound-only optional parameters***

| Parameter | Description |
|---|---|
| -a | Updates the ISA Out control count field only in the All TradePartner record (customer number = 0) when the interchange envelopes are built. |

| Parameter | Description |
|---|---|
| -db | Does not delete outbound temporary files created when processing multiple files (in other words, when the map is set by the user to separate one input file into several input files). These files are not deleted and can be used to diagnose map flow problems. Use this switch when debugging. |
| -dw <directory> | Uses the named directory as the location of the *wixset.dat* (company) file. |
| -ei <full-path filename> | Uses the named filename in place of the input application filename embedded in the map. |
| -kf | Splits files into multiple files only once when processing multiple files (when the map is set by the user to separate one input file into several input files). |
| -mo < full-path filename> | Uses stdout in place of the named file imbedded in the map. |
| -ne | Does not produce an outbound EDI file. |
| -nt | Does not perform trading partner lookup. (Uses the map passed in on the command line.) |
| -pe | Pads alphanumeric fields with trailing spaces if those spaces are required to meet the minimum length of the element. Numeric fields will be padded with leading 0s if those 0s are required to meet the minimum length of the element.<br>This switch is an optional replacement for `PADEDI` environmental variable. |
| -s3<value> | Uses the specified value to create the optional ST03 element on outbound X12 EDI transactions. |

| Parameter | Description |
|---|---|
| -td <directory> | Writes temporary split files to the named directory. Temporary split files (*pfs.**) are created when the Multiple Files option is selected (when the map is set by the user to separate one input file into several input files). If this switch is not set, the temporary files are written to the current directory. (In earlier versions, they were written to the root directory.) |
| -u | Does not write EDIFACT UNB and UNG segments. |
| -xmo | Uses stdout in place of the EDI file during map execution. |

### Inbound-only optional parameters

***Table 6-6: Inbound-only optional parameters***

| Parameter | Description |
|---|---|
| -ab | Specifies a new full path and file name to be used in place of the BAD EDI file. |
| -ag | Specifies a new full path and file name to be used in place of the GOOD EDI file. |
| -as | Checks that each ST Transaction Control Number in a GS to GE is greater than the previous ST Transaction Control Number. Validation assumes control numbers appear in ascending order. |
| -b | Does not save the rejected EDI messages/transactions into the badtrans.nmt file. |
| -clz | Flags leading zeros in numeric X12 fields as an error on HIPAA compliance maps.<br>An error is flagged if leading zeros are not used to meet the minimum length requirement of that particular element. If the leading zeros are necessary to meet the minimum length of the element, no error is generated. This option does not check leading zeros on non-compliance maps. |

| Parameter | Description |
|---|---|
| -cu | Checks for unique control numbers within a transaction. Control numbers can now occur in any sequence, as long as they are unique.<br>If duplicate control numbers are found, ECRTP logs a 6054 error. Use only one of the switches (-as or -cu) for any map run. If both switches are present, ECRTP defaults to the last switch encountered. |
| -ee <number of characters> | Ends processing of the EDI file after it processes the specified number of characters. (Useful when processing is done by VANS who charge by the byte). |
| -eo <full path filename> | Uses the named full path file name in place of the output application file name embedded in the map. |
| -es <number of characters> | Starts processing the EDI file after it has read the specified number of characters. (Useful when processing is done by VANS who charge by the byte.) |
| -k | Sets compliance checking for:<br>• Missing mandatory segments<br>• Exceeding loop counts<br>• Exceeding segment counts<br>• Segments out of sequence<br>• Exceeding Standards definition for number of elements in a segment |
| -m <mapname> | Runs the map with the specified map name without referencing company or trading partner files. (<mapname> is the file name of the map without the extension. For example, 837IN is the <mapname> for the map file *837IN.map*.). |
| -mi <full path filename> | Uses stdin in place of the named file imbedded in the map. |
| -ncb | Indicates that the EDI file to be processed is an NCPDP batch file. Required for all inbound NCPDP files unless -nct is active. |

| Parameter | Description |
|---|---|
| -nct | Indicates that the EDI file to be processed is an NCPDP telecommunications file. Required for all inbound NCPDP files unless -ncb is active. |
| -nt | Does not perform trading partner lookup. (Uses the map passed in on the command line.) |
| -s3 | Processes the optional 12 ST03 element in incoming EDI data. |
| -sc | Validates the sequence of the Interchange (ISA) and Group (GS) control numbers. If you use this command, RTP checks the current control number against the Trading Partner database to validate the entry. If the entry in the file is not the next sequential entry, an error will be reported.<br>If you run an inbound compliance map, choose the “Validate Control Number Sequence" option to add an -sc switch to the command line options. |
| -td <directory> | Writes the bin files to the named directory if the –du switch is not set. (All bin files start with bin.) |
| -w | Overwrites all output map files. (The default is to append the map files.). |
| -xmi | Uses stdin in place of the named EDI file during map execution. |

### Inbound-only optional parameters – trading partner lookup switches

*Table 6-7: Trading partner lookup switches – Inbound-only*

| Parameter | Description |
|---|---|
| -er | Performs a reverse trading partner lookup based on the group receiver code. |
| -ol | Triggers a series of look ups against the Trading Partner database when the ECMap/EC Gateway Log is used as input.<br>If a trading partner match is found, ECRTP uses the entry to populate the EDI envelope. |

