WebSphere Data Interchange

Programmer's Reference

IBM

WebSphere Data Interchange

Programmer's Reference

IBM

03December2004

This edition of this document applies to WebSphere Data Interchange for z/OS, Version 3.2.1.

To send us your comments about IBM WebSphere Business Integration documentation, e-mail doc-comments@us.ibm.com. We look forward to hearing from you.

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

Contents

Figures

Tables

About this book

This book describes general-use programming and provides reference information for developing application programs that use IBM® WebSphere® Data Interchange.

If you are familiar with previous versions of WebSphere Data Interchange and IBM DataInterchange, note the following changes in the content of this book:
- This book provides information for the application programmers working with WebSphere Data Interchange for Multiplatforms Version 3.2 and WebSphere Data Interchange for z/OS®
- Using WebSphere Data Interchange is now explained in 3 separate chapters where each of these chapters represents a different platform
- The following information has been moved from into this book:
  - Using WebSphere Data Interchange in the AIX® and Microsoft® Windows® Environment
  - C++ and Java™ API return codes
  - Mapping the MQRFH2 header to the JMS API
- Using sample JCL has been incorporated into Chapter 6 (Using WebSphere Data Interchange in the z/OS environment)

Who should read this book

This book is intended for the electronic data interchange (EDI) programmer who implements a computer system for electronic exchange of business information. The programmer should be familiar with or have a working knowledge of:
- z/OS
- Customer Information Control System (CICS®)
- IBM DataInterchange Version 3.1 and Version 3.2.x
- A programming language such as Assembler, C, or COBOL
- An IBM relational database management system, for example DB2 Universal Database™

Terms used in this book

All references in this book to z/OS are also applicable to supported releases of OS/390® unless otherwise stated. Customization and configuration differences between the z/OS and OS/390 are transparent to the user.

Syntax conventions used in this book

The following syntax conventions are used throughout this book:
- Bold letters represent values that you must type without change. Unless noted in the text, these values are not case-sensitive. For example:

  EDI PRINT(FILE)
- Bold letters also represent field names from panels. For example:

  Trans data queue

About this book

- Lowercase italicized letters represent variable parameters for which you supply the values. For example:

  SYSID(*system-name*)

Related books

The following books complete the WebSphere Data Interchange for Multiplatforms Version 3.2 library and contain information related to the topics covered in this book. You can view these documents, and download them, from the library page of the WebSphere Data Interchange for Multiplatforms Web site:

http://www.ibm.com/websphere/datainterchange

- WebSphere Data Interchange for z/OS Administration Guide, SC34-6214

  This book provides information for the electronic data interchange (EDI) administrator about entering, sending, and receiving EDI transactions and other documents interactively.

- WebSphere Interchange User's Guide, SC34-6215

  This book provides information on the WebSphere Data Interchange Client/Server user interface.

- WebSphere Data Interchange Messages and Codes, SC34-6216

  This book provides information to assist you in diagnosing errors.

You might also find the following publications useful:

- WebSphere Data Interchange for z/OS Installation Guide, SC34-6999

  This book provides information about installing WebSphere Data Interchange on z/OS and CICS systems.

- Expedite Base/MVS Programming Guide, GC34-2204

  This book provides a description of the Expedite Base/MVS™ communications program IEBASE.

- Expedite/CICS Display Application User's Guide, GC34-3303

  This book provides information about the Expedite/CICS Display Application.

- Customizing and Developing Applications with Expedite/CICS, GC34-3304

  This book provides a description of the CICS (interactive) interface.

- Informaiton Exchange Administration Services User's Guide, GC34-2221

  This book explains the major administrative tasks and describes how to use them.

- Information Exchange Programming Guide, GC34-2222

  This book provides information about programming for the Information Exchange interface.

- VS COBOL II Application Programming Guide, SC26-4045

  This book provides information on run-time parameters for improving performance.

- Using EDI VAN Interconnect, GC34-2263

  This book provides information about using EDI VAN Interconnect to connect to multiple networks.

Chapter 1. Using The Utility

The WebSphere Data Interchange Utility provides command-level access to WebSphere Data Interchange services. These services can be divided into the following categories:

- Performing general data translation
  - Translating data from any EDI, XML, or data format to any other EDI, XML, or data format using a data transformation map
- Processing outbound documents
  - Translating to EDI standard format using a send map
  - Enveloping
  - Sending
- Processing inbound EDI documents
  - Receiving
  - Deenveloping
  - Translating to data format using a receive map
- Managing data
  - Updating records
  - Removing records
  - Controlling record status
- Reporting and extracting data
  - Formatting and printing reports
  - Printing application data
  - Extracting data
- Customizing
  - Exporting and importing administrative data
- Managing mailboxes (CICS only)
  - Closing mailboxes

Any-to-any data translation

Any-to-any translation is a WebSphere Data Interchange feature that allows you to translate data from any supported source document type to any supported target document type. Supported document types include data formats, EDI standards, and XML data.

This feature expands the flexibility of WebSphere Data Interchange processing by allowing you to process a greater variety of formats. Any-to-any translation uses a new type of map called a *data transformation map*. A data transformation map is a set of mapping instructions that describes how to translate data from a source document into a target document. Both the source and target documents can be any one of several supported document types (application data to EDI standard format, EDI standard format to application data, XML data to EDI standard format, and so on).

Data transformation maps use the TRANSFORM command to do any-to-any translation. You should use any-to-any translation with any maps that you create. This is now the recommended method for translating documents.

Any-to-any data translation

Data transformation maps are created using WebSphere Data Interchange client. For more information about the mapping features in WebSphere Data Interchange client, refer to the WebSphere Data Interchange User's Guide.

Outbound processing using send maps

Outbound processing using send maps includes the following functions:

- Translating application data into an EDI standard format and placing it in the Transaction Store
- Enveloping standard transactions or messages so they are ready to be sent
- Sending enveloped data to trading partners

Commands are supplied to do each of these three steps independently or in combination. In some cases, using the combination commands can improve performance. The following commands are used for outbound processing:

- **ENVELOPE**
- **ENVELOPE AND SEND**
- **RECVFILE AND SEND**
- **REENVELOPE**
- **REENVELOPE AND SEND**
- **RESTART SEND**
- **SEND**
- **SENDFILE**
- **TRANSLATE AND ENVELOPE**
- **TRANSLATE AND SEND**
- **TRANSLATE TO STANDARD**

For more information on command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

WebSphere Data Interchange allows you to designate several types of files (print, exception, application, and so on) as MQSeries® Queues.

Fixed-to-fixed translation using send maps

The same outbound processes used for EDI standard translations are also used for fixed-to-fixed translation and enveloping. Fixed-to-fixed translation remains available for use with existing maps in WebSphere Data Interchange for Multiplatforms Version 3.2, but all new maps should be translated using any-to-any translation.

When translating to an EDI standard data format, fixed-to-fixed translation is only done when the standard has an envelope type of **F**. The application data being translated can include a mixture of fixed-to-fixed and EDI standard data types. The results of the translation and an image of the data translated are saved to the Transaction Store in C and D record format and are eligible for enveloping. For more information on the command syntax and command examples, see "TRANSLATE TO STANDARD command" on page 121.

For fixed-to-fixed translation, enveloping and reenveloping transactions from the Transaction Store includes sorting the transactions to obtain the fewest interchanges and groups, and then writing them to the appropriate file.

The appropriate files to use for EDI standard translated data in order of precedence are the file identified by the `FILEID` keyword, the file specified in the `Trans data queue name` field of the network profile member (with **E** or **U** suffix), or by default, **QDATA** (with **E** or **U** suffix).

The appropriate files to use for EDI standard fixed-to-fixed translated data in order of precedence are the file identified by the `FIXEDFILEID` keyword or the ddname formed by the concatenation of the standard ID (the `Application file name` in the data format that created the standard) with the `File` suffix.

Fixed-to-fixed translated data can either be written to the file in a C and D record format or in raw data format. The D record format is always used when multiple D records are found during translation. This format consists of the `Record ID` with a value of **D**, followed by the 16-byte name of the structure, followed by the structure's data. If the fixed translated data does not have an associated data format, the `Segment ID` from the EDI standard is used as the structure name. For more information on the command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

Sending with fixed-to-fixed translation

When the SEND command is performed in a composite command such as ENVELOPE and SEND, WebSphere Data Interchange remembers all the files created during an envelope and sends each file using the appropriate request to the network for sending the data. Standard data is sent using a SEND or SENDFILE command (based on the envelope type used). Fixed translated data with an interchange layer is sent the same way. Fixed translated data without an interchange layer (such as ISA, UNB, and so on) is sent as a file using the SENDFILE command.

You can also use the composite commands such as the TRANSLATE AND ENVELOPE command or ENVELOPE AND SEND command to control the processing of the data you send.

For more information on the command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

Inbound processing using receive maps

Inbound processing using receive maps includes the following functions:

- Receiving data from trading partners
- Deenveloping interchanges and placing the EDI standard transactions or messages into the Transaction Store
- Translating EDI standard transactions or messages into application formats

Inbound processing using receive maps

Commands are supplied to do each of these three steps independently or in combination. In some cases, using the combination commands can improve performance. The following commands are used for inbound processing:

- **DEENVELOPE**
- **DEENVELOPE AND TRANSLATE**
- **RECEIVE**
- **RECEIVE AND DEENVELOPE**
- **RECEIVE AND SEND**
- **RECEIVE AND TRANSLATE**
- **RECVFILE**
- **RESTART RECEIVE**
- **RETRANSLATE TO APPLICATION**
- **TRANSLATE TO APPLICATION**

For more information on command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

In CICS, you can not use the combination RECEIVE commands when Management Reporting is turned on and SYNCVAL is **-1**. For more information on SYNCVAL, see "WebSphere Data Interchange Utility control information" on page 368.

WebSphere Data Interchange allows you to designate several types of files (print, exception, application, and so on) as MQSeries Queues.

TA1 Overview

The TA1 is an X12 Interchange Acknowledgment. It acknowledges the receipt and or syntactical correctness of an X12 interchange header and trailer (ISA/IEA) pair. A request to generate a TA1 is based on the value in the ISA14 element. If the ISA14 element (I13) is 1, then an Interchange acknowledgment (TA1) has been requested. If the ISA14 element (I13) is 0, then a TA1 should not be created in response to the ISA. The Interchange control number of a TA1 is matched to a previously transmitted X12 Interchange with the same control number to complete the acknowledgment process. The TA1 segment may appear within an ISA/IEA of its own, or in conjunction with, but prior to, GS/GE segments.

The TA1 segment consists of five separate elements:

Interchange Control Number
Interchange Date (formatted as YYMMDD)
Interchange Time (formatted as HHMM)
Interchange Acknowledgment Code
A= Accepted
R= Rejected
E= Accepted but the file contains errors and must be resubmitted
Interchange Note Code
000 No error
001 The Interchange Control Number in the header and trailer do not match.
002 This Standard as noted in the Control Standards Identifier is not supported.

003 This Version of the controls is not supported
004 The Segment Terminator is invalid
005 Invalid Interchange ID Qualifier for sender
006 Invalid Interchange Sender ID
007 Invalid Interchange ID Qualifier for receiver
008 Invalid Interchange Receiver ID
009 Unknown Interchange Receiver ID
010 Invalid Authorization Information Qualifier value
011 Invalid Authorization Information value
012 Invalid Security Information Qualifier value
013 Invalid Security Information value
014 Invalid Interchange Date value
015 Invalid Interchange Time value
016 Invalid Interchange Standards Identifier value
017 Invalid Interchange Version ID value
018 Invalid Interchange Control Number
019 Invalid Acknowledgment Requested value
020 Invalid Test Indicator value
021 Invalid Number of Included Group value
022 Invalid control structure
023 Improper (Premature) end-of-file (Transmission)
024 Invalid Interchange Content (e.g., invalid GS Segment)
025 Duplicate Interchange Control Number
026 Invalid Data Element Separator
027 Invalid Component Element Separator
028 Invalid delivery date in Deferred Delivery Request
029 Invalid delivery time in Deferred Delivery Request
030 Invalid delivery time Code in Deferred Delivery Request
031 Invalid grade of Service Code

WDI TA1 processing

To request a TA1 for outbound EDI transactions, the ISA14 element (I13) must be populated with the value 1 by using the SetProperty() command within the data transformation map, or via the envelope profiles. Pending interchange acknowledgments can be viewed as a report of WDI Client. A pending interchange acknowledgment will be identified as a Send Interchange.

When the response TA1 is received for a given interchange, it may be reflected or reconciled in the Transaction Store. TA1 reconciliation is only possible with WDI data transformation processing and if Transaction Store is active. If Transaction Store is not active, the TA1 segment will be ignored. TA1 reconciliation will not be performed for Interchanges which use control numbers by transaction (document ID), such Interchanges cannot be uniquely identified from information in the TA1 and its encompassing ISA. Only the last TA1 received for a given Interchange will be reflected in the Transaction Store.

When a TA1 is requested for inbound EDI transactions, the TA1 segment will be generated for incoming EDI transactions regardless of Transaction Store setting. The TA1 will also be generated and enveloped when using delayed functional acknowledgments FADELAY(Y).

Inbound processing using receive maps

WDI will set the TA1 acknowledgment code and note code based on ISA/IEA validation accomplished using the SERVICESEGVAL keyword and WDI validation map $WDI_X44ENV_VAL. The TA1 acknowledgment code of R will be based on the TA1 note code, which is determined by the first error detected during ISA/IEA validation. Only the first error is reported. Interchanges with E errors will be rejected. WDI will use the functional acknowledgment map $WDI_TA1_ACK to map the errors reported from the validation map to the TA1 segment.

Considerations

Since the TA1 acknowledges the receipt and syntactical correctness of an X12 Interchange header and trailer (ISA/IEA) pair, the E acknowledgment code cannot be supported. These errors are reported in the functional acknowledgment message.

The following TA1 note codes are not supported during interchange validation:

002 This Standard as noted in the Control Standards Identifier is not supported.
004 The Segment Terminator is invalid
022 Invalid control structure
024 Invalid Interchange Content (e.g., invalid GS Segment)
026 Invalid Data Element Separator
027 Invalid Component Element Separator

The TA1 note code 009 will be reported with WDI Rule not found conditions. Additional validation and reporting may be possible by creating and adding mapping commands and code lists based on individual requirements. TA1 note code 020 for Invalid Test Indicator value may result in TA1 reporting code 009 WDI Rule not found condition.

Managing data

Data management includes the following functions:
- Rebuilding interchanges from transactions in the Transaction Store
- Updating status of transactions in the Transaction Store
- Removing transactions from the Transaction Store
- Updating management reporting statistics
- Removing management reporting statistics

The following commands are used for managing data:
- **HOLD**
- **LOAD LOG ENTRIES**
- **PROCESS NETWORK ACKS**
- **PURGE**
- **QUERY**
- **RECONSTRUCT**
- **RECONSTRUCT AND SEND**
- **RELEASE**
- **REMOVE LOG ENTRIES**
- **REMOVE STATISTICS**
- **REMOVE TRANSACTIONS**
- **RESET STATISTICS**

- **UNLOAD LOG ENTRIES**
- **UNPURGE**
- **UPDATE STATISTICS**
- **UPDATE STATUS**

For more information on command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

Removing and archiving event log entries

You can remove event log entries and archive them. Typically, this is a multi-step process because, while event log entries are removed real-time, reclaiming and reorganizing the table space requires an additional step as described in Table 1.

Table 1. Removing and archiving event log entries

Remove DB2® log entries:	Run WebSphere Data Interchange PERFORM command:	Run DBS Utility:
Without archive Without DB2 reorg	REMOVE LOG ENTRIES WHERE APPLID (*value*)	N/A
With archive Without DB2 reorg	UNLOAD LOG ENTRIES WHERE APPLID (*value*) ARCHIVEFILE (*value*) HOLDFILE (*value*)	N/A
Without archive With DB2 reorg	REMOVE LOG ENTRIES WHERE APPLID (*value*)	REORG TABLESPACE EDIELOG
With archive With DB2 reorg	UNLOAD LOG ENTRIES WHERE APPLID (*value*) ARCHIVEFILE (*value*) HOLDFILE (*value*)	REORG TABLESPACE EDIELOG

Reporting and extracting data

Data reporting and extraction includes the following functions:

- Generating formatted reports containing data from the Transaction Store
- Formatting application data into conventional business documents
- Extracting data from the Transaction Store for further processing outside WebSphere Data Interchange
- Extracting data from the management reporting statistics tables for further processing outside WebSphere Data Interchange

WebSphere Data Interchange provides two mechanisms for producing reports:
- Management Reporting data extracts
- Transaction Store data extracts

Both mechanisms collect, update, and extract WebSphere Data Interchange trading partner data and transaction information. Both are invoked with PERFORM commands to extract the data, and both require user-written programs to sort, format, and print data.

The Management Reporting and Transaction Store Data Extracts differ in the type of information they provide. Management Reporting Data Extracts pull information from the statistics tables, while Transaction Store Data Extracts pull information from the Transaction Store.

WebSphere Data Interchange collects information in four management reporting categories and two Transaction Store reporting categories, as follows:
- Management Reporting Categories

- Trading Partner Profile - provides general trading partner information alphabetically, by trading partner. Reported information includes company name, address, point of contact, telephone numbers, account ID, and user ID.
- Trading Partner Capability - relates trading partner names used. For example, this report allows you to determine which maps are installed in test and production, which trading partners are using which translation usages/rules, direction of the translation, EDI standards used, transaction ID, total number of transactions processed, and number of transactions that had errors.
- Transaction Activity - provides information relating transaction volumes to trading partner names. For example, this report can calculate the total number of transactions sent to a trading partner (by date and by map ID) or the total transactions errors by trading partner (and by date or map ID). This information is often used to gauge a trading partner's activity or to reconcile documents sent or received.
- Network Activity - provides information such as network ID, name and account number, user ID, direction, charge code, and total number of bytes sent or received. This information is typically used to determine network charges, by trading partner or application.
- Transaction Store Reporting Categories
  - Transaction Data Extract - extracts detailed technical information from the Transaction Store. It provides data such as trading partner nickname, direction (S/R), transaction image, send acknowledgment data and image, receive acknowledgment data and image, and transaction handle. This information is commonly used for creating daily reports, such as overdue functional acknowledgments.
  - Envelope Data Extract - extracts detailed technical information from the Transaction Store. It provides the same types of information as the transaction data extract except envelope data extract only reports on transactions that have been enveloped, and it sorts the data differently. This information is commonly used for creating daily reports.

PRINT commands

You can use the PRINT commands with selection criteria to extract information from the Transaction Store and write formatted reports to ddname **RPTFILE**. To obtain the same type of information without formatting, use the ENVELOPE and TRANSACTION DATA EXTRACT commands. You can print the information using your system facilities. The cover page for each report lists the selection criteria you entered. You can view most of the information in these reports using the Transaction Store Facility. Samples of the different formats are listed in Chapter 2 with the commands that generate them. For more information, "Producing management reports from the Transaction Store" on page 10.

The following commands are used for printing reports:
- **PRINT**
- **PRINT ACKNOWLEDGEMENT IMAGE**
- **PRINT ACTIVITY SUMMARY**
- **PRINT EVENT LOG**
- **PRINT STATUS SUMMARY**

Reporting and extracting data

- **PRINT STATUS SUMMARY2**
- **PRINT TRANSACTION DETAILS**
- **PRINT TRANSACTION IMAGE**

For more information on command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

Producing management reports from the Transaction Store

The following commands are used for extracting data from the Transaction Store:
- **ENVELOPE DATA EXTRACT**
- **NETWORK ACTIVITY DATA EXTRACT**
- **TRADING PARTNER PROFILE DATA EXTRACT**
- **TRADING PARTNER CAPABILITY DATA EXTRACT**
- **TRANSACTION ACTIVITY DATA EXTRACT**
- **TRANSACTION DATA EXTRACT**

For more information on command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

Creating management reporting reports

To create management reports, do the following:

1. Set the `Management Reporting Active?` field to **Y** in the Application Defaults profile member. The default is **Y**.

   For any trading partner profiles that you want reported, include selection information in the profile fields, such as a department name in a comment field. For more information on management reporting, "TRADING PARTNER PROFILE DATA EXTRACT command examples" on page 107.

2. Execute the UPDATE STATISTICS command to move the pending information to the statistics tables. You must perform this step prior to data extraction. For more information, "UPDATE STATISTICS command" on page 125.

   ```
   PERFORM UPDATE STATISTICS
   ```

3. Execute the DATA EXTRACT command.

   ```
   PERFORM DATA EXTRACT
   ```

   Together, these two PERFORM commands collect the requested data and place it in a QSAM file named EDIQUERY. You can use wild cards on DATA EXTRACT commands, as follows:

Table 2. Interpretation of wildcard characters

Wild Card	Matches
(any character)	The specified character
?	Any single character
*	Any sequence of one or more characters

For more information, see the report examples included with the data extract commands in Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

4. Process the data extract file into reports by:

   • Verifying that you successfully created a QSAM data extract file (EDIQUERY) containing a sequential set of independent records with a record length of 1024. For more information, see "Management reporting" on page 297.

   • Sorting the data according to your requirements through a sort utility.

   • Formatting the data using a report-writing utility or a user-written program.

5. Remove previously collected daily statistics by executing the REMOVE STATISTICS command:

   ```
   PERFORM REMOVE STATISTICS
   ```

Because statistics are not dependent on the Transaction Store, running the REMOVE TRANSACTIONS command does not affect them. For more information, see "PRINT commands" on page 9.

Note: You can use the USERPGM parameter to pass the report records to a program prior to writing them to the EDIQUERY QSAM file. When using this parameter, your program should return a code indicating whether the record is written to the EDIQUERY file or is discarded. If a program is not supplied, WebSphere Data Interchange writes the record to the EDIQUERY file.

Creating transaction data or transaction envelope reports

Using the TRANSACTION DATA EXTRACT and ENVELOPE DATA EXTRACT commands, you can extract detailed information about transaction data and envelopes from the Transaction Store, such as:

• Reporting the number of purchase orders sent in a given period of time.