### Inbound-only optional parameters for ODBC database users – trading partner lookup switches

*Table 6-8: ODBC trading partner lookup switches – inbound-only*

| Parameter | Description |
|---|---|
| -e1 | Performs the trading partner lookup based on the group sender and receiver codes. |
| -e2 | Performs the trading partner lookup based on the group sender code and the interchange sender code and qualifier. |
| -e3 | Performs the trading partner lookup based on the group sender and receiver codes and the interchange sender and receiver codes and qualifiers. |
| -e4 | Performs the trading partner lookup based on the group receiver code and the interchange receiver code and qualifier. |
| -e5 | Performs the trading partner lookup based on the interchange sender code and qualifier. |
| -e6 | Performs the trading partner lookup based on the interchange receiver code and qualifier. |
| -e7 | Performs the trading partner lookup based on the interchange sender and receiver codes and qualifiers. |
| -e8 | Performs a reverse trading partner lookup based on the interchange sender and receiver codes and qualifiers. |

| Parameter | Description |
|---|---|
| -e9 | Performs a reverse trading partner lookup based on the group sender and receiver codes and the interchange sender and receiver codes and qualifiers. |
| -e10 | Performs a reverse trading partner lookup based on the interchange receiver code and qualifier. |
| -e11 | Performs a reverse trading partner lookup based on the interchange sender code and qualifier. |
| -e12 | Performs a reverse trading partner lookup based on the group sender code and the interchange sender code and qualifier. |
| -e13 | Performs a reverse trading partner lookup based on the group receiver code and the interchange receiver code and qualifier. |
| -e14 | Performs the trading partner lookup based on Batch Sender.<br>Validates interchange and group data against the Trading Partner database during inbound processing, and, if any information does not match, ECRTP writes an error to the log and returns an error in the TA1 Acknowledgment. |
| -e15 | Performs the trading partner lookup based on Batch Receiver. |
| -e16 | Performs the trading partner lookup based on Batch Sender and Receiver. |
| -e17 | Performs the trading partner lookup based on Bin Number. |
| -e18 | Performs the trading partner lookup based on Processor Control Number. |
| -e19 | Performs the trading partner lookup based on Batch Sender and Bin Number. |
| -e20 | Performs the trading partner lookup based on Batch Sender and Processor Control Number. |
| -e21 | Performs the trading partner lookup based on Batch Receiver and Bin Number. |

| Parameter | Description |
|---|---|
| -e22 | Performs the trading partner lookup based on Batch Receiver and Processor Control Number. |

CHAPTER 7

# Non-ODBC Database and File Formats

# Trading partner files

Trading partner information is stored in three files: the company data file, the trading partner data file, and the trade agreement data file. Information is logged by ECRTP in two files: the transaction log and the trace file. In addition, ECRTP produces a status file if errors occur during processing.

Trading partner files include information about the user's company, the user's trading partners, and the trade agreements that link trading partners to specific maps. The -t runtime switches indicates that an ODBC trading partner database will be used.

## Company data file (*wixset.dat*)

***Table 7-1: Company data file information***

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 1. | wix_company_name | Character | 35 |
| 2. | <filler> | Character | 12 |
| 3. | wix_gsid | Character | 35 |
| 4. | wix_idqual | Character | 4 |
| 5. | wix_idcode | Character | 35 |
| 6. | wix_auth_qual | Character | 2 |
| 7. | wix_auth_code | Character | 10 |
| 8. | wix_secu_qual | Character | 2 |
| 9. | wix_secu_code | Character | 10 |
| 10. | <filler> | Character | 47 |
| 11. | wix_sndr_route | Character | 14 (35 for EDIFACT Syntax 4) |
| 12. | wix_sndr_subid | Character | 35 (only for EDIFACT Syntax 4) |
| 13. | wix_app_snd_ql | Character | 4 |

## Trading partner file (*customer.mdb*)

***Table 7-2: Trading partner file information***

| Number | Field name | Field type | Field precision | Decimals |
|---|---|---|---|---|
| 1. | CUSTNO | Character | 35 | |
| 2. | TYP_OWNER | Character | 1 (Reserved for Future) | |

| Number | Field name | Field type | Field precision | Decimals |
|---|---|---|---|---|
| 3. | NAME | Character | 35 | |
| 4. | IDQUAL | Character | 4 | |
| 5. | IDCODE | Character | 35 | |
| 6. | AUTH_QUAL | Character | 2 | |
| 7. | AUTH_CODE | Character | 10 | |
| 8. | SECU_QUAL | Character | 2 | |
| 9. | SECU_CODE | Character | 10 | |
| 10. | GSID | Character | 35 | |
| 11. | SHIPQUAL | Character | 2 | |
| 12. | SHIPIDEN | Character | 15 | |
| 13 | BILLQUAL | Character | 2 | |
| 14 | BILLIDEN | Character | 15 | |
| 15 | ADDR1 | Character | 35 | |
| 16 | ADDR2 | Character | 35 | |
| 17. | CITY | Character | 19 | |
| 18. | STATE | Character | 15 | |
| 19. | COUNTRY | Character | 25 | |
| 20. | ZIP | Character | 9 | |
| 21. | CONTACT1 | Character | 35 | |
| 22. | TELEPHONE1 | Character | 22 | |
| 23. | CONTACT2 | Character | 35 | |
| 24. | TELEPHONE2 | Character | 22 | |
| 25. | ISA_IN_NO | Character | 14 | |
| 26. | ISA_OUT_NO | Character | 14 | |
| 27. | SND_GSID | Character | 35 | |
| 28. | SND_IDQUAL | Character | 4 | |
| 29 | SND_IDCODE | Character | 35 | |
| 30. | SUB_DELIMT | Character | 3 | |
| 31. | ELE_DELIMT | Character | 3 | |
| 32. | SEG_DELIMT | Character | 3 | |
| 33. | RELEASE_CH | Character | 3 | |
| 34. | X12_REPEAT | Character | 3 | |
| 35. | DEL_CODE | Character | 1 (Reserved for Future) | |
| 36. | EDIF_SUBDL | Character | 3 | |
| 37. | EDIF_ELEDL | Character | 3 | |
| 38. | EDIF_SEGDL | Character | 3 | |
| 39. | EDIF_RELCH | Character | 3 | |

| Number | Field name | Field type | Field precision | Decimals |
|---|---|---|---|---|
| 40. | EDIF_REPEA | Character | 3 | |
| 41. | HL7_SEGDL | Character | 3 | |
| 42. | HL7_ELEDL | Character | 3 | |
| 43. | HL7_SUBDL | Character | 3 | |
| 44. | HL7_SUBSUB | Character | 3 | |
| 45. | HL7_RELCH | Character | 3 | |
| 46. | HL7_REPEAT | Character | 3 | |
| 47. | EXPORT_FLG | Character | 1 | |
| 48. | MBOX_NAME | Character | 35 | |
| 49. | MAILBOX | Character | 100 | |
| 50. | CURR_FMT | Character | 1 | |
| 51. | POS_LTR | Character | 1 | |
| 52. | SNDR_ROUTE | Character | 14 (35 for EDIFACT Syntax 4) | |
| 53. | SNDR_SUBID | Character | 35 (only for EDIFACT Syntax 4) | |
| 54. | RCVR_ROUTE | Character | 14 (35 for EDIFACT Syntax 4) | |
| 55. | RCVR_SUBID | Character | 35 (only for EDIFACT Syntax 4) | |
| 56. | APP_SND_QL | Character | 4 | |
| 57. | APP_RCV_QL | Character | 4 | |
| 58. | TPKEY | Numeric | 19 | 5 |

## Trade agreement file (*tradstat.mdb*)

***Table 7-3: Trade agreement file information***

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 1. | CUSTNO | Character | 35 |
| 2. | MAP_TRAN | Character | 6 |
| 3. | ST03 | Character | 35 |
| 4. | DIR | Character | 3 |
| 5. | STAT | Character | 1 |
| 6. | VERS | Character | 12 |
| 7. | TBCODE | Character | 60 |
| 8. | MBOX_NAME | Character | 35 |
| 9. | DEST | Character | 100 |

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 10. | FILE | Character | 30 |
| 11. | GS_NO | Character | 14 |
| 12. | ISA_TYPE | Character | 5 |
| 13. | SERV_CODE | Character | 6 |
| 14. | <filler> | Character | 1 |
| 15. | RCV_GSID | Character | 35 |
| 16. | FCV_IDQUAL | Character | 4 |
| 17. | RCV_IDCODE | Character | 35 |
| 18. | ACK_RQSTD | Character | 1 |
| 19. | ACK_RQSTD2 | Character | 1 |
| 20. | EDI_OUT | Character | 1 |
| 21. | DAYS | Character | 2 |
| 22. | HOURS | Character | 2 |
| 23. | MINUTES | Character | 2 |
| 24. | SECONDS | Character | 2 |
| 25. | APPL_REF | Character | 14 |
| 26. | ACK_MSG | Character | 1 |
| 27. | ACK_INTCH | Character | 1 |
| 28. | RCVR_ROUTE | Character | 14 (35 for EDIFACT Syntax 4) |
| 29. | RCVR_SUBID | Character | 35 (only for EDIFACT Syntax 4) |
| 30. | APP_RCV_QL | Character | 4 |
| 31. | PROC_PRIOR | Character | 1 |
| 32. | COMM_AGM | Character | 35 |
| 33. | APP_PSWD | Character | 14 |
| 34. | ASSOC_CODE | Character | 6 |
| 35. | CNT_AG1 | Character | 3 |
| 36. | CLIST_VER | Character | 6 (only for EDIFACT Syntax 4) |
| 37. | MSG_TYPE | Character | 6 (only for EDIFACT Syntax 4) |
| 38. | MSG_SUBID | Character | 14 (only for EDIFACT Syntax 4) |
| 39. | MSG_SUBVER | Character | 3 (only for EDIFACT Syntax 4) |
| 40. | MSG_SUBREL | Character | 3 (only for EDIFACT Syntax 4) |

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 41. | CNT_AG2 | Character | 3 (only for EDIFACT Syntax 4) |
| 42. | MSG_IMPID | Character | 14(only for EDIFACT Syntax 4) |
| 43. | MS_IMPVER | Character | 3 (only for EDIFACT Syntax 4) |
| 44. | MSG_IMPREL | Character | 3 (only for EDIFACT Syntax 4) |
| 45. | CNT_AG3 | Character | 3 (only for EDIFACT Syntax 4) |
| 46. | SCEN_ID | Character | 14 (only for EDIFACT Syntax 4) |
| 47. | SCEN_VER | Character | 3 (only for EDIFACT Syntax 4) |
| 48. | SCEN_REL | Character | 3 (only for EDIFACT Syntax 4) |
| 49. | SCEN_AG4 | Character | 3 (only for EDIFACT Syntax 4) |
| 50. | STD_TYPE | Character | 2 |
| 51. | APP_RCV_QL | Character | 4 |
| 52. | TRADKEY | Numeric | 10 |

## Log files

Information is logged by ECRTP in two files: the transaction log and the trace file. In addition, ECRTP produces a status file if errors occur during processing.