• Reporting the total number of bytes sent in a given period of time. (This can be useful for charging back costs to other departments based on their EDI usage.)

• Creating a customized functional acknowledgment tracking report, selected by application or by department. For example, you could create an Exception Report on Purchase Orders sent more than 2 days ago that have not been acknowledged.

• Creating an exception report flagging missing control numbers for inbound envelopes.

• Creating statistical reports identifying the most frequently used transactions, EDI standards, version sizes, or delimiters.

You can also use the ENVELOPE DATA EXTRACT and TRANSACTION DATA EXTRACT commands for functions other than reporting, such as:

• Archiving Transaction Store data.

• Loading status data directly into the application by application key; for example, the status for each invoice sent could be loaded into the billing system by invoice number.

Producing management reports from the Transaction Store

The TRANSACTION DATA EXTRACT provides data such as trading partner nickname, direction (Send or Receive), transaction image, send acknowledgment data and image, receive acknowledgment data and image, and transaction handle.

The ENVELOPE DATA EXTRACT provides the same types of information as the transaction data extract command except envelope data extracts include only enveloped transactions, sorted by the requested envelope key such as trading partner nickname, direction, interchange control number, receiver ID. Transaction data extracts include all transactions sorted by transaction handle.

For more information on report record layouts, see "Transaction data extract record layout" on page 306.

Creating Transaction Store reports

To create Transaction Store reports, do the following:

1. Select the type of information to collect in the Transaction Store:

   a. To collect information, set the `Trx. store active?` field in the Application Defaults profile to a value other than **N**.

   b. To collect transaction images, set the `Trx. image wanted?` field in the Application Defaults profile to a value other than **N**.

   c. To collect functional acknowledgment images, set the `FA image wanted?` field to a value other than **N**.

2. Perform the usual translations and communications.

3. To extract the information for the reports, execute the appropriate DATA EXTRACT command using one of the following commands:

   ```
   PERFORM TRANSACTION DATA EXTRACT
   ```

   or

   ```
   PERFORM ENVELOPE DATA EXTRACT
   ```

4. Create a user-written program to format the data extract file output (EDIQUERY) into reports.

5. Remove transactions periodically using the following command:

   ```
   PERFORM REMOVE TRANSACTIONS
   ```

For more information on these commands, see "Reporting and extracting data" on page 8.

Exporting and importing

The EXPORT and IMPORT commands allow you to export or import the following:
- Maps (without reworking the entire map)
- Trading partner setup information (such as profile members)
- Administrative data

You tell WebSphere Data Interchange specifically which setup information you want to exchange using a control file.

The following commands are used for exporting and importing:
- **CLOSE MAILBOX**
- **EXPORT**
- **IMPORT**

For more information on command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

Profile maintenance

You can query for specific profiles to review them and you can delete obsolete profiles. The following commands are used for exporting and importing:
- **DELETE PROFILE**
- **QUERY PROFILE**

For more information on command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

Continuous receive

WebSphere Data Interchange can take advantage of the continuous receive capabilities provided by Expedite/CICS. This allows WebSphere Data Interchange to receive data immediately after it is put in the VAN mailbox. The continuous receive function is only available on CICS.

When a continuous receive is started through WebSphere Data Interchange, WebSphere Data Interchange and Expedite/CICS each keep a control record about the continuous receive. Although WebSphere Data Interchange and Expedite/CICS can get out of sync when managing continuous receives, this is an exception to normal processing. For more information about out of sync situations, "Continuous receive session cleanup" on page 396.

The following commands are used for continuous receive:
- **REPORT CONTINUOUS RECEIVE STATUS**
- **START CONTINUOUS RECEIVE**
- **STOP CONTINUOUS RECEIVE**

For more information on command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

Reporting continuous receive status

A continuous receive status report includes the status of either a single continuous receive member or all continuous receives known to WebSphere Data Interchange and Expedite/CICS. The output goes to the report file specified for the report file name and type in the Utility Control Information Block. The default is **RPTFILE** (type **TS** in CICS). If an error is encountered while generating the report, the processing program immediately terminates and does not report on subsequent continuous receives.

Continuous receive

> **Note:** Continuous receive report statuses may be good or bad depending on the result you expected. For example, a status of EXP STARTED N/P may be an acceptable response that perfectly describes the state of that particular continuous receive. On the other hand, a status of EXP STARTED might be a problem and indicate an out-of-sync situation between WebSphere Data Interchange and Expedite/CICS.

Table 3. Continuous receive status codes

If report status is:	This continuous receive is:
STARTED	Considered to be running by both WebSphere Data Interchange and Expedite/CICS. A continuous receive profile member exists.
DI STARTED	Considered to be running by WebSphere Data Interchange, but not by Expedite/CICS. A continuous receive profile member exists.
EXP STARTED	Considered to be running by Expedite/CICS, but not by WebSphere Data Interchange. A continuous receive profile member exists.
STARTED N/P	Considered to be running by both WebSphere Data Interchange and Expedite/CICS. However, a continuous receive profile member does not exist.
DI STARTED N/P	Considered to be running by WebSphere Data Interchange, but not by Expedite/CICS. A continuous receive profile member does not exist.
EXP STARTED N/P	Considered to be running by Expedite/CICS, but not by WebSphere Data Interchange. A continuous receive profile member does not exist.
NOT STARTED	Not considered to be running by either WebSphere Data Interchange or Expedite/CICS. However, a continuous receive profile member does exist and it is eligible to be started.
NOT STARTED N/A	Not considered to be running by either WebSphere Data Interchange or Expedite/CICS. A continuous receive profile member does exist, but it is not eligible to be started.

Table 4. Continuous receive status report record format

Name	Offset	Length	Type	Description
CRSTATUS	0	16	Char	Report status
CDPROFID	16	16	Char	WebSphere Data Interchange continuous receive profile member
CDREQACT	32	08	Char	WebSphere Data Interchange requestor's account
CDREQUSE	40	08	Char	WebSphere Data Interchange requestor's user ID
CDUNIQUE	48	08	Char	WebSphere Data Interchange continuous receive unique ID
CDMSGUC	56	08	Char	WebSphere Data Interchange message user class
CDTPACCT	64	08	Char	WebSphere Data Interchange trading partner's account
CDTPUSER	72	08	Char	WebSphere Data Interchange trading partner's user ID

Table 4. Continuous receive status report record format (continued)

Name	Offset	Length	Type	Description
CEREQACT	80	08	Char	Expedite/CICS account
CEREQUSE	88	08	Char	Expedite/CICS user ID
CEUNIQUE	96	08	Char	Expedite/CICS continuous receive unique ID
CEMSGUCL	104	08	Char	Expedite/CICS message user class
CETPACCT	112	08	Char	Expedite/CICS trading partner's account
CETPUSER	120	08	Char	Expo/CICS trading partner's user ID
RESERVED	128	04	Char	Blanks

Persistent environment

The persistent environment is an optional WebSphere Data Interchange feature available with CICS/ESA. This command is available only in the CICS environment and is not required for normal processing. This command is used to gather information required by WebSphere Data Interchange support personnel when debugging problems related to the Persistent Environment. The resulting dump is written to EDIGDMP1 in the CICS startup JCL.

The following commands are used for debugging the persistent environment:
- **GLB DUMP**
- **GLB TRACE**

For more information on command syntax and command examples, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

Using the WebSphere Data Interchange Utility in the z/OS environment

WebSphere Data Interchange provides sample JCL statements for running the DataInterchange Utility in z/OS. You can copy and customize these statements as necessary. The following is an explanation of the parameters your application can pass in the JCL. Sample JCL files are shipped with the DataInterchange Host product. All the parameters are DataInterchange keywords and optional (except PLAN and SYSTEM, which may be required under certain conditions).

APPLID=*aaaaaaaa*

Specifies the application ID to run the DataInterchange Utility. This parameter also identifies the log file specified in the Activity Log (APPDEFS) profile. Replace *aaaaaaaa* with the ID of the application that initialized DataInterchange. If you specify this parameter, the activity log profile must contain a matching entry to define which log file is used for recording errors and events pertaining to the application. The two `APPLID` values shipped with DataInterchange are:
- **EDIFFS (default)**. Associated with the LOGFFS ddname. The default APPLID and log when using the utilities

- **EDIMP**. Associated with the LOGEDI ddname. The APPLID and log used during online DataInterchange processing

SYSID=*bbbbbbbb*

Identifies the installation-defined WebSphere Data Interchange system used to run the EDIUTIL utility. Specifying this parameter controls access to various components of DataInterchange. The `SYSID` is part of the resource name defined using RACF or some other resource control product. Replace *bbbbbbbb* with the ID used to protect DataInterchange services (for example, RACF). The default ID is **DIENU**.

LANGID=*ccc*

Identifies the language ID used to run the EDIUTIL utility. The value supplied must match an entry in the language(`LANGPROF`) profile. The language ID is used to establish values such as date formats and decimal notation. Replace *ccc* with the following value for the language version:

ENU. English

DLM=*d*

Specifies the delimiter used in place of left and right parentheses to enclose values in the EDIUTIL utility command language. Replace *d* with the delimiter you want to use in place of the left and right parentheses to enclose values in the DataInterchange Utility command language. You must supply this parameter if a keyword value contains either a left or right parenthesis.

MQSYSIN=*eeeeeeee*

Replace *eeeeeeee* with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to either ddname SYSNAME or ddname EDISYSIN.

MQPRT=*fffffff*

Replace *fffffff* with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname PRTFILE.

MQRPT=*ggggggggg*

Replace *ggggggggg* with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname RPTFILE.

MQEXCP=*hhhhhhhh*

Replace *hhhhhhhh* with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname FFSEXCP.

MQTRAK=*iiiiiiii*

Replace *iiiiiiii* with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname FFSTRAK.

MQQUERY=*jjjjjjjj*

Replace *jjjjjjjj* with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname EDIQUERY.

PLAN=*kkkkkkkk*

> In a DB2 environment, if the utility is invoked by way of IKJEFT01 and there is no EDITSIN data set allocated, then this parameter is required. For more information, see "WebSphere Data Interchange DB2 command file (EDITSIN)" on page 194 Replace *kkkkkkkk* with the DB2 plan name.

SYSTEM=*llll*

> In a DB2 environment, if the utility is invoked by way of IKJEFT01 and there is no EDITSIN data set allocated, then this parameter is required. For more information, see "WebSphere Data Interchange DB2 command file (EDITSIN)" on page 194 Replace *llll* with the name of the DB2 subsystem (or group, if Data Sharing).

Optional JCL parameter example

```
PARM('SYSID=TEST LANGID=ENU SYSTEM=DB93 PLAN=EDIENU32')
```

Using the WebSphere Data Interchange Utility in the z/OS environment

Chapter 2. WebSphere Data Interchange commands and keywords

This chapter explains how to use the WebSphere Data Interchange Utility command language. For additional information about invoking the WebSphere Data Interchange Utility in an z/OS environment, see "Using the WebSphere Data Interchange Utility in the z/OS environment" on page 15. For additional information about invoking the WebSphere Data Interchange Utility in a z/OS-CICS environment, see "Running the WebSphere Data Interchange Utility in the CICS environment" on page 347. For additional information about invoking the WebSphere Data Interchange Utility in an AIX and Windows environment, see Chapter 7, "Using WebSphere Data Interchange in the AIX and Windows environment," on page 577.

Command language syntax

The WebSphere Data Interchange Utility command language consists of PERFORM statements, WHERE clauses, and SELECTING clauses. It is represented in the text as follows:

```
PERFORM SAMPLE-COMMAND
    SELECTING SAMPLE-KEYWORD(value)
    WHERE SAMPLE-KEYWORD(value)
```

A PERFORM statement defines the action WebSphere Data Interchange takes. The following statement consists of the word PERFORM followed by the TRANSLATE TO APPLICATION command as follows:

```
PERFORM TRANSLATE TO APPLICATION
```

While your input can include more than one PERFORM statement, the WebSphere Data Interchange Utility handles each statement separately. It verifies the syntax of each statement and processes it before verifying and processing the next statement. If a nonzero (error) return code is generated for a statement, processing stops with the incorrect statement.

A WHERE clause supplies selection criteria and other information that WebSphere Data Interchange requires to process your requests. A WHERE clause consists of the word WHERE, followed by one or more keywords and their associated values. You must enclose the associated values in parentheses. For example, see the keyword value **PISCES** in the WHERE clause below:

```
    WHERE TPNICKN(PISCES)
```

Note: In z/OS, you can redefine the parentheses delimiter with the DLM parameter. For more information, see "Using the WebSphere Data Interchange Utility in the z/OS environment" on page 15.

A PERFORM statement can include more than one WHERE clause. Each WHERE clause can contain more than one keyword, but each keyword can be used only once within a WHERE clause. You can use a keyword again on the same PERFORM statement in a different WHERE clause. The following example shows a command statement that uses a WHERE clause:

```
PERFORM TRANSLATE TO APPLICATION
    WHERE TPNICKN(PISCES) TRXDATE(12/12/01)
    WHERE TPNICKN(PISCES) TRXDATE(12/14/01)
```

The previous statement tells WebSphere Data Interchange to translate to application format all EDI documents that were received from a trading partner whose name is PISCES, and whose documents were placed in the Transaction Store on December 14, 2001. WebSphere Data Interchange processes only those documents that meet both of the conditions specified in the WHERE clause. When you specify all of your conditions in one WHERE clause, WebSphere Data Interchange limits processing to only those transactions that satisfy all the specified conditions.

If you want WebSphere Data Interchange to process EDI documents with either condition, use two separate WHERE clauses. For example, the following statement will translate all records for trading partner PISCES as well as all records with a transaction date of December 14, 2001:

```
PERFORM TRANSLATE TO APPLICATION
    WHERE TPNICKN(PISCES)
    WHERE TRXDATE(12/10/01)
```

Some statements use a SELECTING clause. Like WHERE clauses, SELECTING clauses provide selection criteria. However, you can only include one SELECTING clause in each PERFORM statement. The following example shows a command statement that uses a SELECTING clause:

```
PERFORM ENVELOPE DATA EXTRACT
    SELECTING INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
    WHERE TPNICKN(PISCES) INTCTLNO(000008888) DIR(S)
```

The command language format is free-form. You can insert blanks between keywords to improve readability, but do not insert blanks between a keyword and its associated value. Characters can be in upper, lower, or mixed case. Most fields are not case sensitive. To determine if a field is case sensitive, see the field's definition in "Keyword descriptions" on page 127.

To select a range of values, you can follow some keywords with a pair of values separated by the keyword T0. The first value indicates the low end of the range. The second value indicates the high end of the range. Only transactions with values falling between these two values will be selected.

To insert comments between commands, place an asterisk (*) in column one and type the text on the same line. Each line of a comment must begin with an asterisk or the system will attempt to process the information as commands. For example:

```
//SYSIN DD *
*
*This command translates my data into standard format
*
PERFORM TRANSLATE TO STANDARD
    WHERE APPFILE(APDATA01)
/*
```

DATE, TIME, and HANDLE keywords

Several keywords are used to identify date or time. The format of the date and time values you submit must conform to the Date mask and Time mask fields in the language profile used. You can request the current date by using an asterisk (*), or you can request a previous date by using *-*n*, where *n* is the number of days before today's date. You can also use an asterisk to request the current time.

The HANDLE keyword is used to select a specific transaction or transaction group for processing or reporting. This keyword specifies the date, time, and transaction ID in format YYYYMMDDHHMMSSnnnnnn.

Command language validation

WebSphere Data Interchange performs basic validation on the command language input, as follows:

* Each PERFORM command (such as TRANSLATE TO STANDARD) is validated. WebSphere Data Interchange generates an error if the command is not valid.
* All keywords are validated. WebSphere Data Interchange generates an error if a keyword is not valid.
* If a keyword is specified, an associated value must also be specified. WebSphere Data Interchange generates an error if you do not specify a keyword value.
* You must enclose each keyword value with beginning and ending delimiters. The default delimiters are opening and closing parentheses. WebSphere Data Interchange generates an error if you do not enclose a keyword value with delimiters.
* The length of each keyword value is validated against the maximum allowable length for that keyword. WebSphere Data Interchange generates an error if the length of the value you supply is greater than the maximum allowable length for the keyword.
* Keywords cannot be used more than once in a WHERE clause. However, you can specify multiple WHERE clauses. WebSphere Data Interchange generates an error if you specify a keyword more than once in the same WHERE clause.
* Date and time fields are validated. WebSphere Data Interchange generates an error if the date or time values you specify are not valid.
* When a keyword has an associated default value, keyword values that are not valid are ignored, and defaults are used. An example of a keyword value that is not valid is RAWTEST(**X**). In this situation, WebSphere Data Interchange ignores the value of **X** and does not generate an error. For specific default values, see "Keyword descriptions" on page 127.
* If a keyword is used as part of selection criteria against the Transaction Store, the associated value is accepted. If the associated value is not valid, WebSphere Data Interchange uses the value as part of the selection criteria, and does not find a transaction match. Verify the associated values for each keyword used for Transaction Store selection criteria.

Error filtering

You can use a special named variable called DIERRFILTER to tell the translator to ignore certain error conditions. DIERRFILTER may also be used as a keyword on PERFORM commands to indicate which errors should be ignored during the translation process.

When the DIERRFILTER variable is specified as **SET** or **SAVED**, the translator parses the value of the variable as an indicator of the errors to ignore. The DIERRFILTER variable value should consist of a list of the error codes to be ignored. These values are documented in "Translator Error Codes" on page 670.

When an error is filtered, it does not mean the error did not occur. Eliminating the message and its corresponding condition code does not eliminate the error. For example, a TR0004 error message can be issued if a validation edit fails. This error can be filtered so as not to produce the TR0004 message, but doing so does not alter the fact that the edit failed. Filtering only serves to suppress the error message and its condition code but does not otherwise affect translation processing.

Note: If you filter an error that would normally generate a functional acknowledgment, filtering the error eliminates the functional acknowledgment as well. If the error is not reported because of filtering, it is not reported in a functional acknowledgment either.

For example, if you are not interested in the "mandatory data element missing" error (TR0001-101), or the "data element is too short" error (TR0003-103), you can filter them out by specifying:

```
&SET DIERRFILTER 101,103
```

You can filter many errors without typing a code for each one by specifying a range of values in the variable. For example, you would eliminate errors 101, 103 and all warning messages by specifying:

```
&SET DIERRFILTER 101,103,1-99
```

Because DIERRFILTER is a named variable, you can use a statement like the following to eliminate validation errors, along with those errors previously set:

```
&SET DIERRFILTER &E(DIERRFILTER + ',116')
```

Certain errors occur before translation is performed, even before a map has been determined. Most of these errors fall into the warning category. You can use the following PERFORM command to eliminate all the warning errors before translation is performed:

```
PERFORM TRANSLATE TO STANDARD
    WHERE DIERRFILTER(1-99)
```

The DIERRFILTER variable has a transaction scope and, therefore, the value set on one transaction does not propagate to the next transaction. The translator sets the DIERRFILTER variable to the value specified in the PERFORM command at the start and end of each transaction.

There are certain errors that you cannot turn off. For example, you cannot filter error TR0053 (maximum number of structures exceeded) because the message is too important, as it indicates data is being ignored. If you attempt to filter an error that is not filterable, the request is ignored (no error message is generated).

Errors TR0101, TR0103, TR0151, TR0153, TR0203 and TR0205 indicate that the ISA/IEA, GS/GE, or ST/SE segment pairs are inconsistent with respect to control numbers or control counts. If these errors are filtered, WebSphere Data Interchange will process the interchange, group, or message in spite of the inconsistency.

In the case where a map contain a complex set of DIERRFILTER values, and something is not working correctly, you may need to see all error messages. Rather than changing the map, you can use the DIERRFILTER keyword with a value of **IGNORE** to tell WebSphere Data Interchange that all errors should be reported regardless of the instructions in the map, as follows:

```
PERFORM TRANSLATE TO STANDARD
    WHERE DIERRFILTER(IGNORE)
```

Overriding utility condition codes

The WebSphere Data Interchange Utility returns a condition code to z/OS that can be tested in the JCL. In cases where you want to ignore certain condition codes, you can use the keywords IFCC and SETCC on any PERFORM statement to override up to 10 different condition codes.

Note: When overriding condition codes, do not ignore important error conditions.

These keywords are entered as follows:
- IFCC($c1,c2,c3,c4,c5,c6,c7,c8,c9,c10$)

  The values $c1$-$c10$ represent the utility condition codes to be checked.
- SETCC($n1,n2,n3,n4,n5,n6,n7,n8,n9,n10$)

  The values $n1$-$n10$ will be used to override the IFCC condition codes $c1$-$c10$, respectively.

If SETCC keyword is not specified, all codes specified in the IFCC keyword are overridden to zero (0).

The following is an example of an override of an empty application file condition on TRANSLATE TO STANDARD:

```
PERFORM TRANSLATE TO STANDARD
    WHERE APPFILE(APDATA01) IFCC(6) SETCC(0)
```

Note: Since zero (0) is the default override code, you can omit the SETCC keyword and get the same results as entering a value of zero.

CLOSE MAILBOX command

WebSphere Data Interchange does not wait for send requests to complete when using Expedite/CICS and Information Exchange. When any type of access to Information Exchange is accomplished (send, receive, continuous receive, network status update), the mailbox opens and remains open until you close it. Closing your mailbox is only necessary when the same mailbox will be used in the z/OS environment. To close the mailbox, you must use this command to end the session. This command is only available in the CICS environment and is not required for general processing.

Syntax

CLOSE MAILBOX
REQID(*requestor ID*)

CLOSE MAILBOX command example

Close the mailbox for requestor **DEPTA7F**.

```
PERFORM CLOSE MAILBOX
    WHERE REQID(DEPTA7F)
```

DEENVELOPE command

This command takes EDI transactions from the envelope file, removes the envelope segments, and places the results in the Transaction Store. The envelope file is either the receive file specified in the mailbox (requestor) profile or the file specified in the command.

Syntax

DEENVELOPE
DIERRFILTER(*initial error filter set*)
DUPENV(*process duplicate envelopes*)
EXTENDC(*translate with extended C record format*)
FADELAY(*delay queuing functional acknowledgment*)
FILEID(*processing file ddname*)
FORCETEST(*force test usage*)
FUNACKFILE(*functional acknowledgment ddname*)
FUNACKREQ(*require functional acknowledgment envelope file*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
INMEMTRANS(*transactions in memory*)
MRREQID(*management reporting requestor ID*)
MULTIDOCS(*multiple-document file*)
OPTRECS(*optional record type*)
OUTFILE(*output data file name for SAP Status*)
OUTTYPE(*output data file type for SAP Status*)
PAGE(*pageable translation*)
PURGINT(*purge interval*)
RECOVERY(*recovery unit of work*)
REQID(*requestor ID*)
SAPUPDT(*track SAP status*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
XML(*XML required*)
XMLDICT(*XML dictionary address*)
XMLDTDS(*XML DTD path*)
XMLEBCDIC(*EBDCDIC indicator*)
XMLSEGINP(*line break indicator*)
XMLSTDID(*destination EDI standard ID*)
XMLVALIDATE(*XML validation indicator*)

DEENVELOPE command examples

Example 1:

Deenvelope the transactions previously received for requestor ID **DEPTA7F**. Place the results in the Transaction Store. Mark these transactions for purging after 10 days in the store.

```
PERFORM DEENVELOPE
    WHERE REQID(DEPTA7F) PURGINT(10)
```

Example 2:

Deenvelope the transactions in ddname **A7FIN**. Place the results in the Transaction Store. Do not allow duplicate interchanges to be processed.

```
PERFORM DEENVELOPE
    WHERE FILEID(A7FIN) DUPENV(N)
```

DEENVELOPE AND TRANSLATE command

This command combines the functions of the DEENVELOPE and TRANSLATE TO APPLICATION commands. It takes EDI documents from the receive file defined in the mailbox (requestor) profile member or from the override file specified in the command, removes the envelope segments, and places the results in the Transaction Store. It then translates the documents to the defined application format and places the results in the application file specified by the data format or in the file specified by the trading partner usage/rule. This composite command performs faster than the separate commands.

Note: In CICS, you can also deliver the data to a program of CICS transaction, as specified in the `Application file name` and `Application file type` fields of the data format.

Syntax

DEENVELOPE AND TRANSLATE
ASSERTLVL(*session assertion level*)
BATCHSET(*set transaction batch ID*)
CCEXCEPTION(*job-step condition code*)
DIERRFILTER(*initial error filter set*)
DUPENV(*process duplicate envelopes*)
EXTENDC(*translate with extended C record format*)
FADELAY(*delay queuing functional acknowledgment*)
FILEID(*processing file ddname*)
FORCETEST(*force test usage*)
FUNACKFILE(*functional acknowledgment ddname*)
FUNACKREQ(*require functional acknowledgment envelope file*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
INMEMTRANS(*transactions in memory*)
MRREQID(*management reporting requestor ID*)
MULTIDOCS(*multiple-document file*)
OPTRECS(*optional record type*)
OUTFILE(*output data file name for SAP Status*)
OUTTYPE(*output data file type for SAP Status*)
PAGE(*pageable translation*)
PURGINT(*purge interval*)
RAWDATA(*translate to raw data format*)
RECOVERY(*recovery unit of work*)
REQID(*requestor ID*)
SAPUPDT(*track SAP status*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
XML(*XML required*)
XMLDICT(*XML dictionary address*)
XMLDTDS(*XML DTD path*)
XMLEBCDIC(*EBDCDIC indicator*)

> XMLSEGINP(*line break indicator*)
> XMLSTDID(*destination EDI standard ID*)
> XMLVALIDATE(*XML validation indicator)*

DEENVELOPE AND TRANSLATE command examples

Example 1:

Deenvelope and translate the EDI documents in the receive file specified by requestor ID **DEPTA7F**.

```
PERFORM DEENVELOPE AND TRANSLATE
    WHERE REQID(DEPTA7F)
```

Example 2:

Deenvelope and translate the EDI documents in the receive file specified by requestor ID **DEPTA7F** into raw data format. Return the information records to the exception file.

```
PERFORM DEENVELOPE AND TRANSLATE
    WHERE REQID(DEPTA7F) RAWDATA(Y) OPTRECS(I)
```

DELETE PROFILE command

This command deletes profile members. However, you can not delete a trading partner profile member while it is still associated with a translation usage/rule.

Syntax

DELETE PROFILE
ID(*DataInterchange profile name*)
MEMBER(*member profile name*)

DELETE PROFILE command examples

Example 1:

Delete trading partner profile member **TPMEM**.

```
PERFORM DELETE PROFILE
    WHERE ID(TPPROF) MEMBER(TPMEM)
```

Example 2:

Delete mailbox (requestor) profile member **REQMEM**.

```
PERFORM DELETE PROFILE
    WHERE ID(REQPROF) MEMBER(REQMEM)
```

DELETE USAGE command

This command deletes send or receive usages. Default for DIR is 'R'. If both APPRECID and APPSNDID are specified, APPSNDID will be used. Any management reporting statistics related to the Usages deleted will also be deleted.

Syntax

DELETE USAGE
AGENCY(*Responsible agency code, 8 characters, default is blank*)
APPLTPID(*Application Trading Partner nickname*)
APPRECID(*Application Receiver ID, 35 characters*)
APPSNDID(*Application Sender ID, 35 characters*)
DIR(*S(send) or R(receive), default is R*)
REL(*Release, 8 characters, default is blank*)
TPID(*Internal Trading partner ID, 35 bytes*)
TPNICKN(*EDI Trading Partner Nickname*)
MAPID(*Mapname, 30 characters*)
VERSION(*Version, 8 characters, default is blank*)

DELETE USAGE command examples

Example 1:

Delete send usage on map **mapname** where application trading partner nickname is **atpid** and edi trading partner nickname is **etpid** and internal trading partner id is **intpid**.

```
PERFORM DELETE USAGE
WHERE DIR(S) MAPID(mapname) APPLTPID(atpid)  TPNICKN(etpid) TPID(intpid)
```

Example 2:

Delete receive usage on map **mapname** where application trading partner nickname is **atpid** and edi trading partner nickname is **etpid** and application sender id is **appsenderid** and agency is **agency** and version is **version** and release is **release**.

```
PERFORM DELETE USAGE WHERE DIR(R) MAPID(mapname) APPLTPID(atpid)
TPNICKN(etpid) APPSNDID(appsenderid)
AGENCY(agency) VERSION(version) REL(release)
```

Example 3:

Delete all usages for a given map.

```
PERFORM DELETE USAGE WHERE DIR(S/R) MAPID(mapname)
```

ENVELOPE command

This command takes the EDI transactions from the Transaction Store that are available for enveloping, envelopes them, and places the results in an envelope file. The envelope file is the TD queue specified in the network profile member or a file specified in the command. Your selection criteria determine which transactions go into the interchange envelope. The enveloper sorts the transactions to create the fewest number of functional groups and interchange envelopes. For more information about envelope processing, see "Enveloping services" on page 500.

Syntax

ENVELOPE
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLID(*application ID*)
BATCH(*translated transaction batch ID*)
ENVPRBREAK(*start new envelope*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
FILEID(*processing file ddname*)
FIXEDFILEID(*fixed-to-fixed output ddname*)
FORMAT(*data format ID*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IACCESS(*IEXIT access*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
INMEMTRANS(*transactions in memory*)
ITPBREAK(*new interchange envelope*)
ITYPE(*IEXIT program type*)
NETID(*network ID*)
OPTRECS(*optional record type*)
OUTFILE(*output data file name for SAP Status*)
OUTTYPE(*output data file type for SAP Status*)
PAGE(*pageable translation*)
RAWDATA(*translate to raw data format*)
RECOVERY(*recovery unit of work*)
SAPUPDT(*track SAP status*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
STDTRID(*EDI standard transaction set ID*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)
VERIFY(*verify transaction status*)

ENVELOPE

> **XML**(*XML required*)
> **XMLDICT**(*XML dictionary address*)
> XMLDTDS(*XML DTD path*)

ENVELOPE command examples

Example 1:

Envelope all the EDI documents in the Transaction Store that are associated with data format ID **PO850** or **CORP861**, and with network **IINR41**. The NETID keyword is included in both WHERE clauses to make sure that only **IINR41** transactions are selected. (Each WHERE clause is a separate set of selection criteria.

```
PERFORM ENVELOPE
    WHERE FORMAT(PO850) NETID(IINR41)
    WHERE FORMAT(CORP861) NETID(IINR41)
```

Example 2:

Envelope all the EDI documents in the Transaction Store that are associated with data format ID **PO850** or **CORP861**, and with network **IINR41**. Place the results in file **NETQUEUE**. The NETID keyword is included in both WHERE clauses to make sure that only **IINR41** transactions are selected. (Each WHERE clause is a separate set of selection criteria.)

```
PERFORM ENVELOPE
    WHERE FORMAT(PO850) NETID(IINR41) FILEID(NETQUEUE)
    WHERE FORMAT(CORP861) NETID(IINR41)
```

ENVELOPE AND SEND command

This command combines the functions of the ENVELOPE and SEND commands, It takes EDI transactions from the Transaction Store, envelopes them, and places the results in a ddname file specified in the `Transaction data queue` field of the network profile member or in a file specified by the command. Selection criteria determine which transactions go into the interchange envelope. The enveloper sorts the transactions to create the fewest possible number of functional groups and interchange envelopes. WebSphere Data Interchange then sends the enveloped transactions to the network. This composite command performs faster than the separate commands

For more information on determining the number of groups and envelopes needed, see "Enveloping services" on page 500.

Syntax

ENVELOPE AND SEND
ACFIELD(*starting application control field data*)
        TO(*ending application control field data*)
APPLID(*application ID*)
BATCH(*translated transaction batch ID*)
ENVPRBREAK(*start new envelope*)
EPURDATE(*starting transaction purge date*)
        TO(*ending transaction purge date*)
FILEID(*processing file ddname*)
FIXEDFILEID(*fixed-to-fixed output ddname*)
FORMAT(*data format ID*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IACCESS(*IEXIT access*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
INMEMTRANS(*transactions in memory*)
ITPBREAK(*new interchange envelope*)
ITYPE(*IEXIT program type*)
NETID(*network ID*)
OPTRECS(*optional record type*)
OUTFILE(*output data file name for SAP Status*)
OUTTYPE(*output data file type for SAP Status*)
PAGE(*pageable translation*)
RAWDATA(*translate to raw data format*)
RECOVERY(*recovery unit of work*)
SAPUPDT(*track SAP status*)
SERVICESEGVAL(*service segment validation level*)
SEQNUM(*increment network profile member numbers*)
SETCC(*condition codes*)
STDTRID(*EDI standard transaction set ID*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)

ENVELOPE AND SEND

> TRXSTAT(*transaction processing status*)
> TRXTIME(*starting transaction time*) TO(*ending transaction time*)
> VERIFY(*verify transaction status*)

ENVELOPE AND SEND command examples

Example 1:
Envelope and send all documents in the Transaction Store that have a destination of trading partner **PISCES**.

```
PERFORM ENVELOPE AND SEND
    WHERE REQID(IINREQ) TPNICKN(PISCES)
```

Example 2:
Envelope and send all documents in the Transaction Store with batch ID **121401**.

```
PERFORM ENVELOPE AND SEND
    WHERE REQID(IINREQ) BATCH(121401)
```

Example 3:
Envelope and send all documents in the Transaction Store with batch IDs **INB11214** or **GEIS1214**.

```
PERFORM ENVELOPE AND SEND
    WHERE REQID(INB1REQ) BATCH(INB11214)
    WHERE REQID(GEISREQ) BATCH(GEIS1214)
```

Note: These transactions will be sent using different networks.

ENVELOPE DATA EXTRACT command

This command extracts detailed information about enveloped transactions, sorted by trading partner nickname, direction, interchange control number, or receiver ID.

You can use this command to create report data to:

- Report on the number of purchase orders sent in a given period of time
- Report on the total number of bytes sent in a given period of time (this can be useful for charging back costs to other departments based on their EDI usage)
- Create a customized functional acknowledgment tracking report by application or by department; for example, an exception report on purchase orders sent more than 2 days ago that have not been acknowledged
- Create an exception report flagging missing control numbers for inbound envelopes

This command can also be used for functions other than reporting, such as:

- Archiving Transaction Store data
- Loading status data directly into the application by application key; for example, the status for each invoice sent could be loaded into the billing system by invoice number

To extract detailed information about transactions, use the TRANSACTION DATA EXTRACT command (see 111).

For information about wildcard usage, see Table 2 on page 10.

Syntax

ENVELOPE DATA EXTRACT
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLICATION(*write application data record*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
CONCATENATE(*concatenate extract data*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
      TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
      TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GROUP(*write group data record*)

ENVELOPE DATA EXTRACT

GRPCTLNO(*starting sender's group control number*)
        TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
IMAGE(*write image data record*)
INTCTLNO(*starting sender's interchange control nbr*)
        TO(*ending sender's interchange control nbr*)
INTERCHANGE(*write interchange data record*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
RECEIVEACKDATA(*write detailed acknowledgment data*)
RECEIVEACKIMAGE(*write receive acknowledgment record*)
SENDACKDATA(*write detailed acknowledgment data*)
SENDACKIMAGE(*write acknowledgment record*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRANSACTION(*write transaction data record*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
        TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)
USERPGM(*user program*)

ENVELOPE DATA EXTRACT command examples

Example 1:
The EDI users in Company XYZ want to view the status of their EDI documents for the
last month. Specifically, the Purchasing department would like to look up purchase
orders, by purchase order number, to determine the daily status. The report must
contain:
- Document type (purchase order, invoice, and so on)
- Control numbers
- Date and time
- Trading partner nickname
- Reference number (purchase order number, invoice number, and so on)
- Transaction status

The programmers at Company XYZ have decided to use a ENVELOPE DATA
EXTRACT command to collect the necessary information from WebSphere Data
Interchange. They have written a COBOL program, which they run daily, that will format

the records into a customized report. Users will read this customized report, searching on the reference numbers to easily find the status they are interested in.

```
PERFORM ENVELOPE DATA EXTRACT
    SELECTING CONCATENATE(Y) INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
    WHERE HANDLE(20010201) TO(20010328)
```

Figure 1 shows an example of a customized report.

```
                          XYZ COMPANY

       CONTROL      SET                   TRADING  PO # or
TYPE   NUMBER      NUMBER     DATE    TIME  NICKNAME INVOICE # STATUS
----   ----------- ---------- -------- -------- --------- ----------- ---------
PO     00000000375 00000003386 03/02/01 06:02:93 BOBS     PO100332201 ENVELOPED
PO     00000000383 00000003394 03/01/01 11:12:53 BOBS4790 PO100222210 ENVELOPED
PO     00000000388 00000003399 03/02/01 11:33:06 BOBS4790 PO100232115 ENVELOPED
PO     00000000389 00000003400 03/01/01 06:52:23 BOBS4790             ENVELOPED
PO     00000000377 00000003388 03/01/01 11:41:23 BOBS     PO100123130 ENVELOPED
PO     00000000381 00000003392 03/01/01 06:22:56 BOBS4790 PO121312322 ENVELOPED
IN     00000000145 00000000383 03/02/01 12:53:42 DLGLEVEL IN94410     RECEIVED
IN     00000000144 00000000382 03/02/01 09:12:15 DLGLEVEL IN95443     RECEIVED
PO     00010000130 00000004853 03/02/01 09:43:02 DLGOCCUR IN76445     ENVELOPED
PO     00010000130 00000004854 03/01/01 07:25:16 DLGOCCUR IN76835     ENVELOPED
PO     00010000130 00000004855 03/01/01 11:51:06 DLGOCCUR IN76337     ENVELOPED
PO     00010000130 00000004856 03/02/01 06:34:29 DLGOCCUR IN76838     ENVELOPED
```

Figure 1. Example of a customized report produced by the ENVELOPE DATA EXTRACT command

Example 2:

Retrieve the interchange, group, and transaction information for all transactions in a particular envelope with trading partner name **PISCES** and interchange control number **000008888**. Place the results in the EDIQUERY file.

```
PERFORM ENVELOPE DATA EXTRACT
    SELECTING INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
    WHERE TPNICKN(PISCES) INTCTLNO(000008888) DIR(S)
```

Example 3:

Retrieve the interchange, group, and transaction information for all envelopes with an application sender ID of **SERVO**. Include application and image information.

```
PERFORM ENVELOPE DATA EXTRACT
    SELECTING INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y) APPLICATION(Y)
     IMAGE(Y)
    WHERE APPSNDID(SERVO) DIR(S)
```

EXPORT command

This command uses the control file to extract setup information from WebSphere Data Interchange and write records to one or more files. ("Export/Import control file (CTLFILE)" on page 209 for ddname*s*.) You can request a particular file record format with the `EIFORMAT` keyword. If the Export/Import file is empty, the requested format is used. If the Export/Import file is not empty, the format of the data already in the file is used.

Syntax

> **EXPORT**
> ASSOBJ(*associated objects.*)
> **CTLFILE**(*control file ddname*)
> CTLTYPE(*control file type*)
> EIFORMAT(*requested export file record format*)
> IFCC(*override condition codes*)
> KEYID(*ID*)
> MBRNAME(*member name*)
> NOMSG(*print extraneous messages*)
> SETCC(*condition codes*)

EXPORT command example

Export the information in fixed format using a control file named EXOUT.

```
PERFORM EXPORT
    WHERE CTLFILE(EXOUT) EIFORMAT(FIXED)
```

GLB DUMP command

This command is available only in the CICS environment, and is not required for normal processing. However, this command can be used to gather information required by WebSphere Data Interchange support personnel when debugging problems related to the persistent environment. The dump is written to the file associated with ddname EDIGDMP1 in the CICS startup JCL.

Syntax

GLB DUMP
IFCC(*override condition codes*)
LEVEL(dump or trace level)
RESET(*reset output file*)
SETCC(*condition codes*)

GLB DUMP command examples

Example 1:

Open EDIGDMP1 for output and generate a complete persistent environment dump.

```
PERFORM GLB DUMP
    WHERE LEVEL(1) RESET(Y)
```

Example 2:

Open EDIGDMP1 for append and generate a persistent environment dump of just the data area.

```
PERFORM GLB DUMP
    WHERE LEVEL(12)
```

GLB TRACE command

This command is available only in the CICS environment and is not required for normal processing. However, you can use this command to gather information required by WebSphere Data Interchange support personnel when debugging problems related to the persistent environment. The trace is written to the file associated with ddname EDIGTRC1 in the CICS startup JCL.

Syntax

GLB TRACE
IFCC(*override condition codes*)
LEVEL(dump or trace level)
RESET(*reset output file*)
SETCC(*condition codes*)

GLB TRACE command examples

Example 1:
Open EDIGTRC1 for output and start a persistent environment trace for as much detail as possible.

```
PERFORM GLB TRACE
    WHERE LEVEL(3) RESET(Y)
```

Example 2:
End the persistent environment trace.

```
PERFORM GLB TRACE
    WHERE LEVEL(0)
```

HOLD command

This command puts a transaction into held status. While in held status, no actions are permitted against the transaction that would change the status of the transaction. Nor is it purged automatically when its storage time expires. If the transaction is one of a related group, all transactions in the group are placed in held status. The RELEASE command restores held transactions to their former status or, if their storage time expired during the hold period, changes them to STORE-TIME-EXPIRED status.

The HOLD, PURGE, RELEASE, and UNPURGE commands share a common syntax.

Syntax

HOLD
ACFIELD(*starting application control field data*)
        TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
        TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
        TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
        TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
        TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
        TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(network transaction status)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)

TRXCTLNO(*starting transaction set control number*)
TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

HOLD command example

Place in hold status all EDI documents destined for trading partner **PISCES** that were translated on December 14, 2001.

```
PERFORM HOLD
    WHERE TPNICKN(PISCES) TRXDATE(01/12/14) DIR(S)
```

IMPORT command

This command uses the control file to read a file or group of files for importing setup information.

Syntax

IMPORT
ACTUSAGE(*activate imported usage*)
CTLFILE(*control file ddname*)
CTLTYPE(*control file type*)
DUPCHECK(*perform duplicate checks*)
IFCC(*override condition codes*)
KEYID*ID*
MBRNAME*member name*
REPLACE*replace the same named object*
NOMSG(*print extraneous messages*)
SETCC(*condition codes*)
USAGETID*map name*
WRTCTLNO(*import trading partner control number*)

IMPORT command example

Import all records specified by the batch control file in TS queue **INCTL**.

```
PERFORM IMPORT
    WHERE CTLFILE(INCTL) CTLTYPE(TS)
```

LOAD LOG ENTRIES command

This command copies the selected HOLDFILE records back into the event log table. In a DB2 environment, you can restore deleted records to the event log by specifying the `ARCHIVEFILE` value from the UNLOAD LOG ENTRIES command as the value for the `HOLDFILE` keyword on this command.

Syntax

LOAD LOG ENTRIES
APPLID(*application ID*)
HOLDFILE(*event log hold file name*)
HOLDTYPE(*event log hold file type*)
IFCC(*override condition codes*)
LOGAEID(*starting event log associated entry ID*)
    TO(*ending event log associated entry ID*)
LOGDATE(*starting event log date*) TO(*ending event log date*)
LOGFORM(*starting event log format ID*) TO(*ending event log format ID*)
LOGTIME(*starting event log time*) TO(*ending event log time*)
LOGUSER(*starting event log user ID*) TO(*ending event log user ID*)
NEWAPPLID(*new application ID*)
SETCC(*condition codes*)

Load log entries command example

Copy the held records for **EDIFFS** with a log date of **12/14/01** back into the event log table.

```
PERFORM LOAD LOG ENTRIES
    WHERE APPLID(EDIFFS) LOGDATE(12/14/01)
```

NETWORK ACTIVITY DATA EXTRACT command

This command collects data about the network activity of the requestors defined in the mailbox (requestor) profile and generates reports based on your criteria. The information collected includes the total number of envelopes or characters sent by each requestor for any given day or range of days.