### Text transaction log files

When the user has turned on logging, the text transaction logs are stored in files with fixed-length records in the directory where the map is executed. Unless the user specifies otherwise, information in these files is appended upon execution of the map. (However, the user can cause the files to be overwritten by using a run time switch.)

Based on the Log Type selected, the text transaction log files can be written out in either an expanded format (Explanded Text Log) or a non-expanded format (Text Log). You should always choose the expanded format; the non-expanded format is included only to support earlier versions of the software and cannot be used to generate a 997 functional acknowledgement.

The text transaction log files are written to translog.out for outbound and any-to-any maps and translog.in for inbound and Web maps. You can edit these log files with a text editor to provide report-style information about the transaction run.

## Transaction log files (translog.in and translog.out)

### Non-expanded format

***Table 7-4: Transaction log files – non-expanded format***

| Number | Field name | Field description | Field type | Field length |
|---|---|---|---|---|
| 1. | TYPE | Record Type Flag | Character | 1 |
| 2. | RUN_DATE | DATE: YYMMDD | Date | 6 |
| 3. | RUN_TIME | Time: HHMMSS | Time | 6 |
| 4. | TRANS_CODE | Transaction/Message Letter Code | Character | 2 |
| 5. | TRANS_NAME | Message/Transaction Set Number | Character | 6 |
| 6. | TPTNER_ID | Trading Partner Number | Character | 35 |
| 7. | VERSION | Standard Version | Character | 12 |
| 8. | ISA_TYPE | Standard Type | Character | 5 |
| 9. | INTERCHANG | Interchange Control Number | Character | 35 |
| 10. | GROUP_NO | Group Control Number | Character | 35 |
| 11. | TRANS_NO | Transaction/Message Control Number + Transaction/Message count | Character | 35 |
| 12. | APP_RCV_CD | Group Receiver Code | Character | 35 |
| 13. | APP_SND_CD | Group Sender Code | Character | 35 |
| 14. | SEND_CODE | Interchange Sender Code | Character | 35 |
| 15. | RECV_CODE | Interchange Receiver Code | Character | 35 |
| 16. | RECV_QUAL | Interchange Receiver Qualifier | Character | 4 |
| 17. | SEND_QUAL | Interchange Sender Qualifier | Character | 4 |
| 18. | ERRORS | Number of non-fatal errors this transaction/message | Number | 5 |
| 19. | STAT | This ST or SE Status Code | Character | 1 |
| 20. | BYTE_COUNT | Byte Offset in input file | Character | 9 |

| Number | Field name | Field description | Field type | Field length |
|---|---|---|---|---|
| 21. | DIR | Direction indicator – IN/OUT/PRT/CMP | Character | 33 |
| 22. | FLOW_LEVEL | User-defined flow level number | Character | 5 |
| 23. | RECORD_NAM | Record Name | Character | 10 |
| 24. | RECORD_NO | Record sequence number | Character | 6 |
| 25. | FIELD_NAME | Field Name | Character | 15 |
| 26. | SEGMENT | Segment identifier | Character | 3 |
| 27. | SEG_COUNT | Original Segment count | Number | 6 |
| 28. | ELEMENT | Element sequence number | Character | 2 |
| 29. | SUBELEM | Subelement identifier | Character | 2 |
| 30. | SEV_CODE | Message severity | Character | 2 |
| 31. | MSG_NO | Message number | Character | 5 |
| 32. | MSG_TEXT | Message description | Character | 100 |
| 33. | FILENAME | Full-path EDI file name | Character | 160 |
| 34. | FIELDVAL | Field Value | Character | 30 |

## Transaction log files (translog.in and translog.out)

### Expanded format

***Table 7-5: Transaction log files – expanded format***

| Number | Field name | Field description | Field type | Field length |
|---|---|---|---|---|
| 1. | RUN_ID | Run ID number | Numeric | 9 |
| 2. | TYPE | Record Type | Character | 1 |
| 3. | RUN_DATE | Run Date Time: YYYYMMDDHHMMSS | Date | 14 |
| 4. | ACKBY_DATE | Acknowledgement Date Time: YYYYMMDDHHMMSS | Date | 14 |
| 5. | TRANS_CODE | Message/Transaction Letter Code | Character | 2 |
| 6. | TPANS_NAME | Message/Transaction Letter Code | Character | 6 |
| 7. | TPTNER_ID | Trading Partner Number | Character | 35 |
| 8. | VERSON | Standard Version | Character | 12 |
| 9. | ISA_TYPE | Standard Type | Character | 5 |
| 10. | INTERCHANG | Interchange Control Number | Character | 35 |
| 11. | GROUP_NO | Group Control Number | Character | 35 |
| 12. | TRANS_NO | Transaction/Message Control Number + Transaction/Message count | Number | 35 |

| Number | Field name | Field description | Field type | Field length |
|---|---|---|---|---|
| 13. | APP_RCV_CD | Group Receiver Code | Character | 35 |
| 14. | APP_SND_CD | Group Sender Code | Character | 35 |
| 15. | RECV_CODE | Interchange Receiver Code | Character | 35 |
| 16. | SEND_CODE | Interchange Sender Code | Character | 35 |
| 17. | RECV_QUAL | Interchange Receiver Qualifier | Character | 4 |
| 18. | SEND_QUAL | Interchange Sender Qualifier | Character | 4 |
| 19. | ERRORS | Count of total error messages | Number | 10 |
| 20. | STAT | Status Code (See following chart for values) | Character | 1 |
| 21. | BYTE_COUNT | This ST or SE Status Code | Character | 9 |
| 22. | DIR | Direction indicator – IN/OUT/PRT/CMP | Character | 3 |
| 23. | FLOW_LEVEL | User-defined flow level number | Character | 5 |
| 24. | RECORD_NAM | Record Name | Character | 10 |
| 25. | RECORD_NO | Record sequence number | Character | 6 |
| 26. | FIELD_NAME | Field Name | Character | 15 |
| 27. | SEGMET | Segment identifier | Character | 3 |
| 28. | SEG_COUNT | Count of Segment in Transaction/Message | Number | 10 |
| 29. | ELEMENT | Element sequence number | Character | 2 |
| 30. | SUBELEM | Subelement identifier | Character | 2 |
| 31. | SEV_CODE | Message severity | Character | 2 |
| 32. | MSG_NO | Message number | Character | 5 |
| 33. | MSG_TEXT | Message description | Character | 100 |
| 34. | FILENAME | Full-path EDI File Name | Character | 160 |
| 35. | FIELDVAL | Field Value | Character | 30 |
| 36. | USER_IDENT | User Field | Character | 35 |
| 37. | ACK_EXPECT | Acknowledgement Expected Flag | Character | 1 |
| 38. | TR_ACK_TYP | Filler | Character | 1 |
| 39. | T_P_IND | Test/Production Indicator - T or P | Character | 1 |
| 40. | TRANS_CNT | Count of Transactions/Messages | Number | 10 |
| 41 | FILEOFFSET | EDI Input File/Output File Offset | Number | 10 |
| 42 | RCOUNT | Field for record manipulation | Number | 1 |
| 43 | SNDR_ROUTE | Interchange-level internal sender ID code | Character | 14 |
| 44 | SNDR_SUBID | Interchange-level internal sender sub ID code | Character | 35 |

| Number | Field name | Field description | Field type | Field length |
|---|---|---|---|---|
| 45 | RCVR_ROUTE | Interchange-level internal receiver ID code | Character | 14 |
| 46 | RCVR_SUBID | Interchange-level internal receiver sub-ID code | Character | 35 |
| 47 | APPL_REF | Name of messages contained in UNB envelope | Character | 14 |
| 48 | PROC_PRIOR | Processing priority code | Character | 1 |
| 49 | COMM_AGM | Interchange agreement identifier | Character | 35 |
| 50 | APP_SND_QL | Group-level sender code qualifier | Character | 4 |
| 51 | APP_RCV_QL | Group-level receiver code qualifier | Character | 4 |
| 52 | ASSOC_CODE | Association-assigned code | Character | 6 |
| 53 | APP_PSWD | Application password | Character | 14 |
| 54 | CLIST_VER | Code list directory version number | Character | 6 |
| 55 | MSG_TYPE | Message type sub-function identifier | Character | 6 |

The following chart displays possible Status Code values and the definition for each. Status Code is a field in the transaction log file.

*Table 7-6: Status code information*

| Status codes | Definitions |
|---|---|
| W | Wrote ST or SE |
| S | Skipped write |
| A | User abort |
| T | Unknown Trading Partner |
| U | Stop run |
| F | Fatal error |
| D | Inbound destination file transfer |
| E | Other error |

## Trace files (incoming.err and outgoing.err)

Trace files are created during the processing of the map and placed in the same directory as the map files (–dg <directory> switch). The trace file for inbound and web maps is placed in the incoming.err file, and the trace file for outbound and any-to-any maps is placed in the outgoing.err file. Error messages are created in the trace file whenever the execution of the map (either inbound or outbound) encounters an error. Based on the Trace Type selected, the trace file can be short or long. The trace files have a free text format.

## Status file (status.in and status.out)

When ECRTP encounters an error, it produces a status file - status.in for inbound maps or status.out for outbound maps. The user can quickly and easily check to see whether errors occurred during a map run by checking for the existence of a status file. If no status.out or status.in file exists in the executables directory, then no errors occurred. If a file does exist, it contains an error code with an error count. The chart below lists the definition of each of the possible error codes. The creation of status.in and status.out can be turned off with the -ef switch. In addition to creating a status file, ECRTP exits with a numeric return value. See the table below for the meaning of the return values:

*Table 7-7: Status file information*

| Return value | Error code | Definition |
|---|---|---|
| 0 | Wrote ST or SE | No errors |
| 1 | W## - ## | Errors but no transaction skipped |
| 2 | BADTRAN W## | Transaction skipped with ## errors |
| 3 | UABORT W## | User Abort Rule with ## errors |

| Return value | Error code | Definition |
|---|---|---|
| 4 | USTOP W## | User Stop Rule and ## errors |
| 5 | EFATAL W## | Fatal error stop and ## errors |

When application programs are linked to the RTP DLL files, no status file is produced. However, the status of the inbound or outbound run is returned to the calling program as a number (0 thru 5). This number corresponds to a return value in the chart above - the same value that is returned by the executables.