You can use this command to create reports to:
- Reconcile network charges
- List network usage by user ID
- List heaviest network users
- List inactive requestors

For information about using wild cards, see Table 2 on page 10.

Syntax

NETWORK ACTIVITY DATA EXTRACT
ACCTID(*network account ID*)
DAYS(*starting date*) TO(*ending date*)
DYNSQL(*use dynamic SQL*)
IFCC(*override condition codes*)
NETID(*network ID*)
NETNAME(*network name*)
REQID(*requestor ID*)
SETCC(*condition codes*)
USERID(*network user ID*)
USERPGM(*user program*)

NETWORK ACTIVITY DATA EXTRACT command examples

Example 1:

Your network charges jumped significantly last month and you want to understand which of your applications has increased its network usage. To do this, you need a report that shows the:
- Account ID
- User ID
- Number of envelopes exchanged, both sent and received
- Number of characters exchanged, both sent and received

```
PERFORM NETWORK ACTIVITY DATA EXTRACT
    WHERE DAYS(01/08/01) TO(01/08/31)
```

The data resulting from this command was written to file EDIQUERY, sorted by a local sort utility, and used as input to a user-written program that created the report shown in Figure 2 on page 46.

NETWORK ACTIVITY DATA EXTRACT

```
                  Network Traffic Activity Report              Date: 01/09/20
                                                               Time: 12:31:02

 Interchange    Total
 Month  Network Name  Net ID   Account ID  User ID   S/R  Envelopes   Characters
 -----  ------------  -------  ----------  --------  ---  ----------  ----------
 08/01  AT&T Global   IINR41   XTEV        PURCHASE   S    7776          9998376
        Network                                       R    4356          2333157
                                           ACCTPAYB   S    1526          8867837
                                                      R     662           333457
                                                          ----------  ----------
                                                           14320         21532827
                               XTEW        RECEIVNG   S     856            82251
                                                      R    2856           434457
                                           TRAFFIC    R     770            66615
                                                          ----------  ----------
                                                           4482           583323
                      IINCICS XTEX        PURCHASE    S    4476          2648376
                                                      R     756           453457
                                           PRODPO     S    6625           783890
                                                      R     559           467457
                                                          ----------  ----------
                                                           12416         11408192
```

Figure 2. Example of a network traffic activity report

Example 2:

Extract information to reconcile your EDI traffic through a particular account (**CMPY**) and user ID (**USER21**) on the network for the month of May during 2001. Include a list of the requestors that use that account/user ID on the network, and the number of messages sent and received by each requestor (only the traffic through WebSphere Data Interchange for this account and user ID).

```
PERFORM NETWORK ACTIVITY DATA EXTRACT
    WHERE ACCTID(CMPY) USERID(USER21) DAYS(01/05/01) TO(01/05/31)
```

PRINT ACKNOWLEDGMENT IMAGE command

This command writes an image to RPTFILE of the functional acknowledgments for the transactions that match your selection criteria. The image does not include the envelope segments.

Syntax

PRINT ACKNOWLEDGMENT IMAGE
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
      TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
      TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
      TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
      TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
MERGED(*merge transaction image with functional acknowledgment*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
SEGMENTED(*print segment on new line*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
      TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)

PRINT ACKNOWLEDGMENT IMAGE

TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

PRINT ACKNOWLEDGMENT IMAGE command example

Print functional acknowledgment images for all EDI documents with batch ID **121401**.

```
PERFORM PRINT ACKNOWLEDGMENT IMAGE
    WHERE BATCH(121401)
```

Figure 3 shows an example of the Functional Acknowledgment Image report.

```
TF13        Functional Acknowledgment Image        Date:  01/12/14
                                                   Time:  12:12:12


Trading partner nickname:  PISCES
Transaction handle  . . :  20011214101533000001
Transaction status  . . :  Transaction accepted


AK1*IN*40088!AK2*810*000048118!AK5*A!AK9*A*1*1*1!
```

Figure 3. Example of a Functional Acknowledgement Image report

PRINT ACTIVITY SUMMARY command

This command creates a summary of activity for the inbound and outbound transactions that match your selection criteria. The summary is written to RPTFILE.

Syntax

PRINT ACTIVITY SUMMARY
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
      TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
      TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
      TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
      TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
MERGED(*merge transaction image with functional acknowledgment*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
SEGMENTED(*print segment on new line*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
      TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

PRINT ACTIVITY SUMMARY command example

Print the transaction Activity Summary Report for all EDI documents entering the Transaction Store between November 29 and November 30, 2001.

```
PERFORM PRINT ACTIVITY SUMMARY
   WHERE TRXDATE(01/11/29) TO(01/11/30)
```

Note: This example is only valid when the date mask in the language profile (LANGPROF) is &Y/&M/&D.

Figure 4 shows an example of the Activity Summary Report.

```
TF11                 Activity Summary Report              Date: 01/12/14
                                                          Time: 12:12:12


         Outbound Transactions          Count
         Selected transactions . . . . :    1940
            Translation . . . . . . . . :    1940
              Acceptably translated . . :    1930
              Unacceptably translated . :      10
            Enveloping  . . . . . . . . :    1000
              Enveloped . . . . . . . . :     900
              Enveloping errors . . . . :     100
            Send requests . . . . . . . :     900
              Sent  . . . . . . . . . . :     800
                Pending functional ack  :      10
                Pending network ack . . :      10
              Not sent  . . . . . . . . :     100
                Error on request to send:      73
                Network error . . . . . :      27
            Detached  . . . . . . . . . :       0


         Inbound Transactions           Count

         Selected transactions . . . . :    1000
            Acceptably translated . . . :     800
            Unacceptably translated . . :      10
            Not yet translated  . . . . :     190
            Detached  . . . . . . . . . :       0
```

Figure 4. Example of an Activity Summary Report

PRINT EVENT LOG command

This command writes an image of event log entries to RPTFILE for the transactions that match your selection criteria.

Syntax

PRINT EVENT LOG
ACFIELD(*application control field data*)
APPLICATION(*write application data record*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
CONCATENATE(*concatenate extract data*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
        TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
        TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GROUP(*write group data record*)
GRPCTLNO(*starting sender's group control number*)
        TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
IMAGE(*write image data record*)
INTCTLNO(*starting sender's interchange control nbr*)
        TO(*ending sender's interchange control nbr*)
INTERCHANGE(*write interchange data record*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
RECEIVEACKDATA(*write detailed acknowledgment data*)
RECEIVEACKIMAGE(*write receive acknowledgment record*)
SENDACKDATA(*write detailed acknowledgment data*)
SENDACKIMAGE(*write acknowledgment record*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)

TRANSACTION(*write transaction data record*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)
USERPGM(*user program*)

PRINT EVENT LOG command example

Print log entries for all EDI documents that encountered errors during send processing on December 12, 2001 for trading partner **ABMETAL1**.

```
PERFORM PRINT EVENT LOG
    WHERE TRXSTAT(42) TRXDATE(01/12/14) TPID(ABMETAL1)
    WHERE TRXSTAT(43) TRXDATE(01/12/14) TPID(ABMETAL1)
```

Figure 5 on page 53 shows an example of the Events Logging Report.

```
                        EVENTS LOGGING REPORT

                        REQUESTOR . . . . : USER21

                        DATE  . . . . . . : 01/12/14

                        TIME  . . . . . . : 08:37:36

                    Selection Criteria for Log Report

                    Application ID . . . . :  EDIFFS
                    Date . . . . . . . . . :  01/07/21   To  01/07/28
                    Time . . . . . . . . . :
                    User ID  . . . . . . . :
                    Format ID  . . . . . . :
                    Associated Entry ID  . :  20010721164325210000
   EVENTS LOGGING REPORT  DATE:  01/12/14     TIME:  08:37:36     Page  1

 Date . . . . . . . . . : 01/07/21
 Time . . . . . . . . . : 16:43:26
 Entry ID . . . . . . . : E000000000000000000000000000000020010721164326213000000
 Associated Entry ID  . : 20010721164325210000
 User ID  . . . . . . . : USER21
 Job ID . . . . . . . . : EDITR
 Application ID . . . . : EDIFFS
 Format ID  . . . . . . : $$STD-'++:.?
 Data . . . . . . . . . : UNB+UNOA:1+GBLLD.LLD324+GBLLD.LLD053:ZZ+930602:1646+
40000092++PLACIN++1++1'UNH+40000200+LIMUWM:1:2:LI'RFF+UMR+B9999LIMUWMUMR1'RFF+TR+B
9999LIMUWMTR1'RFF+BQR+B9999LIMUWMBQR1' NAD+BK+9999'NAD+BU+LL'NAD+CS+10
AEC'GIS+02:04'NAD+IV+ LINE 1:LL'GIS+01:26'GIS+01:15'GIS+Y:16'
RFF+LIR+0001'RFF+CS1+COSLINE 1'RFF+CS1+COSLINE
1'RFF+LIN+001'PCD+001:1.0000001'PCD+002:2.0000002'
PCD+003:3.0000003'PCD+004:4.0000004' PCD+005:5.0000005'L01+034+UNDERWRITER MESSAGE
LINE 1  TE'L02+1::LINE ONE OF THE INVITEDITER LINE 1  TEXT+3::LINE TWO OF THE
INVITEDITER LINE 1  TEXT+::LINE THREE  THE INVITEDITER LINE 1  TEXT'NAD+IV+
LINE2:LL'GIS+02:26'GIS+01:15'GIS+Y:16'RFF+LIR+0002'FF+CS1+COSLINE
2'RFF+CS1+COSLINE 2'RFF+LIN+002'PCD+001:1.0000001'PCD+002:2.0000002'
PCD+003:3.0000003' PCD+004:4.0000004'PCD+005:5.0000005'L01+034+UNDERWRITER MESSAGE
LINE 2  TE' L02+1::LINE ONE OF THE INVITEDITER LINE 2  TEXT+3::LINE TWO OF THE
INVITEDITER LINE 2  TEXT+::LINE THREE  THE INVITEDITER LINE 2
TEXT'UNT+39+40000200'UNZ+1+40000092'
The number of records printed from the input file was 2
Printing has terminated. Return code is 0; extended return code is 0.
```

Figure 5. Example of an Events Logging Report

PRINT STATUS SUMMARY command

This command extracts status information about the transactions that match your selection criteria and writes the information to RPTFILE. This report shows transaction status (translated, enveloped), network status (delivered, purged), and store status (active, held, marked for purging). An **R** (related) after the transaction handle indicates the transaction is a part of a bundle.

Status summaries are not available for online viewing.

Syntax

PRINT STATUS SUMMARY
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
      TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
      TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
      TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
      TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
MERGED(*merge transaction image with functional acknowledgment*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
SEGMENTED(*print segment on new line*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)

TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

PRINT STATUS SUMMARY command example

Print the status summary for all EDI documents that were translated on December 14, 2001.

```
PERFORM PRINT STATUS SUMMARY
    WHERE TRXDATE(01/12/14)
```

Figure 6 shows an example of the Status Summary Report for Outbound Transactions. For more information on the fields included in this report, refer to the WebSphere Data Interchange for z/OS Administration Guide.

```
TF80                                Status Summary Report for      Date: 01/12/14
                                      Outbound Transactions         Time: 12:12:12


Transaction Handle   Trading Partner Nickname    Data Format ID     Transaction Status
    Date Enveloped       Interchange Cntrl No    Network Status         Group Control No
200112140930120000    Partner111111111           Format1111111111  Transaction accepted
    01/12/14            12121212121212          Accepted by network   11111111111111

200112121045330001    Partner111111111           Format3333333333  Send translate error

200112141033110000    Partner222222222           Format2222222222  Transaction accepted
    01/12/14            22222222222222          Accepted by network   11111111111111

200112141033110004 R Partner222222222            Format3333333333  Transaction accepted
    01/12/14            22222222222222          Accepted by network   22222222222222

200112141033110003    Partner333333333           Format1111111111  Transaction accepted
    01/12/14            33333333333333          Recall request error  11111111111111
    01/12/14            33333333333333          Not sent - net error  22222222222222
```

Figure 6. Example of the Status Summary Report for Outbound Transactions

The same report format is used for Status Summary reports for Inbound Transactions, shown in Figure 7 on page 56.

PRINT STATUS SUMMARY

```
TF80                                    Status Summary Report for     Date: 01/12/14
                                          Inbound Transactions        Time: 12:12:12

Transaction Handle    Trading Partner Nickname    Standard Trans ID  Transaction Status
          Date Translated    Data Format ID         Translation Status
200112141344270000    Partner222222222            12121212          Receive translated
          01/12/14           Format1212121212     Acceptable

200112141010100001    Partner222222222            12345678          Receive translated
          01/12/14           Format3333333333     Acceptable
          01/12/14           Format3333333333     Unacceptable
```

Figure 7. Example of the Status Summary Report for Inbound Transactions

PRINT STATUS SUMMARY2 command

This command extracts the same information as the PRINT STATUS SUMMARY command but adds a line containing the application control number and the internal trading partner ID for each transaction. An **R** (related) after the transaction handle indicates the transaction is part of a bundle.

Status summaries are not available for online viewing.

Syntax

PRINT STATUS SUMMARY2
ACFIELD(*starting application control field data*)
    TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
    TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
    TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
    TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
    TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
    TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
MERGED(*merge transaction image with functional acknowledgment*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
SEGMENTED(*print segment on new line*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)

PRINT STATUS SUMMARY2

```
        TRXCTLNO(starting transaction set control number)
              TO(ending transaction set control number)
        TRXDATE(starting transaction date) TO(ending transaction date)
        TRXSTAT(transaction processing status)
        TRXTIME(starting transaction time) TO(ending transaction time)
```

PRINT STATUS SUMMARY2 command example

Print transaction images for all EDI documents with receive translation errors.

```
PERFORM PRINT STATUS SUMMARY2
    WHERE TRXSTAT(73)
```

Figure 8 shows an example of the Selection Criteria for Status Summary Report.

```
1                         Selection Criteria for              Date: 01/12/14
                           Status Summary Report               Time: 14:25:44


SELECTION CRITERIA        1

Transaction handle. . . . . . . . : 20010323000000000000  TO  20010323240000000000
1TF80                        Status Summary Report for               Date:  01/12/14
                        Outbound Transactions                   Time:  14:25:44


Transaction Handle  Trading Partner Nickname  Data Format ID  Transaction Status Store Status
       Application Control Number                 Internal Trading Partner ID
Date Enveloped  Interchange Cntrl No  Network Status    Group Control No  Func Ack Status

200103230921579000  BOBS                         BOBS8004833     ENVELOPED    ACTIVE
                                                 1111122222
01/12/14          00000000000418      ENVELOPED          00000000000399    PENDING

20011214116562000   BOBS                         BOBSPORDER      ENVELOPED    ACTIVE
        BOBS     12345678                         BOBS
01/12/14          00000000000419      ENVELOPED          00000000000400    NOT REQUESTED
```

Figure 8. Example of the Selection Criteria for Status Summary Report

PRINT TRANSACTION DETAILS command

This command extracts detailed information about the transactions that match your selection criteria and writes the information to RPTFILE.

Syntax

PRINT TRANSACTION DETAILS
ACFIELD(*starting application control field data*)
     TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
     TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
     TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
     TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
     TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
     TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
MERGED(*merge transaction image with functional acknowledgment*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
SEGMENTED(*print segment on new line*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
     TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)

PRINT TRANSACTION DETAILS

           TRXSTAT(*transaction processing status*)
           TRXTIME(*starting transaction time*) TO(*ending transaction time*)

PRINT TRANSACTION DETAILS command example

Print transaction details for all EDI documents that were placed in the Transaction Store on December 14, 2001.

```
PERFORM PRINT TRANSACTION DETAILS
    WHERE TRXDATE(01/12/14)
```

Figure 9 on page 61 shows an example of the Transaction Details report. For more information on the fields included in this report, refer to the WebSphere Data Interchange for z/OS Administration Guide.

```
TF73                 Transaction Details              Date:  01/12/14
                                                      Time:  12:12:12


Transaction handle . . . . . . . . . . : 20011214101533000001
Trading partner nickname . . . . . . . : PISCES
Internal trading partner ID  . . . . . : 23456789012345678901234567890012345
Direction  . . . . . . . . . . . . . . : SEND
Data format ID . . . . . . . . . . . . : POSEND
Application control number . . . . . . : 12345678901234567890123456789012345
Interchange control number . . . . . . : 12345678901234
Group control number . . . . . . . . . : 12345678901234
Transaction control number . . . . . . : 12345678901234
Transaction status . . . . . . . . . . : Translated
Added to store . . . . . . . . . . . . : 01/12/14-10:29:48
Store status   . . . . . . . . . . . . : Active
Delivered to application . . . . . . . :
Translation error level  . . . . . . . : 1
Translation  . . . . . . . . . . . ..: Acceptable
Network acknowledgment requested . . . : D
Network status . . . . . . . . . . . . :
Functional acknowledgment  . . . . . . :
Functional ack date  . . . . . . . . . :
Application ID . . . . . . . . . . . . : EDIMP
Batch ID . . . . . . . . . . . . . . . : BATCH123
Earliest purge date  . . . . . . . . . : 01/12/14
Earliest envelope date . . . . . . . . : 01/12/14
Enveloped/deenveloped  . . . . . . . . :
Sent . . . . . . . . . . . . . . . . . :
Envelope profile member  . . . . . . . : X12V2R2
Standard ID  . . . . . . . . . . . . . : X12V2R2
Standard version . . . . . . . . . . . : V2
Standard level . . . . . . . . . . . . : R2
Standard transaction ID  . . . . . . . : 850
Network ID . . . . . . . . . . . . . . : IN
Usage indicator  . . . . . . . . . . . : P
Total segment count  . . . . . . . . . : 22
Transaction size (bytes) . . . . . . . : 742
```

Figure 9. Example of the Transaction Details report

PRINT TRANSACTION IMAGE command

This command writes an image of the transactions that match your selection criteria to RPTFILE. The images do not include envelope or security segments.

Syntax

PRINT TRANSACTION IMAGE
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
      TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
      TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
      TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
      TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
MERGED(*merge transaction image with functional acknowledgment*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
SEGMENTED(*print segment on new line*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
      TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

PRINT TRANSACTION IMAGE command example

Print transaction images for all EDI documents with receive translation errors.

```
PERFORM PRINT TRANSACTION IMAGE
    WHERE TRXSTAT(73)
```

Figure 10 shows an example of the Transaction Image report.

```
TF75                      Transaction Image              Date:  01/12/14
                                                         Time:  12:12:12


Trading partner nickname:  PISCES
Transaction handle  . . :  20011214101533000001
Transaction status  . . :  Receive trans error

SEG1*dATA:dATAN**dANTA:dANTAN:-99:-99!SEG2*123456:-1.00:-100*12.3456:-
1.00:010920:123000:-10000!SEG3*123.456:99:001128:083
0:990*1234.56:-1:010920:1259:-1*1234.56:-1.0000:010920:125900:-
100!SEG4*15:1.00:001128:1259:1000*15:-1.0:001128:1259:99*C1

C2:D1D2D3D4D5:1.001128:1259:1000!SEG5*001128*1259!TOTALS*210.000000*96.000000*3
28.000000*196.000000*32.000000*8.000000!
```

Figure 10. Example of the Transaction Image report

PROCESS command

This is a generic command that tells WebSphere Data Interchange to execute the commands in a service profile. It allows you to author and maintain your commands (along with their data sources and targets) in the client tool, rather than in the programs that invoke WebSphere Data Interchange. You name and specify the services you want WebSphere Data Interchange to provide to your applications by creating service profiles. You then have your applications call WebSphere Data Interchange and submit a **PROCESS** command naming the services you created. If you decide to change the command a service invokes, or specifiy different options for the command, you simply update the service profile. You do not have to update or recompile your application programs. Complex services may require multiples commands and in addition, different input messages may require different processing. The **PROCESS** command invokes the ″command chaining″ feature of WebSphere Data Interchange that will chain together multiple service profiles based upon the destination of the output data from the previous command in the chain. The **PROCESS** and **RECEIVE AND PROCESS** commands are the only commands that invoke the command chaining feature in WebSphere Data Interchange.

Syntax

PROCESS
FILEID(*logical name of the source data*)

PROCESS command examples

Example 1:

XML purchase orders are sent from the purchasing system to WebSphere Data Interchange to be transformed into EDI and sent to the trading partner. A service profile called **XML_IN** is created in the WebSphere Data Interchange client tool. Each XML purchase order is written to a file whose logical name is set to **XML_IN** and then WebSphere Data Interchange is invoked with the following command:

```
PERFORM PROCESS
   WHERE FILEID(XML_IN)
```

Example 2:

Vendor catalogs in EDI format are received from trading partners using FTP. The catalogs must be translated into the internal proprietary XML format and loaded into the internal application. A service profile called **EDI_IN** is created in the WebSphere Data Interchange client tool and each time a catalog is received, the logical name of the file received is set to **EDI_IN** and then WebSphere Data Interchange is invoked with the following command:

```
PERFORM PROCESS
   WHERE FILEID(EDI_IN)
```

PROCESS NETWORK ACKS command

This command processes acknowledgments that have already been received into a file. It is used internally by WebSphere Data Interchange when network acknowledgments are processed on a continuous receive basis. This command can also be used if network acknowledgments are received outside WebSphere Data Interchange control, but must still be applied to the Transaction Store.