CHAPTER 8

# ODBC Database Table Formats

# How ODBC trading partner data is stored

ODBC trading partner data is stored in three tables in the trading partner database – the company table, the trading partner table, and the trade agreement table. Log information is stored in the log database – in the transaction log table and the trace file. The ODBC log database also has an error table and a run ID table. Like the non-ODBC log, the ECRTP produces a status file if errors occur during processing.

## Trading partner database tables

The three ODBC trading partner database tables contain information about the company, the trading partners, and the trade agreements that link trading partners to specific maps. These tables can be created during map development if an ODBC link has been established to the UNIX computer from the PC. The -st runtime switch indicates that an ODBC trading partner database is being used.

**Note** The TPKEY field in the trading partner table (TP) and the TRADKEY field in the trade agreement table (TRADSTAT) should be AUTOINCREMENT fields. If they are not, a provision must be made to assign a unique numeric value to these fields every time a record is inserted into one of these tables.

### Company table (WIXSET)

***Table 8-1: Company table***

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 1. | RECORD_NO | SQL_SMALLINT | 4 |
| 2. | GSID | SQL_VARCHAR | 35 |
| 3. | NAME | SQL_VARCHAR | 35 |
| 4. | IDQUAL | SQL_VARCHAR | 4 |
| 5. | IDCODE | SQL_VARCHAR | 35 |
| 6. | AUTH_QUAL | SQL_VARCHAR | 2 |
| 7. | AUTH_CODE | SQL_VARCHAR | 10 |
| 8. | SECU_QUAL | SQL_VARCHAR | 2 |
| 9. | SECU_CODE | SQL_VARCHAR | 10 |
| 10. | SNDR_ROUTE | SQL_VARCHAR | 14 |

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 11. | SNDR_SUBID | SQL_VARCHAR | 35 |
| 12. | APP_SND_QL | SQL_VARCHARr | 4 |

## Trading partner table (TP)

***Table 8-2: Trading partner table***

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 1. | CUSTNO | SQL_VARCHAR | 35 |
| 2. | <filler> | SQL_VARCHAR | 1 |
| 3. | NAME | SQL_VARCHAR | 35 |
| 4. | IDCODE | SQL_VARCHAR | 35 |
| 5. | AUTH_QUAL | SQL_VARCHAR | 2 |
| 6. | AUTH_CODE | SQL_VARCHAR | 10 |
| 7. | SECU_QUAL | SQL_VARCHAR | 2 |
| 8. | SECU_CODE | SQL_VARCHAR | 10 |
| 9. | GSID | SQL_VARCHAR | 35 |
| 10. | SHIPQUAL | SQL_VARCHAR | 2 |
| 11. | SHIPIDEN | SQL_VARCHAR | 15 |
| 12. | BILLQUAL | SQL_VARCHAR | 2 |
| 13. | BILLIDEN | SQL_VARCHAR | 15 |
| 14. | ADDR1 | SQL_VARCHAR | 35 |
| 15. | ADDR2 | SQL_VARCHAR | 35 |
| 16. | CITY | SQL_VARCHAR | 19 |
| 17. | STATE | SQL_VARCHAR | 15 |
| 18. | COUNTRY | SQL_VARCHAR | 25 |
| 19. | ZIP | SQL_VARCHAR | 9 |
| 20. | CONTACT1 | SQL_VARCHAR | 35 |
| 21. | TELEPHONE1 | SQL_VARCHAR | 22 |
| 22. | CONTACT2 | SQL_VARCHAR | 35 |
| 23. | TELEPHONE2 | SQL_VARCHAR | 22 |
| 24. | ISA_IN_NO | SQL_VARCHAR | 14 |
| 25. | ISA_OUT_NO | SQL_VARCHAR | 14 |
| 26. | SND_GSID | SQL_VARCHAR | 35 |
| 27. | SND_IDQUAL | SQL_VARCHAR | 4 |
| 28. | SND_IDCODE | SQL_VARCHAR | 35 |
| 29. | SUB_DELIMT | SQL_VARCHAR | 3 |
| 30. | ELE_DELIMT | SQL_VARCHAR | 3 |

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 31. | SEG_DELIMT | SQL_VARCHAR | 3 |
| 32. | RELEASE_CH | SQL_VARCHAR | 3 |
| 33. | X12_REPEAT | SQL_VARCHAR | 3 |
| 34. | <filler> | SQL_VARCHAR | 1 |
| 35. | EDIF_SUBDL | SQL_VARCHAR | 3 |
| 36. | EDIF_ELEDL | SQL_VARCHAR | 3 |
| 37. | EDIF_SEGDL | SQL_VARCHAR | 3 |
| 38. | EDIF_RELCH | SQL_VARCHAR | 3 |
| 39. | EDIF_REPEA | SQL_VARCHAR | 3 |
| 40. | HL7_SEGDL | SQL_VARCHAR | 3 |
| 41. | HL7_ELEDL | SQL_VARCHAR | 3 |
| 42. | HL7_SUBDL | SQL_VARCHAR | 3 |
| 43. | HL7_SUBS | SQL_VARCHAR | 3 |
| 44. | HL7_RELCH | SQL_VARCHAR | 3 |
| 45. | HL7_REPEAT | SQL_VARCHAR | 3 |
| 46. | EXPORT_FLG | SQL_VARCHAR | 1 |
| 47. | MBOX_NAME | SQL_VARCHAR | 35 |
| 48. | MAILBOX | SQL_VARCHAR | 100 |
| 49. | CURR_FMT | SQL_VARCHAR | 1 |
| 50. | POS_LTR | SQL_VARCHAR | 1 |
| 51. | SNDR_ROUTE | SQL_VARCHAR | 14 |
| 52. | SNDR_SUBID | SQL_VARCHAR | 35 |
| 53. | RCVR_ROUTE | SQL_VARCHAR | 14 |
| 54. | RCVR_SUBID | SQL_VARCHAR | 35 |
| 55. | APP_SND_QL | SQL_VARCHAR | 4 |
| 56. | APP_RCV_QL | SQL_VARCHAR | 4 |
| 57. | TPKEY | SQL_INTEGER | 10 |

## Trade agreement table (TRADSTAT)

***Table 8-3: Trade agreement table***

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 1. | CUSTNO | SQL_VARCHAR | 35 |
| 2. | MAP_TRAN | SQL_VARCHAR | 6 |
| 3. | ST03 | SQL_VARCHAR | 35 |
| 4. | DIR | SQL_VARCHAR | 3 |
| 5. | STAT | SQL_VARCHAR | 1 |

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 6. | VERS | SQL_VARCHAR | 12 |
| 7. | TBCODE | SQL_VARCHAR | 60 |
| 8. | MBOX_NAME | SQL_VARCHAR | 35 |
| 9. | DEST | SQL_VARCHAR | 100 |
| 10. | FILE | SQL_VARCHAR | 30 |
| 11. | GS_NO | SQL_VARCHAR | 14 |
| 12. | ISA_TYPE | SQL_VARCHAR | 5 |
| 13. | SERV_CODE | SQL_VARCHAR | 6 |
| 14. | <filler> | SQL_VARCHAR | 1 |
| 15. | RCV_GSID | SQL_VARCHAR | 35 |
| 16. | RCV_IDQUAL | SQL_VARCHAR | 2 |
| 17. | RCV_IDCODE | SQL_VARCHAR | 35 |
| 18. | ACK_RQSTD | SQL_VARCHAR | 1 |
| 19. | ACK_RQSTD2 | SQL_VARCHAR | 1 |
| 20. | EDI_OUT | SQL_VARCHAR | 1 |
| 21. | DAYS | SQL_VARCHAR | 2 |
| 22. | HOURS | SQL_VARCHAR | 2 |
| 23. | MINUTES | SQL_VARCHAR | 2 |
| 24. | SECONDS | SQL_VARCHAR | 1 |
| 25. | APPL_REF | SQL_VARCHAR | 14 |
| 26. | ACK_MSG | SQL_VARCHAR | 1 |
| 27. | ACK_INTCH | SQL_VARCHAR | 1 |
| 28. | RCVR_ROUTE | SQL_VARCHAR | 14 |
| 29. | RCVR_SUBID | SQL_VARCHAR | 35 |
| 31. | PROC_PRIOR | SQL_VARCHAR | 1 |
| 32. | COMM_AGM | SQL_VARCHAR | 35 |
| 33. | APP_PSWD | SQL_VARCHAR | 14 |
| 34. | ASSOC_CODE | SQL_VARCHAR | 6 |
| 35. | CNT_AG1 | SQL_VARCHAR | 3 |
| 36. | CLIST_VER | SQL_VARCHAR | 6 |
| 37. | MSG_TYPE | SQL_VARCHAR | 4 |
| 38. | MSG_SUBID | SQL_VARCHAR | 14 |
| 39. | MSG_SUBVER | SQL_VARCHAR | 3 |
| 40. | MSG_SUBREL | SQL_VARCHAR | 3 |
| 41. | CNT_AG2 | SQL_VARCHAR | 3 |
| 42. | MSG_IMPID | SQL_VARCHAR | 14 |
| 43. | MSG_IMPVER | SQL_VARCHAR | 3 |

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 44 | MSG_IMPREL | SQL_VARCHAR | 3 |
| 45 | CNT_AG3 | SQL_VARCHAR | 3 |
| 46 | SCEN_ID | SQL_VARCHAR | 14 |
| 47 | SCEN_VER | SQL_VARCHAR | 3 |
| 48. | SCEN_REL | SQL_VARCHAR | 3 |
| 49. | CNT_AG4 | SQL_VARCHAR | 3 |
| 50. | STD_TYPE | SQL_VARCHAR | 2 |
| 51 | APP_RCV_QL | SQL_VARCHAR | 4 |
| 52 | TRADKEY | SQL_INTEGER | 10 |

## Log database tables

The log tables contain information that is written to the log tables during processing. The log database includes the transaction log table, the trace file, the run ID table, and an error table. The -sl runtime switch indicates that an ODBC log database is being used.

**Note** The AFLD field in the log table (TRLOG) should be an AUTOINCREMENT field. If it is not, a provision must be made to assign a unique numeric value to this field every time a record is inserted into this table.