Syntax

PROCESS NETWORK ACKS
ACKFILE(*network acknowledgment ddname*)
ACKTYPE(*acknowledgment file type*)

PROCESS NETWORK ACKS command example

Process a file allocated to ddname **PROCACKS** containing network acknowledgments associated with requestor ID **IINB41REQ**.

```
PERFORM PROCESS NETWORK ACKS
    WHERE REQID(IINB41REQ) ACKFILE(PROCACKS)
```

PURGE command

This command marks a transaction for purging from the Transaction Store but does not remove it. Transactions are marked for purging when old outbound transactions are still pending functional reconciliation. Transactions marked for purge are deleted from the store when you execute the REMOVE TRANSACTIONS command. For more information, see "REMOVE TRANSACTIONS command" on page 89. If the transaction is one of a related group, all transactions in the group are marked for purging. Only transactions with status of PURGE-USER REQUEST or PURGE-DATE EXPIRED are eligible for purging. Reconciled transactions are automatically marked PURGE-DATE EXPIRED after the purge internal has expired.

The default expiration date is 30 days from translation. You can change the default by specifying a PURGINT value on PERFORM statements that add transactions to the Transaction Store (for example, TRANSLATE AND ENVELOPE). See "PURGINT" on page 167 for applicable PERFORM commands.

The HOLD, PURGE, RELEASE, and UNPURGE commands share a common syntax.

Syntax

PURGE
ACFIELD(*starting application control field data*)
        TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
        TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
        TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
        TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
        TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
        TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)

SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
        TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

PURGE command example

Mark for purging all EDI documents that have been delivered to trading partner **PISCES** and accepted.

```
PERFORM PURGE
    WHERE TPNICKN(PISCES) TRXSTAT(61)
```

QUERY command

This command generates a list of transactions that meet the selection criteria you specify in the keywords. In the returned list, transactions are identified by their transaction handle (`Transaction Store ID`). WebSphere Data Interchange writes the list of handles to the specified file (ddname EDIQUERY for z/OS). Each item in the list is 30 bytes long and has the format shown in Table 5:

Table 5. Transaction handle format

Bytes	Handle format
1-10	Packed decimal
11-30	EBCDIC character

You can use the QUERY command to provide input for a user-written reporting program.

Note: In CICS, you can pass the file name and type to WebSphere Data Interchange as parameters.

Syntax

QUERY
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
      TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
      TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
      TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
      TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
NETACKP(*pending network acknowledgment*)

NETID(*network ID*)
NETSTAT(*network transaction status*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
       TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

QUERY command example

Return a list of transactions with batch ID **121401** and translation error level **2** (data element and segment errors).

```
PERFORM QUERY
    WHERE BATCH(121401) TRERLVL(2)
```

QUERY PROFILE command

You can query for information about defined profiles. The results are written in tagged, fixed, or native format to the file specified in the command. The default format is **FIXED**. The default output file name is **EDIQUERY**, and the default output file type in CICS is **TS**. If no member is specified in the command, the entire profile is queried for the requested information.

Syntax

QUERY PROFILE
ID(*DataInterchange profile name*)
IFCC(*override condition codes*)
MEMBER(*member profile name*)
OUTFILE(*output file name*)
OUTFORMAT(*output format*)
OUTTYPE(*output file type*)
SETCC(*condition codes*)

QUERY PROFILE command examples

Example 1:

Query all trading partner profile members and write results to **MYFILE** in fixed format.

```
PERFORM QUERY PROFILE
    WHERE ID(TPPROF) OUTFILE(MYFILE)
```

Example 2:

Query mailbox (requestor) profile member **REQMEM** and write results to **MYFILE** in the original format.

```
PERFORM QUERY PROFILE
    WHERE ID(REQPROF) MEMBER(REQMEM) OUTFILE(MYFILE) OUTFORMAT(N)
```

RECEIVE command

This command receives EDI data from the network identified in the mailbox (requestor) profile member. It places the data in the receive file specified in the mailbox (requestor) profile or in the file specified in the command.

Syntax

RECEIVE
CLEARFILE(*clear specified file contents*)
ENVTYPE(*transaction envelope type*)
FILEID(processing file ddname)
IFCC(*override condition codes*)
MSGUCLASS(*override message user class*)
ONEMSG(*read only one MQ message*)
REQID(requestor ID)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RECEIVE command examples

Example 1:

Receive data for requestors defined in mailbox (requestor) profile members **DEPTA7F** and **DEPTB8R**. Place the data into the receive files specified by the profile members. Clear both files before the data is received. The data is in X12 format.

```
PERFORM RECEIVE
    WHERE REQID(DEPTA7F) CLEARFILE(Y)
    WHERE REQID(DEPTB8R) CLEARFILE(Y)
```

Example 2:

In CICS, receive data for the requestor defined by mailbox (requestor) profile member **DEPTA7F**. Place the data in CICS temporary storage queue **RECVPO**. The data is in EDIFACT format.

```
PERFORM RECEIVE
    WHERE REQID(DEPTA7F) FILEID(RECVPO) ENVTYPE(E)
```

RECEIVE AND DEENVELOPE command

This command combines the functions of the RECEIVE and DEENVELOPE commands. It receives EDI data from the network indicated in the mailbox (requestor) profile member and places the data in the receive file specified by the mailbox (requestor) profile member or in the file specified in the command. It then removes the envelope segments and places the results in the Transaction Store. This composite command performs faster than the separate commands.

Syntax

RECEIVE AND DEENVELOPE
CLEARFILE(*clear specified file contents*)
DIERRFILTER(*initial error filter set*)
DUPENV(*process duplicate envelopes*)
ENVTYPE(*transaction envelope type*)
EXTENDC(*translate with extended C record format*)
FADELAY(*delay queuing functional acknowledgment*)
FILEID(*processing file ddname*)
FORCETEST(*force test usage*)
FUNACKFILE(*functional acknowledgment ddname*)
FUNACKREQ(*require functional acknowledgment envelope file*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
INMEMTRANS(*transactions in memory*)
MRREQID(*management reporting requestor ID*)
MSGUCLASS(*override message user class*)
ONEMSG(*read only one MQ message*)
OPTRECS(*optional record type*)
OUTFILE(*output data file name for SAP Status*)
OUTTYPE(*output data file type for SAP Status*)
PAGE(*pageable translation*)
PURGINT(*purge interval*)
RECOVERY(*recovery unit of work*)
REQID(*requestor ID*)
SAPUPDT(*track SAP status*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RECEIVE AND DEENVELOPE command examples

Example 1:

Receive all EDI documents for requestor **DEPTA7F** and deenvelope them. Place them in the Transaction Store. The documents are in X12 format.

```
PERFORM RECEIVE AND DEENVELOPE
   WHERE REQID(DEPTA7F)
```

Example 2:

Receive all EDI documents for requestor **DEPTA7F**, deenvelope them. Place them in the Transaction Store. Set the purge interval for these documents at 20 days. The documents are in EDIFACT format.

```
PERFORM RECEIVE AND DEENVELOPE
    WHERE REQID(DEPTA7F) PURGINT(20) ENVTYPE(E)
```

RECEIVE AND PROCESS command

This is a combined command that tells WebSphere Data Interchange to do a **PERFORM RECEIVE** followed by a **PERFORM PROCESS**. (See "RECEIVE command" on page 71and "PROCESS command" on page 64 for more information).

The **RECEIVE AND PROCESS** command is most commonly used in trigger programs. For example, when a message arrives on a WebSphere MQ queue, the WebSphere Data Interchange Adapter for WebSphere MQ (a trigger program) invokes WebSphere Data Interchange and passes it a **RECEIVE AND PROCESS** command which causes WebSphere Data Interchange to receive the message off of the queue and process it as specified in the associated Service Profile. The valid keywords on this command are the keywords that are valid for a **PERFORM RECEIVE** command. (see "Syntax" on page 71 for more information).

Syntax

RECEIVE AND PROCESS
CLEARFILE(*clear specified file contents*)
ENVTYPE(*transaction envelope type*)
FILEID(*logical name of the source data*)
IFCC(*override condition codes*)
MSGUCLASS(*override message user class*)
ONEMSG(*read only one MQ message*)
REQID(*requestor ID*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RECEIVE AND PROCESS command example

XML purchase orders are routed from the purchasing system to WebSphere Data Interchange by WebSphere MQ Integrator on a WebSphere MQ queue named **XML_IN**. WebSphere Data Interchange job is to transform them into EDI and send to the appropriate trading partner. The customer has elected to write their own trigger program instead of using the WebSphere Data Interchange Adapter for WebSphere MQ since they want more control over the triggering process. A service profile called **SNDXMLPO** is created in the WebSphere Data Interchange client tool. Whenever a message arrives on the **XML_IN** queue, the trigger program invokes WebSphere Data Interchange with the following command:

```
PERFORM RECEIVE AND PROCESS
    WHERE REQID(XML_IN) FILEID(SNDXMLPO) CLEARFILE(Y)
```

RECEIVE AND SEND command

This command is valid only with MQSeries queues and combines the functions of the RECEIVE and SEND commands. Data can be received from an MQSeries queue, the data translated, and the translated output placed in a destination MQSeries queue.

Syntax

RECEIVE AND SEND
APPFILE(*application data file name*)
CLEARFILE(*clear specified file contents*)
ENVTYPE(*transaction envelope type*)
FILEID(*processing file ddname*)
IFCC(*override condition codes*)
MSGUCLASS(*override message user class*)
OUTFILE(*output data file name for SAP Status*)
OUTTYPE(*output data file type for SAP Status*)
RAWDATA(*translate to raw data format*)
REQID(requestor ID)
SAPUPDT(*track SAP status*)
SCRIPT(*script ID*)
SEQNUM(*increment network profile member numbers*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RECEIVE AND SEND command example

Receive raw data from MQSeries queue **MQTESTREQ1**. Place the data into the receive file **MQSM**.

```
PERFORM RECEIVE AND SEND
    WHERE REQID(MQTESTREQ1) RAWDATA(Y) APPFILE(MQSM)
```

RECEIVE AND TRANSLATE command

This command combines the functions of the RECEIVE, DEENVELOPE, and TRANSLATE TO APPLICATION commands. This composite command performs faster than the separate commands.

Note: In CICS, you can also deliver the data to a program or CICS transaction, as specified in the `Application file name` and `Application file type` fields of the data format.

Syntax

RECEIVE AND TRANSLATE
ASSERTLVL(*session assertion level*)
BATCHSET(*set transaction batch ID*)
CCEXCEPTION(*job-step condition code*)
CLEARFILE(*clear specified file contents*)
DIERRFILTER(*initial error filter set*)
DUPENV(*process duplicate envelopes*)
ENVTYPE(*transaction envelope type*)
EXTENDC(*translate with extended C record format*)
FADELAY(*delay queuing functional acknowledgment*)
FILEID(*processing file ddname*)
FORCETEST(*force test usage*)
FUNACKFILE(*functional acknowledgment ddname*)
FUNACKREQ(*require functional acknowledgment envelope file*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
MSGUCLASS(*override message user class*)
ONEMSG(*read only one MQ message*)
OPTRECS(*optional record type*)
OUTFILE(*output data file name for SAP Status*)
OUTTYPE(*output data file type for SAP Status*)
PAGE(*pageable translation*)
PURGINT(*purge interval*)
RAWDATA(*translate to raw data format*)
REQID(*requestor ID*)
SAPUPDT(*track SAP status*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RECEIVE AND TRANSLATE command examples

Example 1:

Receive and translate all EDI documents for requestor **DEPTA7F**. Along with the transaction data, return all optional records produced during deenveloping and translation. The documents are in X12 format.

```
PERFORM RECEIVE AND TRANSLATE
    WHERE REQID(DEPTA7F) OPTRECS(IEGTQ)
```

Example 2:

Receive and translate all EDI documents for requestor **DEPTA7F**. Return the transaction data in raw data format. The documents are in EDIFACT format.

```
PERFORM RECEIVE AND TRANSLATE
    WHERE REQID(DEPTA7F) RAWDATA(Y) ENVTYPE(E)
```

RECONSTRUCT command

This command takes information that has been saved in the Transaction Store and rebuilds an interchange just as it was sent or received (using the same control numbers). This command can be used if your trading partner lost an interchange you sent, and you must send the same interchange again. It can also be used to rebuild interchanges sent to you.

Syntax

RECONSTRUCT
DIR(*processing direction*)
FILEID(*processing file ddname*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
PAGE(*pageable translation*)
RAWDATA(*translate to raw data format*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RECONSTRUCT command example

You received and processed interchange control number **5** from your trading partner **MYTP**, but have since destroyed the sequential file containing the interchange. You would like to reconstruct this file so you can save it for the auditors. The command statement will reconstruct the interchange for trading partner **MYTP** with interchange receiver ID **123456789**, interchange control number **5**, and file name **AUDITOR**.

```
PERFORM RECONSTRUCT
    WHERE TPNICKN(MYTP) INTRECID(123456789) INTCTLNO(5) DIR(R)
    FILEID(AUDITOR)
```

RECONSTRUCT AND SEND command

This command combines the functions of the RECONSTRUCT and SEND commands. It rebuilds interchanges from data in the Transaction Store, and then sends those interchanges to the network. This composite command performs faster than the separate commands.

Syntax

RECONSTRUCT AND SEND
CLEARFILE(*clear specified file contents*)
DIR(*processing direction*)
ENVTYPE(*transaction envelope type*)
FILEID(*processing file ddname*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
MSGUCLASS(*override message user class*)
PAGE(*pageable translation*)
RAWDATA(*translate to raw data format*)
REQID(*requestor ID*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RECONSTRUCT AND SEND command example

Your trading partner **MYTP** called and indicated that interchange control number **5** is missing, and that they would like you to resend that interchange. The trading partner's DUNS number **123456789**is used as the interchange receiver ID value. The command statement will reconstruct the interchange for trading partner **MYTP** with interchange receiver ID **123456789** and interchange control number **5**.

```
PERFORM RECONSTRUCT AND SEND
    WHERE TPNICKN(MYTP) INTRECID(123456789) INTCTLNO(5) DIR(S)
    REQID(MYREQ)
```

RECVFILE command

This command receives application data files that do not need an interchange header or that do not contain EDI data. The data is not necessarily in any EDI standard format.

Syntax

RECVFILE
CLEARFILE(*clear specified file contents*)
FILEID(*processing file ddname*)
IFCC(*override condition codes*)
MSGUCLASS(*override message user class*)
REQID(*requestor ID*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RECVFILE command example

Receive data for requestors defined in mailbox (requestor) profile member **DEPTA7F**. Place the data into file **RCVFILE**. Clear **RCVFILE** before the data is received.

```
PERFORM RECVFILE
    WHERE REQID(DEPTA7F) FILEID(RCVFILE) CLEARFILE(Y)
```

RECVFILE AND SEND command

This command is valid only with MQSeries queues and combines the functions of the RECVFILE and SEND commands. Application data can be received from an MQSeries queue, the data translated, and the translated output placed in a destination MQSeries queue. The data is not necessarily in any EDI standard format.

Syntax

RECVFILE AND SEND
CLEARFILE(*clear specified file contents*)
ENVTYPE(*transaction envelope type*)
FILEID(*processing file ddname*)
IFCC(*override condition codes*)
MSGUCLASS(*override message user class*)
OUTFILE(*output data file name for SAP Status*)
OUTTYPE(*output data file type for SAP Status*)
REQID(*requestor ID*)
SAPUPDT(*track SAP status*)
SCRIPT(*script ID*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RECVFILE AND SEND command example

Receive raw data from MQSeries queue **MQTESTREQ1**. Place the data into the receive file **QDATA**.

```
PERFORM RECVFILE AND SEND
    WHERE REQID(MQTESTREQ1) FILEID(QDATA) RAWDATA(Y)
```

REENVELOPE command

This command takes the EDI transactions from the Transaction Store that were previously enveloped, envelopes them again, and places the results in an envelope file. The envelope file is either the TD queue specified in the network profile member or a file specified in the command. Selection criteria determine which transactions go into the interchange envelope. The enveloper sorts the transactions to create the fewest number of functional groups and interchange envelopes. For more information about envelope processing, see "Enveloping services" on page 500.

Syntax

REENVELOPE
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIERRFILTER(*initial error filter set*)
ENVDATE(*starting transaction envelope date*)
      TO(*ending transaction envelope date*)
ENVPRBREAK(*start new envelope*)
ENVTIME(*starting transaction envelope time*)
      TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
FILEID(*processing file ddname*)
FIXEDFILEID(*fixed-to-fixed output ddname*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
      TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IACCESS(*IEXIT access*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
INMEMTRANS(*transactions in memory*)
INTCTLNO(*starting sender's interchange control nbr*)
      TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
ITPBREAK(*new interchange envelope*)
ITYPE(*IEXIT program type*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
OPTRECS(*optional record type*)
OUTFILE(*output data file name for SAP Status*)

OUTTYPE(*output data file type for SAP Status*)
PAGE(*pageable translation*)
RAWDATA(*translate to raw data format*)
RECOVERY(*recovery unit of work*)
SAPUPDT(*track SAP status*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
STDTRID(*EDI standard transaction set ID*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
        TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)
VERIFY(*verify transaction status*)

REENVELOPE command example

Reenvelope the EDI document with the Transaction Store handle
20011214101533000001 and a status of **31**. Place the results in the network TD queue.

```
PERFORM REENVELOPE
    WHERE HANDLE(20011214101533000001) TRXSTAT(31)
```

REENVELOPE AND SEND command

This command provides the same functions as the ENVELOPE AND SEND command but for EDI data that was previously enveloped. You must supply the requestor ID of each network for which data is reenveloped. This composite command performs faster than the separate commands.

Syntax

REENVELOPE AND SEND
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
CLEARFILE(*clear specified file contents*)
DIERRFILTER(*initial error filter set*)
ENVDATE(*starting transaction envelope date*)
      TO(*ending transaction envelope date*)
ENVPRBREAK(*start new envelope*)
ENVTIME(*starting transaction envelope time*)
      TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
FILEID(*processing file ddname*)
FIXEDFILEID(*fixed-to-fixed output ddname*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
      TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IACCESS(*IEXIT access*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
INMEMTRANS(*transactions in memory*)
INTCTLNO(*starting sender's interchange control nbr*)
      TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
ITPBREAK(*new interchange envelope*)
ITYPE(*IEXIT program type*)
MSGUCLASS(*override message user class*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
OPTRECS(*optional record type*)
OUTFILE(*output data file name for SAP Status*)
OUTTYPE(*output data file type for SAP Status*)

REENVELOPE AND SEND

PAGE(*pageable translation*)
RAWDATA(*translate to raw data format*)
RECOVERY(*recovery unit of work*)
REQID(*requestor ID*)
SAPUPDT(*track SAP status*)
SCRIPT(*script ID*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
STDTRID(*EDI standard transaction set ID*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TPNICKNESEND(*trading partner profile member*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
        TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)
VERIFY(*verify transaction status*)

REENVELOPE AND SEND command example

Reenvelope the EDI document with transaction handle **20011212101533000001** and a status of **31**. Place the results in the network queue and send the data.

```
PERFORM REENVELOPE AND SEND
    WHERE HANDLE(20011212101533000001) TRXSTAT(31) REQID(MYREQ)
```

RELEASE command

This command restores a transaction in held status to its former status (or to PURGE-PENDING status if its store time expired during the hold period). If the transaction is one of a related group, all transactions in the group are released. WebSphere Data Interchange never automatically sets a transaction's store status to HELD; you must use the HOLD command.

The HOLD, PURGE, RELEASE, and UNPURGE commands share a common syntax.

Syntax

RELEASE
ACFIELD(*starting application control field data*)
　　　　TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
　　　　TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
　　　　TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
　　　　TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
　　　　TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
　　　　TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)

RELEASE

                TO(*ending transaction set control number*)
        TRXDATE(*starting transaction date*) TO(*ending transaction date*)
        TRXSTAT(*transaction processing status*)
        TRXTIME(*starting transaction time*) TO(*ending transaction time*)

RELEASE command example

Release all EDI documents destined for trading partner **PISCES** that are in
HELD status.

```
PERFORM RELEASE
    WHERE TPNICKN(PISCES) DIR(S) STSTAT(1)
```

REMOVE LOG ENTRIES command

This command is valid only for DB2 event logs and can be a very quick way to remove entries. You may run this command as the first step of a multi-step maintenance process.

If the `Archivefile` keyword is used, this command will work the same as the UNLOAD LOG ENTRIES command by copying selected entries to the archive file.

If the `Numdels` keyword is used, rows are deleted sequentially with COMMITs performed each time the number of rows specified in `Numdels` is reached. If the `Numdels` keyword is omitted, a single fully-qualified SQL DELETE statement is issued. (This can be very efficient, but can also quickly exhaust DB2 resources. Omit `Numdels` with caution.) To ensure concurrency, set the `NUMDELS keyword` to a relatively low value.

Syntax

REMOVE LOG ENTRIES
APPLID(*application ID*)
ARCHIVEFILE(*event log archive file name*)
ARCHIVETYPE(*event log archive file type*)
HOLDFILE(*event log hold file name*)
HOLDTYPE(*event log hold file type*)
IFCC(*override condition codes*)
LOGAEID(*starting event log associated entry ID*)
       TO(*ending event log associated entry ID*)
LOGDATE(*starting event log date*) TO(*ending event log date*)
LOGFORM(*starting event log format ID*) TO(*ending event log format ID*)
LOGTIME(*starting event log time*) TO(*ending event log time*)
LOGUSER(*starting event log user ID*) TO(*ending event log user ID*)
NUMDELS(*number of database deletes before commit*)
SETCC(*condition codes*)

REMOVE LOG ENTRIES command example

Remove log entries from application log file for application EDIFFS dated December 14, 2001.

```
PERFORM REMOVE LOG ENTRIES
    WHERE APPLID(EDIFFS) LOGDATE(12/14/01)
```

REMOVE STATISTICS command

This command deletes outdated management reporting statistics. If you do not specify a date, tomorrow's date is used as the selection criteria. To reset the cumulative records to zero, use the RESET STATISTICS command.