### Transaction log (TRLOG)

***Table 8-4: Transaction log***

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 1 | AFLD | AUTOINCREMENT | 10 |
| 2 | RUN_ID | SQL_BIG_INT | 9 |
| 3 | TYP | SQL_VARCHAR | 1 |
| 4 | RUN_DATE | SQL_TIMESTAMP | 14 |
| 5 | ACKBY_DATE | SQL_TIMESTAMP | 14 |
| 6 | TRANS_CODE | SQL_VARCHAR | 2 |
| 7 | TRANS_NAME | SQL_VARCHAR | 6 |
| 8 | TPTNER_ID | SQL_VARCHAR | 35 |
| 9 | VERSION | SQL_VARCHAR | 12 |
| 10 | ISA_TYPE | SQL_VARCHAR | 5 |
| 11 | INTERCHANG | SQL_VARCHAR | 35 |
| 12 | GROUP_NO | SQL_VARCHAR | 35 |

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 13 | TRANS_NO | SQL_VARCHAR | 35 |
| 14 | APP_RCV_CD | SQL_VARCHAR | 35 |
| 15 | APP_SND_CD | SQL_VARCHAR | 35 |
| 16 | RECV_CODE | SQL_VARCHAR | 35 |
| 17 | SEND_CODE | SQL_VARCHAR | 35 |
| 18 | RECV_QUAL | SQL_VARCHAR | 4 |
| 19 | SEND_QUAL | SQL_VARCHAR | 4 |
| 20 | ERRORS | SQL_BIGINT | 10 |
| 21 | STAT | SQL_VARCHAR | 1 |
| 22 | BYTE_COUNT | SQL_BIGINT | 10 |
| 23 | DIR | SQL_VARCHAR | 3 |
| 24 | FLOW_LEVEL | SQL_VARCHAR | 5 |
| 25 | RECORD_NAM | SQL_VARCHAR | 10 |
| 26 | RECORD_NO | SQL_VARCHAR | 6 |
| 27 | FIELD_NAME | SQL_VARCHAR | 15 |
| 28 | SEGMENT | SQL_VARCHAR | 3 |
| 29 | SEG_COUNT | SQL_INTEGER | 10 |
| 30 | ELEMENT | SQL_VARCHAR | 2 |
| 31 | SUBELEM | SQL_VARCHAR | 2 |
| 32 | SEV_CODE | SQL_VARCHAR | 2 |
| 33 | MSG_NO | SQL_VARCHAR | 5 |
| 34 | MSG_TEXT | SQL_VARCHAR | 100 |
| 35 | FILENAME | SQL_VARCHAR | 160 |
| 36 | FIELDVAL | SQL_VARCHAR | 30 |
| 37 | USER_IDENT | SQL_VARCHAR | 35 |
| 38 | ACK_EXPECT | SQL_VARCHAR | 1 |
| 39 | TR_ACK_TYP | SQL_VARCHAR | 1 |
| 40 | T_P_IND | SQL_VARCHAR | 1 |
| 41 | TRANS_CNT | SQL_INTEGER | 10 |
| 42 | FILEOFFSET | SQL_BIGINT | 10 |
| 43 | RCOUNT | SQL_SMALLINT | 1 |
| 44 | SNDR_ROUTE | SQL_INTEGER | 14 |
| 45 | SNDR_SUBID | SQL_INTEGER | 35 |
| 46 | RCVR_ROUTE | SQL_INTEGER | 14 |
| 47 | RCVR_SUBID | SQL_INTEGER | 35 |
| 48 | APPL_REF | SQL_INTEGER | 14 |
| 49 | PROC_PRIOR | SQL_INTEGER | 1 |

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 50 | COMM_AGM | SQL_INTEGER | 35 |
| 51 | APP_SND_QL | SQL_INTEGER | 4 |
| 52 | APP_RCV_QL | SQL_INTEGER | 4 |
| 53 | ASSOC_CODE | SQL_INTEGER | 6 |
| 54 | APP_PSWD | SQL_INTEGER | 14 |
| 55 | CLIST_VER | SQL_INTEGER | 6 |
| 56 | MSG_TYPE | SQL_INTEGER | 6 |

### Run ID table (RUN_ID)

*Table 8-5: Run ID table*

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 1 | RUN_NO | SQL_BIGINT | 8 |

### Trace File (TRNN.DAT)

When an ODBC log database is used, the trace files are stored in the file TRNN.DAT in the current working directory (unless the -et switch is used to specify another directory). NN is the run number (RUN_NO field) from the RUN_ID table in the log database. RUN_NO is incremented for each run.

### Error log (ERROR) - used with functional acknowledgements

*Table 8-6: Error log*

| Number | Field name | Field type | Field precision |
|---|---|---|---|
| 1. | RUN_ID | SQL_BIGINT | 9 |
| 2. | ISA_SEND | SQL_VARCHAR | 35 |
| 3. | ISA_RECV | SQL_VARCHAR | 35 |
| 4. | GS_SEND | SQL_VARCHAR | 35 |
| 5. | GS_RECV | SQL_VARCHAR | 35 |
| 6. | GS_NUMBER | SQL_VARCHAR | 35 |
| 7. | ST_NUMBER | SQL_VARCHAR | 35 |
| 8. | TRANS_NAME | SQL_VARCHAR | 3 |
| 9. | SEGMENT | SQL_VARCHAR | 3 |
| 10. | SEG_NUMBER | SQL_VARCHAR | 10 |
| 11. | SEG_ERROR | SQL_VARCHAR | 50 |
| 12. | ELEM_NO | SQL_VARCHAR | 2 |
| 13. | SUBELEM_NO | SQL_VARCHAR | 2 |
| 14. | ELEM_ERROR | SQL_VARCHAR | 50 |

# Index

## F



## G



## I



## J



## L

## M



## N



## O



## P

## R



## S



## T

## U



## V



## W



## X