Syntax

REMOVE STATISTICS
IFCC(*override condition codes*)
NUMDELS(*number of database deletes before commit*)
PRIORTO(*deletion end date*)
SETCC(*condition codes*)

REMOVE STATISTICS command example

Remove any statistics over six months old. In addition, suppose you use the DB2 version of WebSphere Data Interchange and previously exceeded the maximum number of page locks when issuing a REMOVE STATISTICS command. To get rid of statistics over six months old, use the PRIORTO keyword with a date of six months ago. To reduce the accumulation of page locks by the REMOVE STATISTICS process, you can set the Numdels parameter to a low number, such as 25. This causes the REMOVE STATISTICS command to commit work and release page locks after every 25 deletes.

```
PERFORM REMOVE STATISTICS
    WHERE PRIORTO(*-180) NUMDELS(25)
```

Notes:

1. Running this command does not remove cumulative records.
2. To reset the Management Reporting cumulative statistics counts to zero, use the RESET STATISTICS command.

REMOVE TRANSACTIONS command

This command deletes transactions from the Transaction Store that have a status of either:
* PURGE-STORE TIME EXPIRED
* PURGE-USER REQUEST

You can protect transactions you want to keep with the HOLD or UNPURGE command. UNPURGE, however, protects only transactions with a status of PURGE-USER REQUEST. It does not protect those with a status of PURGE-STORE TIME EXPIRED. Once deleted, transactions are not recoverable.

The default expiration date is 30 days from translation. You can change the default by specifying a PURGINT value on PERFORM statements which add transactions to the Transaction Store (for example, TRANSLATE AND ENVELOPE). See "PURGINT" on page 167 for applicable PERFORM commands.

Note: Running the REMOVE TRANSACTIONS command with selection criteria specified in the WHERE clause may remove other entries from the EDIVTSTH table that do not match the selection criteria. These additional entries reported as being removed in the audit trail. These entries are removed because they have no direction assigned to them and are over 10 days old. "Orphan" entries like these are created when the first transaction of a translation process does not complete normally. These entries cannot be accessed in any way and are only shown when removed. They are removed whenever the REMOVE TRANSACTIONS command is executed.

The WHERE clause is optional for this command. If you do not specify a WHERE clause, all eligible transactions are removed from the Transaction Store.

After purging documents from the Transaction Store, you can use the Archive action to remove the event logs associated with these transactions. For more information, refer to the WebSphere Data Interchange for z/OS Administration Guide.

This command does not delete ACTIVE or HELD transactions.
* To make ACTIVE transactions eligible for removal, they must first be marked for purge.
* To make HELD transactions eligible for removal, they must first be released from held status.

Syntax

REMOVE TRANSACTIONS
ACFIELD(*starting application control field data*)
       TO(ending application control field data)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)

DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
            TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
            TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
            TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
            TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
            TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
MAXRUNTIME(*maximum remove runtime minutes*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(network transaction status)
NUMDELS(*number of database deletes before commit*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STANDALONE(*operate DataInterchange only*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
            TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

REMOVE TRANSACTIONS command example

Delete all eligible transactions that are more than 30 days old. Only transactions that
have a status of PURGE-USER REQUEST or PURGE-DATE EXPIRED are eligible.

```
PERFORM REMOVE TRANSACTIONS
    WHERE HANDLE(*-999) TO(*-30)
```

REPORT CONTINUOUS RECEIVE STATUS command

This command generates a continuous receive status report about the status of either a single continuous receive member or all continuous receives known to WebSphere Data Interchange and Expedite/CICS. The output goes to the report file. The report file name and type can be specified in the Utility Control Information block. The default file name is **RPTFILE** and the default file type in CICS is **TS**.

Syntax

REPORT CONTINUOUS RECEIVE STATUS
IFCC(*override condition codes*)
MEMBER(*member profile name*)
SETCC(*condition codes*)

REPORT CONTINUOUS RECEIVE STATUS command examples

Example 1:

Report all continuous receives.

```
PERFORM REPORT CONTINUOUS RECEIVE STATUS
```

Example 2:

Report the status of continuous receive profile member CRMEM.

```
PERFORM REPORT CONTINUOUS RECEIVE STATUS
    WHERE MEMBER(CRMEM)
```

RESET STATISTICS command

This command resets the cumulative records to zero. Run this command after running the REMOVE STATISTICS command to restart the statistical history.

Syntax

RESET STATISTICS
IFCC(*override condition codes*)
NUMDELS(*number of database deletes before commit*)
PRIORTO(*deletion end date*)
SETCC(*condition codes*)

RESET STATISTICS command example

Reset the cumulative record values recorded for the last 180 days to zero.

```
PERFORM RESET STATISTICS
    WHERE PRIORTO(*-180)
```

RESTART RECEIVE command

This command is used when receiving EDI data, and is valid only when both the following are true:

- The network supports restart
- You specify checkpoint-level recovery when initially receiving the data

If an error causes network processing to enter a restart situation while processing EDI data, this command can be used to restart and complete the receive. For more information on checkpoint recovery, refer to the Expedite Base/MVS Programming Guide.

Notes:

1. The requestor ID must be the same value specified on the initial receive.
2. This command is not supported in the CICS environment.

Syntax

RESTART RECEIVE
REQID(*requestor ID*)

RESTART RECEIVE command example

Restart the receive of data from the network defined in mailbox (requestor) profile **DEPTA7F**.

```
PERFORM RESTART RECEIVE
    WHERE REQID(DEPTA7F)
```

RESTART SEND command

This command is used when sending EDI data, and is valid only when both the following are true:

- The network supports restart
- You specified checkpoint-level recovery when initially sending the data

If an error causes network processing to enter a restart situation while processing EDI data, this command can be used to restart and complete the send. For more information on checkpoint recovery, refer to the Expedite Base/MVS Programming Guide.

Notes:

1. The envelope file and the requestor ID must be the same values specified on the initial send.
2. This command is not supported in the CICS environment.

Syntax

RESTART SEND
CLEARFILE(*clear specified file contents*)
ENVTYPE(*transaction envelope type*)
FILEID(*processing file ddname*)
IFCC(*override condition codes*)
MSGUCLASS(*override message user class*)
REQID(*requestor ID*)
SAPUPDT(*track SAP status*)
SCRIPT(*script ID*)
SEQNUM(*increment network profile member numbers*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

RESTART SEND command example

Restart the send of data from file **NETQUEUE** to the network defined in mailbox (requestor) profile **IINREQ**.

```
PERFORM RESTART SEND
    WHERE REQID(IINREQ) FILEID(NETQUEUE)
```

RETRANSLATE TO APPLICATION command

This command provides the same functions as the TRANSLATE TO APPLICATION command for transactions that were previously translated. It takes EDI documents from the Transaction Store, translates them to application format, and places the results in the file specified by the data format or in the override file specified by the trading partner usage/rule for translating the transaction. The data can be formatted as C and D records or as raw data.

Syntax

RETRANSLATE TO APPLICATION
ACFIELD(*starting application control field data*)
      TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
ASSERTLVL(*session assertion level*)
BATCH(*translated transaction batch ID*)
BATCHSET(*set transaction batch ID*)
CCEXCEPTION(*job-step condition code*)
DIERRFILTER(*initial error filter set*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
      TO(*ending transaction purge date*)
EXTENDC(*translate with extended C record format*)
FORCETEST(*force test usage*)
FORMAT(*data format ID*)
GRPCTLNO(*starting sender's group control number*)
      TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
      TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
NETID(*network ID*)
OPTRECS(*optional record type*)
PAGE(*pageable translation*)
RAWDATA(*translate to raw data format*)
SETCC(*condition codes*)
STDTRID(*EDI standard transaction set ID*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
      TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)

RETRANSLATE TO APPLICATION

TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

RETRANSLATE TO APPLICATION command example

Re-translate all EDI documents in the Transaction Store that were received on December 14, 2001 from trading partner **PISCES**. Place the data in the application file specified by the data format or in the override file specified in the receive usage/rule for the trading partner.

```
PERFORM RETRANSLATE TO APPLICATION
    WHERE TRXDATE(01/12/14) TPNICKN(PISCES)
```

SAP STATUS EXTRACT command

This command allows you to extract SAP status information from the Transaction Store.

Syntax

SAP STATUS EXTRACT
EXTRACTED(*previously extracted status*) Default is N.
OUTFILE(*output file name*)
OUTTYPE(*output file type*)
CLIENT(*SAP client ID*)
SAPSTAT(*SAP status starting value*) TO(*SAP status ending value*)
DAYS(*record starting date*) TO(*record ending date*)

SAP STATUS EXTRACT command example

Extract SAP status records from the WebSphere Data Interchange database and write them to the file specified in the keywords OUTFILE and OUTTYPE.

```
PERFORM SAP STATUS EXTRACT
    WHERE OUTFILE() OUTTYPE() CLIENT() SAPSTAT() TO() DAYS() TO()
```

SAP STATUS REMOVE command

This command allows you to remove SAP status information from the Transaction Store.

Syntax

SAP STATUS REMOVE
CLIENT(*SAP client ID*)
EXTRACTED(*previously extracted status*) Default is Y.
PRIORTO(*Date before which to remove records*)
SAPSTAT(*SAP status starting value*) TO(*SAP status ending value*)
DAYS(*Record starting date*) TO(*Record ending date*)

SAP STATUS REMOVE command example

Remove SAP status information from the WebSphere Data Interchange database for all clients with status **16**.

```
PERFORM SAP STATUS REMOVE
WHERE SAPSTAT(16) TO(19)
```

SEND command

This command sends EDI transactions from an envelope file to the network. The envelope file is either the TD queue specified in the network profile member or a file specified in the command.

Syntax

SEND
CLEARFILE(*clear specified file contents*)
ENVTYPE(*transaction envelope type*)
FILEID(*processing file ddname*)
IFCC(*override condition codes*)
MSGUCLASS(*override message user class*)
ONEMSG(*read only one MQ message*)
REQID(*requestor ID*)
SAPUPDT(*track SAP status*)
SCRIPT(*script ID*)
SEQNUM(*increment network profile member numbers*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

SEND command examples

Example 1:

Send the transactions in the TD queue. Use the network specified by mailbox (requestor) profile member **IINREQ**.

```
PERFORM SEND
    WHERE REQID(IINREQ)
```

Example 2:

Send the EDIFACT transactions in the file **NETQUEUE**. Use the network specified by mailbox (requestor) profile member **IINREQ**.

```
PERFORM SEND
    WHERE REQID(IINREQ) FILEID(NETQUEUE) ENVTYPE(E)
```

SENDFILE command

This command sends application data files that do not need an interchange header or that do not contain EDI data.

Syntax

SENDFILE
CLEARFILE(*clear specified file contents*)
FILEID(*processing file ddname*)
IFCC(*override condition codes*)
MSGUCLASS(*override message user class*)
ONEMSG(*read only one MQ message*)
RAWDATA(*translate to raw data format*)
REQID(*requestor ID*)
SCRIPT(*script ID*)
SEQNUM(*increment network profile member numbers*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)

Mapping issues with RAWDATA keyword on SENDFILE

The use of the **RAWDATA** keyword on the **SENDFILE** command will cause WebSphere Data Interchange to strip C and D records from an existing DF file generated in a previous step. When WebSphere Data Interchange generates a DF file in raw-data format the structure ID is placed in the first column of the data. This is not done with C and D record data. You must map the structure ID as the first element in the data if you wish to see the structure name. This can be done using the mapping tool in the WebSphere Data Interchange client and allows you to place the structure ID anywhere in the data.

SENDFILE command examples

Example 1:

Send the data in the file named **FLATFILE** to trading partner **MYTP**.

```
PERFORM SENDFILE
    WHERE REQID(ME) FILEID(FLATFILE) TPNICKN(MYTP)
```

Example 2:

Send the data in the file named **BULK** to trading partner **YOURTP** with a network message user class of **SPECIAL**.

```
PERFORM SENDFILE
    WHERE REQID(ME) FILEID(BULK) TPNICKN(YOURTP) MSGUCLASS(SPECIAL)
```

START CONTINUOUS RECEIVE command

This command is available only in the CICS environment. Like the CICS transaction EDIR, this command can be used to start a single continuous receive or start all eligible continuous receives. A continuous receive can be started if:

- A valid requestor ID is specified in the continuous receive profile member.
- The continuous receive profile member is marked **ACTIVE**.
- The network ID in the associated mailbox (requestor) profile member is **IINCICS**.

Syntax

START CONTINUOUS RECEIVE
IFCC(*override condition codes*)
MEMBER(*member profile name*)
SETCC(*condition codes*)

START CONTINUOUS RECEIVE command examples

Example 1:
Start all eligible continuous receives.

```
PERFORM START CONTINUOUS RECEIVE
```

Example 2:
Start a continuous receive for profile member **CRMEM**.

```
PERFORM START CONTINUOUS RECEIVE
    WHERE MEMBER(CRMEM)
```

STOP CONTINUOUS RECEIVE command

This command is available only in the CICS environment. Like the CICS transaction EDIS, this command can be used to stop a single continuous receive or stop all eligible continuous receives. A continuous receive can be stopped if:

- A valid requestor ID is specified in the continuous receive profile member;
- The continuous receive profile member is marked **ACTIVE**; and
- The network ID in the associated mailbox (requestor) profile member is **IINCICS**.

Syntax

STOP CONTINUOUS RECEIVE
IFCC(*override condition codes*)
MEMBER(*member profile name*)
SETCC(*condition codes*)

STOP CONTINUOUS RECEIVE command examples

Example 1:
Stop all active continuous receives.

```
PERFORM STOP CONTINUOUS RECEIVE
```

Example 2:
Stop the continuous receive for profile member **CRMEM**.

```
PERFORM STOP CONTINUOUS RECEIVE
    WHERE MEMBER(CRMEM)
```

TRADING PARTNER CAPABILITY DATA EXTRACT command

This command gathers data about the transaction capabilities of trading partners including:

- Which transactions have maps, sorted by trading partner
- The total number of transactions translated against these maps
- The total number of transactions with unacceptable error levels translated against these maps

With a report writer or custom program, you can use this command to create reports such as:

- Pre-Migration Status
- Trading Partner Implementation Status
- Trading Partner Testing Status
- Cumulative Transaction Activity by Trading Partner
- Where Used Maps by Trading Partner
- Trading Partners by Maps Used

> **Note:** If your customer tables are not in the same database as your runtime tables, all data fields may not be included in this report.

Syntax

TRADING PARTNER CAPABILITY DATA EXTRACT
ADDRLN1(*trading partner address line 1*)
ADDRLN2(*trading partner address line 2*)
CMMTLN1(*trading partner comment line 1*)
CMMTLN2(*trading partner comment line 2*)
CMPYNM(trading partner company name)
DAYS(*starting date*) TO(*ending date*)
DIR(*processing direction*)
DYNSQL(*use dynamic SQL*)
IFCC(*override condition codes*)
MAPID(*map name*)
SETCC(*condition codes*)
STDDESC(*EDI standard description*)
STDID(*EDI standard ID*)
STDLV(*EDI standard release level*)
STDTRID(*EDI standard transaction set ID*)
STDVR(*EDI standard version*)
TESTMODE(*test transaction mode*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
USERPGM(*user program*)

TRADING PARTNER CAPABILITY DATA EXTRACT command examples

Example 1:

Company X is implementing EDI with all suppliers to their Atlanta assembly plant. They have set a target date of 12/31 to have 100% EDI on several transactions. Suppliers start out in test mode for each new transaction and, after 100 successful test transactions, are eligible to move into production. The EDI steering committee meets monthly to review progress and needs a report that shows information for each supplier, such as:

- All transactions sent and received
- When the supplier went into test for each transaction type
- Total number of transactions sent or received in test mode
- When the supplier went into production for each transaction type
- Total number of transactions sent or received in production mode

```
PERFORM TRADING PARTNER CAPABILITY DATA EXTRACT
    WHERE CMMTLN1(Atlanta) CMMTLN2(Supplier)
```

Figure 11 shows an example of a trading partner capability report.

```
                  Trading Partner Capability Report Data

Company    Location    Int ID Trx Std. ID Dir     ID        Started Test   Started
 NameID                                                      Testing Trx's  Productn
---------- ----------- ------ --- ------- --- ---------------- -------- ------ --------
Global     Tampa, FL   233119 850 X12V2R1  S  NONPRODX12V2R1PO 07/03/01 101    08/15/01
Services

Only1 Co.  Lima, OH    399129 850 X12V2R3  S  PRODSUPX12V2R3PO 08/15/01 71     *NODATE*

Pulse      Armonk, NY  997223 850 X12V2R3  S  PRODSUPX12V2R3PO *NODATE*  0     *NODATE*
Services

Handlebars Raleigh, NC 877519 850 X12V2R3  S  PRODSUPX12V2R3PO 05/07/01  115   07/02/01
Etc.                          855 X12V2R3  R  PRODSUPX12V2R3PA 05/22/01  109   07/26/01
                              856 X12V2R3  R  PRODSUPX12V2R3SN 06/13/01  187   07/22/01
                              861 X12V2R3  S  PRODSUPX12V2R3RA 08/08/01  242   *NODATE*

IBM        Dallas, TX  660599 850 X12V2R3  S  PRODSUPX12V2R3PO 01/08/01  126   03/14/01
                              855 X12V2R3  R  PRODSUPX12V2R3PA 08/22/01  054   *NODATE*
                              856 X12V2R3  R  PRODSUPX12V2R3SN 04/06/01  378   *NODATE*

Pen&Paper  Carson, CA  523567 850 X12V2R3  S  PRODSUPX12V2R3PO 02/02/01  113   03/02/01
                              855 X12V2R3  R  PRODSUPX12V2R3PA 03/26/01  104   03/12/01
                              856 X12V2R3  R  PRODSUPX12V2R3SN 03/13/01  127   11/23/01
                              861 X12V2R3  S  PRODSUPX12V2R3RA 04/08/01  102   06/08/01
```

Figure 11. Example of a trading partner capability report

Example 2:

You want to extract information associated with migrating the purchase order transaction from X12V2R1 to X12V3R1. Create a list of all the trading partners and usages or rules you must migrate to the new EDI standard.

```
PERFORM TRADING PARTNER CAPABILITY DATA EXTRACT
    WHERE STDID(X12V2R1) STDTRID(850)
```

TRADING PARTNER PROFILE DATA EXTRACT command

This command provides detailed information about your trading partners including:
- Company name
- Address
- Contacts
- Telephone number
- Nickname
- Network name
- Interchange ID
- Account ID
- User ID
- Latest data transmission date, sorted by trading partner

Before running this command, you must run the UPDATE STATISTICS command to gather the data.

With a report writer or custom program, you can use this command to create reports such as:
- Simple trading partner lists
- Comprehensive trading partner lists
- Trading partners sorted by division
- Inactive trading partners (no activity for *x* period of time)

Syntax

TRADING PARTNER PROFILE DATA EXTRACT
ACCTID(*network account ID*)
ADDRLN1(*trading partner address line 1*)
ADDRLN2(*trading partner address line 2*)
CMMTLN1(*trading partner comment line 1*)
CMMTLN2(*trading partner comment line 2*)
CMPYNM(trading partner company name)
CNTCTNM(*trading partner contact name*)
CNTCTPH(*trading partner contact phone number*)
DYNSQL(*use dynamic SQL*)
GRPCTLNO(*starting sender's group control nbr.*)
       TO(*ending sender's group control nbr.*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
       TO(*ending sender's interchange control nbr*)
INTID(*trading partner interchange ID*)
LASTTRXDATE(*starting latest transaction date*)
       TO(*ending latest transaction date*)
NETID(*network ID*)
SETCC(*condition codes*)
TPNICKN(*trading partner nickname*)
TRXCTLNO(*starting transaction set control number*)
       TO(*ending transaction set control number*)
USERID(*network user ID*)
USERPGM(*user program*)

TRADING PARTNER PROFILE DATA EXTRACT command examples

Example 1:

Your EDI Administrator has requested that you provide a hard copy report every Monday morning that shows the trading partners who have sent data within the last six months to the Metal Finishing and Metal Shipping Divisions. This report is used to interface with the various departments and to contact the trading partner in case of trouble. The report contains:

- Company name and location
- Supplier/customer Dun and Bradstreet (DUNS) number
- EDI contact's name and phone number
- Date of the last transaction sent to them
- Control numbers of the last transaction sent to them

```
PERFORM TRADING PARTNER PROFILE DATA EXTRACT
    WHERE CMMTLN2(Metal Finishing) LASTTRXDATE(*-180) TO(*)
    WHERE CMMTLN2(Metal Shipping) LASTTRXDATE(*-180) TO(*)
```

Or, if these are the only two divisions beginning with the word Metal, you can use this command:

```
PERFORM TRADING PARTNER PROFILE DATA EXTRACT
    WHERE CMMTLN2(Metal*) LASTTRXDATE(*-180) TO(*)
```

Figure 12 shows an example of a trading partner profile report.

```
                    Trading Partner Information Report Data


Company Location   IntID  Net Accnt  UserID    Contact       Contact      LastTrx
  Name                                         Name          Phone        Date
--------------------------------------------------------------------------------
A0 Corp Morris,MN  333219 NETA MMIDOO 55532214  Sam Heusen    813-555-5668 08/01/01
IBM     Tampa, FL  200762 NETA ADV001 99665593  Deb Garrie    762-555-2328 08/15/01
Alloy   Chicago,IL 543189 NETA ALUM00 18654771  Lou Green     619-555-7784 08/15/00
Inc.
DeKant  Queens, NY 337690 IIN  DECANT USER01    Bob Kling     252-555-9006 11/06/01
Systems London, UK 856690 IIN  DECAHH USER02    Joe Miller    413-555-5555 03/17/01
        London, UK 009210 IIN  DECA00 USER03    Heather Liu   413-555-2256 03/27/01


IBM     Paris   FR 665767 IIN  IBMIM  01NYC     Mary Lockler 919-555-8585 02/22/01
        Raleigh,NC 976554 IIN  IB     SANJOSE   Jane Hart     878-555-2876 02/06/01


MMM     Edison, NJ 332323 NETA MMM01  MMADMIN   Jon Murphy    828-555-9058 03/18/01
Motors  Miami,  FL 003292 IIN  MMM02  MMTECH    Pauline Gold 813-555-8421 01/22/01
```

Figure 12. Example of a trading partner profile report

Example 2:

Generate a data extract of all the trading partners for your **ABC** and **DEF** divisions with whom you have not exchanged any transactions within the last six months. Print only information about trading partners on the network. Next, put the division in the `Comment Line 2` field of the trading partner profile.

```
PERFORM TRADING PARTNER PROFILE DATA EXTRACT
    WHERE CMMTLN2(ABC) LASTTRXDATE(*-999) TO(*-180) NETID(IN*)
    WHERE CMMTLN2(DEF) LASTTRXDATE(*-999) TO(*-180) NETID(IN*)
```

TRANSACTION ACTIVITY DATA EXTRACT command

This command collects data about the daily transaction activity of trading partners including:

- Total number of transactions sent or received on any day or range of days, sorted by ID
- Total number of transactions with errors that were sent or received on any day or range of days, sorted by ID

You can use this command to create reports such as:
- Outbound Document Audit Summary (overview)
- Inbound Document Audit Summary (detailed
- Transaction Activity by Transaction
- Transaction Activity by Trading Partner
- Outbound/Inbound Errors
- Activity Summary by Trading Partner or by Transaction

Syntax

TRANSACTION ACTIVITY DATA EXTRACT
ADDRLN1(*trading partner address line 1*)
ADDRLN2(*trading partner address line 2*)
CMMTLN1(*trading partner comment line 1*)
CMMTLN2(*trading partner comment line 2*)
CMPYNM(trading partner company name)
DAYS(*starting date*) TO(*ending date*)
DIR(*processing direction*)
DYNSQL(*use dynamic SQL*)
IFCC(*override condition codes*)
MAPID(*map name*)
SETCC(*condition codes*)
STDDESC(*EDI standard description*)
STDID(*EDI standard ID*)
STDLV(*EDI standard release level*)
STDTRID(*EDI standard transaction set ID*)
STDVR(*EDI standard version*)
TESTMODE(*test transaction mode*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
USERPGM(*user program*)

TRANSACTION ACTIVITY DATA EXTRACT command example

Extract a list of all your trading partners and the number of invoices they sent you through EDI in the first six months of 2001. This report can be used to spot-check the accounts receivable system to verify that the invoice receipt process is in control. You receive both EDIFACT and X12 invoices. The transaction activity report provides the necessary data listed by trading partner.

```
PERFORM TRANSACTION ACTIVITY DATA EXTRACT
    WHERE STDTRID(INVOIC) DAYS(01/01/01) TO(01/06/30) DIR(R)
    WHERE STDTRID(810)    DAYS(01/01/01) TO(01/06/30) DIR(R)
```

TRANSACTION ACTIVITY DATA EXTRACT

Figure 13 shows an example of a transaction activity data extract report, after the data has been sorted and formatted.

```
           EDI Invoice Activity Summary for 1H01

Compamy                                        Trx   Errors  Trx   Errors
Name         Location    Int ID  Trx ID  Std. ID   Dir  MTD   MTD     YTD   YTD
---------    ----------  ------  ------  --------  ---   ---   ---     ---   ------
IBM Global Tampa, FL   933489   810     X12V2R1   R     14    1       136   9
Services

CULATER    Dallas, TX  322217   810     X12V2R3   R     49    5       450   8
                                810     X12V2R3   R     10    5        90   14
                                810     X12V2R3   R    250   23      2200   28
                                INVOIC  EDIL921   R     24    0       224   0

Definite   Paris, FR   005901   810     X12V2R3   R    456    0      4104   0
Inc.                            810     X12V2R3   R     45    0       405   0
                                INVOIC  X12V2R3   R    789    0      7101   0

Jay's      Perry, NC   555489   810     X12V2R3   R    102    2       918   7
Turbo                           810     X12V2R3   R     83    3       747   12
                                810     X12V2R3   R     89    0       789   0
                                810     X12V2R3   R    105    0       945   0

Cool Pool  Carmel, CA  748723   810     X12V2R3   R    234    0      1245   2
Service                         810     X12V2R3   R    216    0      1245   4
```

Figure 13. Example of a formatted transaction activity report

TRANSACTION DATA EXTRACT command

This command extracts detailed information about transactions, sorted by transaction handle. You can use this command to create report data to:

- Report on the number of purchase orders sent in a given period of time
- Report on the total number of bytes sent in a given period of time (this can be useful for charging back costs to other departments based on their EDI usage)
- Create a customized functional acknowledgment tracking report by application or by department; for example, an exception report on purchase orders sent more than 2 days ago that have not been acknowledged
- Create an exception report flagging missing control numbers for inbound envelopes

You can also use this command for functions other than reporting, such as:

- Archiving Transaction Store data
- Loading status data directly into the application by application key; for example, the status for each invoice sent could be loaded into the billing system by invoice number

To extract detailed information about enveloped transactions, use the ENVELOPE DATA EXTRACT command (see 35).

Syntax

```
TRANSACTION DATA EXTRACT
ACFIELD(starting application control field data)
        TO(ending application control field data)
APPLICATION(write application data record)
APPLID(application ID)
APPRECID(application receiver department ID)
APPSNDID(sender's department ID)
BATCH(translated transaction batch ID)
CONCATENATE(concatenate extract data)
DIR(processing direction)
DLVDATE(starting delivery date) TO(ending delivery date)
DLVTIME(starting delivery time) TO(ending delivery time)
ENVDATE(starting transaction envelope date)
        TO(ending transaction envelope date)
ENVTIME(starting transaction envelope time)
        TO(ending transaction envelope time)
ENVTYPE(transaction envelope type)
EPURDATE(starting transaction purge date)
        TO(ending transaction purge date)
FORMAT(data format ID)
FUNACKP(pending functional acknowledgment)
GROUP(write group data record)
GRPCTLNO(starting sender's group control number)
        TO(ending sender's group control number)
HANDLE(starting transaction ID) TO(ending transaction ID)
IFCC(override condition codes)
```

TRANSACTION DATA EXTRACT

        IMAGE(*write image data record*)
        INTCTLNO(*starting sender's interchange control nbr*)
            TO(*ending sender's interchange control nbr*)
        INTERCHANGE(*write interchange data record*)
        INTRECID(*interchange receiver ID*)
        INTSNDID(*interchange sender ID*)
        NETACKP(*pending network acknowledgment*)
        NETID(*network ID*)
        NETSTAT(*network transaction status*)
        RECEIVEACKDATA(*write detailed acknowledgment data*)
        RECEIVEACKIMAGE(*write receive acknowledgment record*)
        SENDACKDATA(*write detailed acknowledgment data*)
        SENDACKIMAGE(*write acknowledgment record*)
        SETCC(*condition codes*)
        SNDDATE(*starting request sent date*) TO(*ending request sent date*)
        SNDTIME(*starting request sent time*) TO(*ending request sent time*)
        STDTRID(*EDI standard transaction set ID*)
        STSTAT(*transaction status*)
        TPID(*trading partner ID*)
        TPNICKN(*trading partner nickname*)
        TRANSACTION(*write transaction data record*)
        TRERLVL(*maximum translation error level*)
        TRXCTLNO(*starting transaction set control number*)
            TO(*ending transaction set control number*)
        TRXDATE(*starting transaction date*) TO(*ending transaction date*)
        TRXSTAT(*transaction processing status*)
        TRXTIME(*starting transaction time*) TO(*ending transaction time*)
        USERPGM(*user program*)

TRANSACTION DATA EXTRACT command examples

Example 1:

Retrieve the interchange, group, and transaction information for all transactions with trading partner name **PISCES** and interchange control number **000008888**. Place the results in the EDIQUERY file.

```
PERFORM TRANSACTION DATA EXTRACT
    SELECTING INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y)
    WHERE TPNICKN(PISCES) INTCTLNO(000008888) DIR(S)
```

Example 2:

Retrieve the interchange, group, and transaction information for all transactions with application sender ID **SERVO**. Include application and image information.

```
PERFORM TRANSACTION DATA EXTRACT
    SELECTING INTERCHANGE(Y) GROUP(Y) TRANSACTION(Y) APPLICATION(Y) IMAGE(Y)
    WHERE APPSNDID(SERVO) DIR(S)
```

TRANSFORM command

You can use this command to translate data in any format to any other format defined in your WebSphere Data Interchange system. This command uses data transformation maps to translate the data.

Syntax

TRANSFORM
ACFIELD(*starting application control field data*)
TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
BATCHSET(*set transaction batch ID*)
CLEARFILE(*clear specified file contents*)
DEENVONLY(*deenvelope only indicator*)
DICTIONARY(*input dictionary name*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*)
TO(*ending delivery date*)
DLVTIME(*starting delivery time*)
TO(*ending delivery time*)
DOCUMENT(*input data document name*)
DUPENV(*process duplicate envelopes*)
ENVDATE(*starting transaction envelope date*)
TO(*ending transaction envelope date*)
ENVELOPE(*envelope indicator*)
ENVTIME(*starting transaction envelope time*)
TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
TO(*ending transaction purge date*)
FADELAY(*delay queuing functional acknowledgment*)
FILTERMSGS(*filter error messages*)
FORMAT(*data format ID*)
FUNACKFILE(*functional acknowledgment file name*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLLEN(*group control number length*)
GRPCTLNO(*starting sender's group control number*)
TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*)
IFCC(*override condition codes*)
IGNOREINFO(*ignore informational error messages*)
IGNOREWARN(*ignore warning error messages*)
INFILE(*input data file name*)
INTCTLLEN(*interchange control number length*)
INTCTLNO(*starting sender's interchange control number*)
TO(*ending sender's interchange control number*)
NTRECID(*interchange receiver ID*)

INTSNDID(*interchange sender ID*)
INTYPE(*input data file type*)
MAPID(*map name*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
ONEMSG(*read only one MQ message*)
OPTRECS(*optional record type*)
OUTFILE(*output data file name*)
OUTLEN(*maximum output record length*)
OUTTYPE(*output data file type*)
PURGINT(*purge interval*)
RECOVERY(*recovery unit of work*)
SAPFILE(*SAP file name*)
SAPTYPE(*SAP file type*)
SAPUPDT(*track SAP status*)
SNDDATE(*starting request sent date*)
TO(*ending request sent date*)
SNDTIME(*starting request sent time*)
TO(*ending request sent time*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
STDTRID(*EDI standard transaction set ID*)
SYNTAX(*input data syntax type*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRACELEVEL(*trace level*)
TRXCTLLEN(*transaction control number length*)
TRXCTLNO(*starting transaction set control number*)
TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*)
TO(*ending transaction date*)
TRXSTAT(*starting transaction status*)
TO(*ending transaction status*)
TRXTIME(*starting transaction time*)
TO(*ending transaction time*)
XMLDTDS(*XML DTD path*)
XMLEBCDIC(*EBDCDIC indicator*)
XMLNS(*XML namespace indicator*)
XMLSCHEMAVAL(*XML schema validation*)
XMLSPLIT(*XML Split document*)
XMLVALIDATE(*XML validation indicator*)
XRECS(*Generate X records for C&D*)

TRANSFORM command example

Transform the MQSeries queue file **PURCH1** and output the results in file **PURCHTRN**.

```
PERFORM TRANSFORM
WHERE INFILE(PURCH1) INTYPE(MQ) SYNTAX(X) OUTFILE(PURCHTRN)
```

TRANSLATE AND ENVELOPE command

This command combines the functions of the TRANSLATE TO STANDARD and ENVELOPE commands. This command takes EDI transactions from an application file, translates them to EDI format, places the results in the Transaction Store, and creates envelopes for the transactions. Then, the enveloped transactions are placed in the TD queue or in a file specified in the command. Processing is quicker than using the separate commands, but optimum enveloping is best achieved with the separate commands.

Syntax

TRANSLATE AND ENVELOPE
APPFILE(*application data file name*)
APPSEC(*application file security level*)
APPTYPE(*application file type*)
BATCHSET(*set transaction batch ID*)
DELFILE(*delete application file associated with logical filename*))
DIERRFILTER(*initial error filter set*)
ENVPRBREAK(*start new envelope*)
EXTENDC(*translate with extended C record format*)
FIXEDFILEID(*fixed-to-fixed output ddname*)
IACCESS(*IEXIT access*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
INMEMTRANS(*transactions in memory*)
ITPBREAK(*new interchange envelope*)
ITYPE(*IEXIT program type*)
ONELOGICAPP(*single logical file*)
ONEMSG(*read only one MQ message*)
OPTRECS(*optional record type*)
PAGE(*pageable translation*)
PURGINT(*purge interval*)
RAWDATA(*translate to raw data format*)
RAWFMTID(*raw data format ID*)
RAWUSAGE(*raw data transaction type*)
RECOVERY(*recovery unit of work*)
SAPUPDT(*track SAP status*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
TPID(*trading partner ID*)
XML(*XML required*)
XMLDICT(*XML dictionary address*)
XMLDTDS(*XML DTD path*)

TRANSLATE AND ENVELOPE command examples

Example 1:

Translate and envelope the transactions in ddname **APPDATA**. Place the enveloped transactions in the TD queue specified by the network profile. Return the information record created during translation.

```
PERFORM TRANSLATE AND ENVELOPE
    WHERE APPFILE(APPDATA) OPTRECS(I)
```

Example 2:

In CICS, translate and envelope the transactions in TD queue **AP01**. Place the enveloped transactions in TS queue named **ENVDATA**. Return the information record and envelope headers created during translation.

```
PERFORM TRANSLATE AND ENVELOPE
    WHERE APPFILE(AP01) APPTYPE(TD) FILEID(ENVDATA) OPTRECS(IE)
```

TRANSLATE AND SEND command

This command combines the functions of the TRANSLATE TO STANDARD, ENVELOPE, and SEND commands into one step. Processing is quicker than using the separate commands, but optimum enveloping is best achieved by using the separate commands.

Note: This command will not work properly with direct connection networks such as point-to-point if the application data file contains transactions for more than one trading partner.

Syntax

TRANSLATE AND SEND
APPFILE(*application data file name*)
APPSEC(*application file security level*)
APPTYPE(*application file type*)
ASSERTLVL(*session assertion level*)
BATCHSET(*set transaction batch ID*)
CLEARFILE(*clear specified file contents*)
DELFILE(*delete application file associated with logical filename)*)
DIERRFILTER(*initial error filter set*)
ENVPRBREAK(*start new envelope*)
EXTENDC(*translate with extended C record format*)
FILEID(*processing file ddname*)
FIXEDFILEID(*fixed-to-fixed output ddname*)
IACCESS(*IEXIT access*)
IAREA(*IEXIT information*)
IEXIT(*interchange control program*)
IFCC(*override condition codes*)
INMEMTRANS(*transactions in memory*)
ITPBREAK(*new interchange envelope*)
ITYPE(*IEXIT program type*)
MSGUCLASS(*override message user class*)
ONELOGICAPP(*single logical file*)
ONEMSG(*read only one MQ message*)
OPTRECS(*optional record type*)
PAGE(*pageable translation*)
PURGINT(*purge interval*)
RAWFMTID(*raw data format ID*)
RAWTEST(*raw test data*)
RAWUSAGE(*raw data transaction type*)
RECOVERY(*recovery unit of work*)
REQID(*requestor ID*)
SAPUPDT(*track SAP status*)
SCRIPT(*script ID*)
SEQNUM(*increment network profile member numbers*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)

TRANSLATE AND SEND command examples

Example 1:

Translate and send the transactions in file **APPDATA**. Assign batch number **121401** to these transactions.

```
PERFORM TRANSLATE AND SEND
    WHERE APPFILE(APPDATA) REQID(IINREQ) BATCHSET(121401)
```

Example 2:

Translate and send the transactions in file **RAWFILE**. Data format **POSEND** describes these transactions, which are in raw data format.

```
PERFORM TRANSLATE AND SEND
    WHERE APPFILE(RAWFILE) REQID(IINREQ) RAWFMTID(POSEND)
```

TRANSLATE TO APPLICATION command

This command takes EDI documents from the Transaction Store, translates them to application format, and places the results in the file specified by the data format or in the override file specified by the trading partner usage/rule for translating the transaction. The data can be formatted as C and D records or as raw data.

Note: In CICS, you can also deliver the data to a program or CICS transaction, as specified in the `Application file name` and `Application file type` fields of the data format.

Syntax

TRANSLATE TO APPLICATION
ACFIELD(*starting application control field data*)
       TO(*ending application control field data*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
ASSERTLVL(*session assertion level*)
BATCHSET(*set transaction batch ID*)
CCEXCEPTION(*job-step condition code*)
DIERRFILTER(*initial error filter set*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
       TO(*ending transaction purge date*)
EXTENDC(*translate with extended C record format*)
FORCETEST(*force test usage*)
FORMAT(*data format ID*)
GRPCTLNO(*starting sender's group control number*)
       TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
       TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
MULTIDOCS(*multiple-document file*)
NETID(*network ID*)
OPTRECS(*optional record type*)
PAGE(*pageable translation*)
RAWDATA(*translate to raw data format*)
SETCC(*condition codes*)
STDTRID(*EDI standard transaction set ID*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRXCTLNO(*starting transaction set control number*)
       TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)
XML(*XML required*)

TRANSLATE TO APPLICATION

> XMLDICT(*XML dictionary address*)
> XMLDTDS(*XML DTD path*)
> XMLEBCDIC(*EBDCDIC indicator*)
> XMLSEGINP(*line break indicator*)
> XMLSTDID(*destination EDI standard ID*)
> XMLVALIDATE(*XML validation indicator*)

TRANSLATE TO APPLICATION command examples

Example 1:

Translate all EDI documents in the Transaction Store that were received on December 14, 2001 with network ID **IINR41**. Place the data in the application file specified by the data format or in the override file specified in the receive usage/rule for the trading partner.

```
PERFORM TRANSLATE TO APPLICATION
    WHERE TRXDATE(01/12/14) NETID(IINR41)
```

Example 2:

Translate all EDI documents in the Transaction Store that were received between December 14, 2001 and today's date. Place the data in the application file specified by the data format or in the override file specified in the receive usage/rule for the trading partner.

```
PERFORM TRANSLATE TO APPLICATION
    WHERE TRXDATE(01/12/14) TO(*)
```

Example 3:

Translate all EDI documents in the Transaction Store that were received on December 14, 2001 with network ID **IINR41**. Also translate all EDI data received on December 14, 2001 from trading partner **PISCES**. Place the data in the application file specified by the data format or in the override file specified in the receive usage/rule for the trading partner.

```
PERFORM TRANSLATE TO APPLICATION
    WHERE TRXDATE(01/12/14) NETID(IINR41)
    WHERE TRXDATE(01/12/14) TPNICKN(PISCES)
```

TRANSLATE TO STANDARD command

This command translates application data to an EDI standard format and places the results in the Transaction Store. The application data can be in C and D record format or in raw data format.

Syntax

TRANSLATE TO STANDARD
APPFILE(*application data file name*)
APPSEC(*application file security level*)
APPTYPE(*application file type*)
ASSERTLVL(*session assertion level*)
BATCHSET(*set transaction batch ID*)
DELFILE(*delete application file associated with logical filename)*)
DIERRFILTER(*initial error filter set*)
EENVDATE(*earliest transaction enveloping date*)
EXTENDC(*translate with extended C record format*)
FIXEDFILEID(*fixed-to-fixed output ddname*)
IFCC(*override condition codes*)
ONEMSG(*read only one MQ message*)
OPTRECS(*optional record type*)
PAGE(*pageable translation*)
PURGINT(*purge interval*)
RAWDATA(*translate to raw data format*)
RAWFMTID(*raw data format ID*)
RAWUSAGE(*raw data transaction type*)
SAPUPDT(*track SAP status*)
SETCC(*condition codes*)
TPID(*trading partner ID*)
XML(*XML required*)
XMLDICT(*XML dictionary address*)
XMLDTDS(*XML DTD path*)

TRANSLATE TO STANDARD command examples

Example 1:

Translate the transactions in file **INVOICES** from a raw data format to an EDI standard format and place the results in the Transaction Store. The data format **SENDINVOICES** describes the raw data.

```
PERFORM TRANSLATE TO STANDARD
    WHERE APPFILE(INVOICES) RAWFMTID(SENDINVOICES)
```

Example 2:

Translate the transactions in two application files from C and D record format to EDI standard format and place the results in the Transaction Store. Assign batch ID **121401** to these transactions.

```
PERFORM TRANSLATE TO STANDARD
    WHERE APPFILE(APDATA01) BATCHSET(121401)
    WHERE APPFILE(APDATA02) BATCHSET(121401)
```

UNLOAD LOG ENTRIES command

This command reads all the entries in the event log associated with the `Application ID`. The entries selected for removal are copied to the archive file, and all other entries associated with the `Application ID` are copied to the hold file.

Event log entries that are associated with active or held transactions in the Transaction Store are not eligible for archive and are not selected for removal. The entries selected for removal are copied to the archive file and immediately deleted.

To ensure concurrency, set the number of deletes (`Numdels`) performed before a COMMIT is issued to a relatively low value. This command may be run as the first step of a multi-step process. (See the table on 8.)

Syntax

UNLOAD LOG ENTRIES
APPLID(*application ID*)
ARCHIVEFILE(*event log archive file name*)
ARCHIVETYPE(*event log archive file type*)
HOLDFILE(*event log hold file name*)
HOLDTYPE(*event log hold file type*)
IFCC(*override condition codes*)
LOGAEID(*starting event log associated entry ID*) TO(*ending event log associated entry ID*)
LOGDATE(*starting event log date*) TO(*ending event log date*)
LOGFORM(*starting event log format ID*) TO(*ending event log format ID*)
LOGTIME(*starting event log time*) TO(*ending event log time*)
LOGUSER(*starting event log user ID*) TO(*ending event log user ID*)
NUMDELS(*number of database deletes before commit*)
SETCC(*condition codes*)

UNLOAD LOG ENTRIES command example

Unload log entries from the application log file for application EDIFFS dated December 14, 2001.

```
PERFORM UNLOAD LOG ENTRIES
    WHERE APPLID(EDIFFS) LOGDATE(12/14/01)ARCHIVEFILE(ARCHTRAN)
      HOLDFILE (HOLDTRAN)
```

UNPURGE command

This command restores a transaction to the status it had before it was marked for purging unless the transaction's storage time has expired. You can restore transactions marked for purging by the PURGE command until you execute the REMOVE TRANSACTIONS command. Once the REMOVE TRANSACTIONS command runs, you cannot restore the transactions. If a transaction is one of a related group, all transactions in the group are restored to their former status. If a transaction's storage time has expired, it can still be eligible for purging. Running the UNPURGE command does not change the purge status of transactions whose storage time has expired.

The HOLD, PURGE, RELEASE, and UNPURGE commands share a common syntax.

Syntax

UNPURGE
ACFIELD(*starting application control field data*)
    TO(*ending application control field data*)
APPLID(*application ID*)
APPRECID(*application receiver department ID*)
APPSNDID(*sender's department ID*)
BATCH(*translated transaction batch ID*)
DIR(*processing direction*)
DLVDATE(*starting delivery date*) TO(*ending delivery date*)
DLVTIME(*starting delivery time*) TO(*ending delivery time*)
ENVDATE(*starting transaction envelope date*)
    TO(*ending transaction envelope date*)
ENVTIME(*starting transaction envelope time*)
    TO(*ending transaction envelope time*)
ENVTYPE(*transaction envelope type*)
EPURDATE(*starting transaction purge date*)
    TO(*ending transaction purge date*)
FORMAT(*data format ID*)
FUNACKP(*pending functional acknowledgment*)
GRPCTLNO(*starting sender's group control number*)
    TO(*ending sender's group control number*)
HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
IFCC(*override condition codes*)
INTCTLNO(*starting sender's interchange control nbr*)
    TO(*ending sender's interchange control nbr*)
INTRECID(*interchange receiver ID*)
INTSNDID(*interchange sender ID*)
NETACKP(*pending network acknowledgment*)
NETID(*network ID*)
NETSTAT(*network transaction status*)
SETCC(*condition codes*)
SNDDATE(*starting request sent date*) TO(*ending request sent date*)
SNDTIME(*starting request sent time*) TO(*ending request sent time*)
STDTRID(*EDI standard transaction set ID*)
STSTAT(*transaction status*)
TPID(*trading partner ID*)

TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*)
        TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)

UNPURGE command example

Restore all EDI documents with application control numbers **PO112233** through **PO112244** that are marked for purging by user request.

```
PERFORM UNPURGE
    WHERE ACFIELD(PO112233) TO(PO112244) STSTAT(4)
```

UPDATE STATISTICS command

The UPDATE STATISTICS command is used to update the management reporting statistics from the pending statistics table to the reporting tables. You can use the NUMUPDTS keyword to limit the number of updates performed at one time.

Note: You must run this command before running management reports.

Syntax

UPDATE STATISTICS
NUMUPDTS(*number of database updates before commit*)

UPDATE STATISTICS command example

Update the statistics tables before you run a management reporting data extract. Suppose you use the DB2 version of WebSphere Data Interchange. Because of application requirements, the system is running multiple simultaneous translation jobs and it is heavily loaded. Also, the database administrator has set the DB2 timeout value rather low. As a result, you are encountering DB2 timeouts. To reduce the amount of time that the UPDATE STATISTICS process holds table locks, set the NUMUPDTS value to a low number such as 25. This causes the UPDATE STATISTICS job to commit work and release all locks after every 25 DB2 updates.

```
PERFORM UPDATE STATISTICS
    WHERE NUMUPDTS(25)
```

UPDATE STATUS command

This command retrieves network acknowledgments for all mailbox (requestor) profile members, specified members, or specified network profile members. It processes all network acknowledgments that resulted from previous SEND commands, pairs the acknowledgments with the transactions originally sent, and updates transaction status.

The WHERE clause is optional for this command. If you do not specify a WHERE clause, all network profile members are processed.

You can request network acknowledgments in the Net acknowledgment field of the trading partner or mailbox (requestor) profile.

Syntax

UPDATE STATUS
IFCC(*override condition codes*)
NETID(*network ID*)
REQID(*requestor ID*)
SETCC(*condition codes*)

UPDATE STATUS command examples

Example 1:
Receive and process all network acknowledgments for all mailbox (requestor) profile members.

```
PERFORM UPDATE STATUS
```

Example 2:
Receive and process network acknowledgments for all mailbox (requestor) profile members with network profile **IINB41**.

```
PERFORM UPDATE STATUS
    WHERE NETID(IINB41)
```

Example 3:
Receive and process all network acknowledgments for mailbox (requestor) profile members, **ROBOX** and **KANBOX**.

```
PERFORM UPDATE STATUS
    WHERE REQID(ROBOX)
    WHERE REQID(KANBOX)
```

Keyword descriptions

This section describes the keywords used with the WebSphere Data Interchange Utility commands.

ACCTID

The Network Account ID of the trading partner as entered in the `Account Number` field of the trading partner profile. The maximum length is 32.

This keyword is used with the following commands:

```
NETWORK ACTIVITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
```

ACFIELD

The application control field or fields. Consists of either the `AC` field in the data format or the concatenation of data in the fields specified during transaction translation. For more information about the application control field, refer to the WebSphere Data Interchange User's Guide. Concatenated field data is used if any application control field names are specified in transaction translation. This field contains control numbers such as purchase order numbers and is case sensitive. The maximum length is 35.

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATION TO APPLICATION
UNPURGE
```

ACKFILE

The ddname of a file containing network acknowledgments. If you do not specify this keyword, the value of the `Net output file` field in the associated network profile member is used.

This keyword is used only with the PROCESS NETWORK ACKS command.

ACKFILE

> **Note:** In CICS, WebSphere Data Interchange does not change the ACKFILE value to uppercase.

ACKTYPE

The acknowledgments file type used only for CICS and MQ (which is supported in both z/OS and CICS). Valid values are:

MQ   WebSphere Data Interchange MQSeries queue profile member name

TD   Transient data queue

TM   Temporary storage queue - main storage

TS   Temporary storage queue - auxiliary storage (default for CICS)

VS   VSAM entry sequenced data set

This keyword is used only with the PROCESS NETWORK ACKS command.

If you do not specify this keyword for z/OS, this field is ignored and the corresponding file name (the ddname of a sequential file) in the network profile member is used.

ACTUSAGE

Indicates how WebSphere Data Interchange should handle the activation of imported usages and rules. Valid values are:

NOREPL

An active imported usage/rule replaces an existing active usage/rule in the database (default). If an active usage/rule that matches the key for the imported usage/rule already exists, the imported usage/rule is made inactive.

REPL   An active imported usage/rule replaces an existing active usage/rule in the database. In this case, the existing usage/rule is made inactive and the imported usage/rule becomes the active usage.

FORCE

All imported usages/rules are forced to be active regardless of their current status, or the existence of another active usage/rule in the database. If an active usage/rule that matches the key for the imported usage/rule exists in the database, it is made inactive and the imported usage/rule becomes the active.

This keyword is used only with the IMPORT command.

ADDRLN1

The first line of the address as entered in the `Address line 1` field of the trading partner profile. This field is case sensitive. The maximum length is 40.

This keyword is used with the following commands:
```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

ADDRLN2

The second line of the address as entered in the `Address line 2` field of the trading partner profile. This field is case sensitive. The maximum length is 40.

This keyword is used with the following commands:

```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

AGENCY

The responsible agency code specified on the send or receive map usage. The maximum length is 8. Default is blank.

This keyword is used with the following commands:

```
DELETE USAGE
```

APPFILE

When used with EDI processing commands, this keyword specifies the ddname of the file containing the application data. The data can be C and D records or raw data records. For the format of these records, "Application file" on page 195. The maximum length is eight.

This keyword is used with the following commands:

```
DEENVELOPE AND TRANSLATE
RECEIVE AND SEND
RECEIVE AND TRANSLATE
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

Note: In CICS, WebSphere Data Interchange does not change the `APPFILE` value to uppercase.

APPLICATION

Indicates whether an application data record is written to the EDIQUERY file. Valid values are:

Y   Writes application data records

N   Discards application data records (default)

Received transactions can be translated multiple times and will have an application record for each attempt. Send transactions only have one application record. For the format of these records, "Application data extract record layout" on page 309.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

APPLID

The application ID. When used as a selection criteria, this field specifies the application ID that was used to create the records being examined. (These records may be transactions in the Transaction Store or entries in the event log.) Valid values are:

EDIFFS
WebSphere Data Interchange Utility

EDIMP WebSphere Data Interchange Facility

(user-defined)
Your application APPLID

You can switch to another application ID by using WebSphere Data Interchange's API function codes. The maximum length is eight.

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
LOAD LOG ENTRIES
NETWORK ACTIVITY DATA EXTRACT
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE LOG ENTRIES
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNLOAD LOG ENTRIES
UNPURGE
```

APPLTPID

The application trading partner nickname specified on the send or receive map usage. The maximum length is 16.

This keyword is used with the following commands:

```
DELETE USAGE
```

APPRECID

The application receiver ID. Identifies the specific department or business area in the receiver's company. The translator uses this ID first to attempt to locate the trading partner usage/rule for receive transactions. The maximum length is 35.

This keyword is used with the following commands:

```
DELETE USAGE
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNPURGE
```

APPSEC

The security level of the application file. Indicates whether the application files specified with the `APPFILE` keyword are read only or read and write. Valid values are:

U   Opens the application files for both read and write. The files can be opened for extend, and then closed (default). This process ensures that each file has been properly initialized and that WebSphere Data Interchange does not process data that is not valid.

R   Opens the application files in read only mode. If this option is chosen, you must make sure the application files are properly initialized.

This keyword is used with the following commands:

```
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD
```

APPSNDID

A specific department or business area in the sender's company. The maximum length is 35.

This keyword is used with the following commands:

```
DELETE USAGE
HOLD
PRINT ACKNOWLEDGMENT IMAGE
```

APPSNDID

```
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSLATE TO APPLICATIONS
UNPURGE
```

APPTYPE

Indicates the application file type. Used only for CICS and MQ (which is supported in both z/OS and CICS). You can use this keyword to override the `Receive application file name` specified in the data format or specified on the Receive Transaction Usage Override panel. Valid values are:

MQ   WebSphere Data Interchange MQSeries queue profile member name

PG   Response program (inbound processing only)

TD   Transient data queue

TM   Temporary storage queue - main storage

TS   Temporary storage queue - auxiliary storage (default for CICS)

TX   Response transaction (inbound processing only)

This keyword is used with the following commands:
```
DEENVELOPE AND TRANSLATE
RECEIVE AND TRANSLATE
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

If you do not specify this keyword for z/OS, this field is ignored and the corresponding file keyword (the ddname of a sequential file) is used.

ARCHIVEFILE

The event log archive file name. Event log entries selected for removal are written to this file. The maximum length is eight.

This keyword is used with the following commands:
```
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES
```

ARCHIVETYPE

Indicates the event log archive file type used only for CICS and MQ (which is supported in both z/OS and CICS). Valid values are:

MQ   WebSphere Data Interchange MQSeries queue profile member name

TD   Transient data queue

TM   Temporary storage queue - main storage

TS   Temporary storage queue - auxiliary storage (default for CICS)

This keyword is used with the following commands:

```
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES
```

If you do not specify this keyword for z/OS, this field is ignored and the corresponding file keyword (the ddname of a sequential file) is used.

ASSERTLVL

Specifies the level of assertions that are active for this translation session. Assertions are established during the translation process using the &ASSERT*n* special literals. The *n* can be a value from **0** to **9**, which establishes 10 assertion levels. Only &ASSERT requests with an *n* value greater than or equal to the ASSERTLVL value will be evaluated. For example, when ASSERTLVL(**5**) is specified, only the literals &ASSERT5, &ASSERT6, &ASSERT7, &ASSERT8, and &ASSERT9 will be evaluated. Level 9 assertions (&ASSERT9) are always active.

This keyword is used with the following commands:

```
DEENVELOPE AND TRANSLATE
RECEIVE AND TRANSLATE
RETRANSLATE TO APPLICATION
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

BATCH

Specifies the batch ID assigned to a transaction when it was translated. You can use this keyword to select only transactions with a particular batch ID. The maximum length is eight.

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
```

BATCH

```
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNPURGE
```

BATCHSET

Specifies that a user-defined ID be assigned to the transaction when it is translated. You can use this keyword to identify transactions you want to retrieve as a group. The transactions remain independent. An action performed on one transaction does not affect other transactions with the same batch ID. The maximum length is eight. The default batch ID is a date and time stamp in the format ddhhmmss.

This keyword is used with the following commands:

```
DEENVELOPE AND TRANSLATE
RECEIVE AND TRANSLATE
RETRANSLATE TO APPLICATION
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

CCEXCEPTION

Indicates whether job-step condition code 0003 or 0000 is returned when application data is written to the exception file. Valid values are:

Y Generates a minimum job step condition code of 0003 if application data is written to the exception file.

N Generates a job-step condition code of 0000 if translation completes successfully and the application data was written to the exception file (default). Does not report application data written to the exception file as an error.

X Generates a job-step condition code of 903 to distinguish this from other errors that result in a 0003 condition code.

This keyword is used with the following commands:

```
DEENVELOPE AND TRANSLATE
RECEIVE AND TRANSLATE
RETRANSLATE TO APPLICATION
TRANSLATE TO APPLICATION
```

CLEARFILE

Indicates whether the specified file is cleared before it is used for further processing.

For the TRANSFORM command, this keyword indicates whether the output file or queue is cleared before writing the translated output. Valid values are:

Y   Clears the file before writing the translated output

N   Does not clear the file before writing the translated output (default)

For the following commands, this keyword indicates whether the receive file is cleared after a send is completed or before a receive is issued. Valid values are:

Y   Clears the file after a send is completed or before a receive is issued

N   Does not clear the file after a send is completed or before a receive is issued (default)

```
ENVELOPE AND SEND
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND SEND
RECEIVE AND TRANSLATE
RECVFILE AND SEND
REENVELOPE AND SEND
SEND
TRANSLATE AND SEND
```

Note: This keyword is ignored for batch receive processes. The file allocation disposition in the batch JCL will determine if the TD queue file is cleared.

CLIENT

Indicates which SAP status records are extracted or removed using the SAP client ID. The default is **ALL**.

This keyword is used with the following commands:

```
SAP STATUS EXTRACT
SAP STATUS REMOVE
```

CMMTLN1

The first comment line as entered in the `Comment line 1` field of the trading partner profile. This field is case sensitive. The maximum length is 40.

This keyword is used with the following commands:

```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

CMMTLN2

The second comment line as entered in the `Comment line 2` field of the trading partner profile. This field is case sensitive. The maximum length is 40.

CMMTLN2

This keyword is used with the following commands:
```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

CMPYNM

The company name as entered in the `Company name` field of the trading partner profile. This field is case sensitive. The maximum length is 40.

This keyword is used with the following commands:
```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

CNTCTNM

The contact name as entered in the `Contact name` field of the trading partner profile. This field is case sensitive. The maximum length is 30.

This keyword is used only with the TRADING PARTNER PROFILE DATA EXTRACT command.

CNTCTPH

The contact phone number as entered in the `Contact phone number` field of the trading partner profile. The maximum length is 25.

This keyword is used only with the TRADING PARTNER PROFILE DATA EXTRACT command.

CONCATENATE

Indicates whether requested data extract information is written as separate records, or concatenated and written as a single record. Valid values are:

Y   Concatenates categories and writes them as a single record

N   Writes information as separate records (default)

If concatenation is requested, the following hierarchy is used:
1. Interchange
2. Group
3. Transaction
4. Application

Each application entry written for a transaction is included with duplicate interchange, group, and transaction information.

Note: Concatenation does not apply to image records. Transaction and acknowledgment images are always written as separate records. Detailed acknowledgment data is always concatenated.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

CTLFILE

The ddname of an import or export control file that describes what data is to be imported or exported. For information about the control file format, see the table on 209.

This keyword is used with the following commands:

```
EXPORT
IMPORT
```

CTLTYPE

Indicates the import or export control file type. Used only for CICS and MQ (which is supported in both z/OS and CICS). Valid values are:

MQ   WebSphere Data Interchange MQSeries queue profile member name

TD   Transient data queue

TM   Temporary storage queue - main storage

TS   Temporary storage queue - auxiliary storage (default for CICS)

This keyword is used with the following commands:

```
EXPORT
IMPORT
```

If you do not specify this keyword for z/OS, this field is ignored and the default for the corresponding file keyword is the ddname of a sequential file.

DAYS

A single date, or the start date of a date range. If the start date of a date range, this keyword must be followed by the T0 keyword with an ending value.

This keyword is used with the following commands:

```
NETWORK ACTIVITY DATA EXTRACT
SAP STATUS EXTRACT
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

DEENVONLY

Default is N. If set to Y, this means the data will only be deenveloped and put in transaction store, and will not be translated. (Functionally equivalent to PERFORM DEENVELOPE.)

DELFILE

If **DELFILE** is specified on a **PERFORM TRANSLATE TO STANDARD**, **PERFORM TRANSLATE AND ENVELOPE** or a **PERFORM TRANSLATE AND SEND** command, the application file associated with the logical filename will be deleted once a successful translation is complete.

If an error occurs and the DF could not be translated then the file will not be deleted.

Note: **DELFILE** will delete the file on AIX and Windows but on z/OS the file will be emptied of its contents rather than being deleted.

DICTIONARY

Specifies the dictionary name for the input data. For application data, this field is required. For EDI and XML data, this field will override values extracted from the data. For EDI data, if you do not specify this keyword, the values in GS08 (X12) and UNH02 (EDIFACT) will be used to determine the value for this field in the EDI2DICT translation table shipped with WebSphere Data Interchange. For XML data, if you do not specify this keyword, the root element will be used to determine the dictionary name.

This keyword is used only with the TRANSFORM command.

DIERRFILTER

Specifies the initial set of errors to filter for this translation session. To override the filters defined in a usage/rule, you can use the DIERRFILTER keyword with a value of **IGNORE** to tell WebSphere Data Interchange to ignore all reported errors. This allows you to view all errors for a particular session without changing the map generally used. For more information on error filtering, see "Error filtering" on page 22 or the field definition for ERRFILTER on page 455.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE
REENVELOPE AND SEND
RETRANSLATE TO APPLICATION
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

DIR

Indicates the direction of the transmission. Valid values are:

R   Receiving

S   Sending

This keyword is used with the following commands:

```
DELETE USAGE(default R)
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RECONSTRUCT
RECONSTRUCT AND SEND
RELEASE
REMOVE TRANSACTIONS
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
TRANSACTION DATA EXTRACT
UNPURGE
```

DLVDATE

Specifies the date, or a range of dates, when the transactions you want to work with were delivered to the application. If this is the start date of a date range, it must be followed by the TO keyword with an ending value.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNPURGE
```

DLVTIME

Specifies the time, or a period of time, when the transactions you want to work with were delivered to the application. If this is the start time of a time range, it must be followed by the T0 keyword with an ending value.

This keyword is used with the following commands:

DLVTIME

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNPURGE
```

DOCUMENT

Specifies the document name for the input data. For application data, this field is required. For EDI and XML data, this field will override values extracted from the data. For EDI data, if you do not specify this keyword, the values in ST01 (X12) and UNH02 (EDIFACT) will be used to determine the value for this field. For XML data, if you do not specify this keyword, the root element will be used to determine the document name.

This keyword is used only with the TRANSFORM command.

DUPCHECK

Indicates whether duplicate account/user ID and duplicate interchange ID/qualifier checks are performed when importing TPPROF members. This keyword is optional. Valid values are:

Y (or other)
Checks for duplicate IDs and ID qualifiers on TPPROF import (default)

N   Bypasses the duplicate checks on TPPROF import

Note: The import program checks the incoming TPPROF members to ensure that no duplicate accounts/user IDs or duplicate interchange ID qualifiers are imported. This is a time-consuming task and if you are confident that there are no duplicates in the import file, this keyword can be specified with a value of **N** to improve import performance. If duplicates are imported into the TPPROF and are referenced during translation, unpredictable results can occur.

This keyword is used only with the IMPORT command.

DUPENV

Indicates whether duplicate envelopes should be processed. Valid values are:

Y   Processes duplicate envelopes (default).

N   Does not process duplicate envelopes. A condition code of 0005 is returned if you attempt to process a duplicate envelope.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
```

DYNSQL

Indicates whether dynamic SQL should be used for extracting the management report data. Dynamic SQL provides significant performance improvements if you maintain a lot of statistics. It allows DB2 to create an optimized plan specifically for your query. To use dynamic SQL, you must be authorized by your database administrator to access the appropriate views. Valid values are:

Y   Uses dynamic SQL

N   Uses static SQL (default)

This keyword is used with the following commands:

```
NETWORK ACTIVITY DATA EXTRACT
TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

EENVDATE

Specifies the earliest date that transactions created during this run can be enveloped. The `Date mask` field in the language profile determines the format of the date code.

This keyword is used only with the TRANSLATE TO STANDARD command.

EIFORMAT

Indicates the requested export file record format. Valid values are:

TAGGED
          Exports in tagged record format (default)

FIXED   Exports in fixed record format

This keyword is used only with the EXPORT command.

ENVDATE

Specifies the date on which the transaction was enveloped. The maximum length is 10.

This keyword is used with the following commands:

```
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
```

ENVDATE

```
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
TRANSLATE TO STANDARD
UNPURGE
```

ENVELOPE

Indicates if the EDI data should be enveloped (only used for outbound EDI). If N, the transactions are just added to the store for later enveloping. Default is Y.

ENVPRBREAK

Indicates whether a new interchange envelope or a new group envelope is started when the EDI standard envelope profile member name changes. Usually, the envelope profile provides the group envelope information. Use this keyword if your envelope profile provides interchange envelope information. Valid values are:

Y   Starts a new interchange envelope

N   Starts a new group envelope (default)

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

ENVTIME

The time at which the transaction was enveloped. The maximum length is eight.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
```

```
REENVELOPE AND SEND
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
UNPURGE
```

ENVTYPE

For selecting transactions, indIcates the type of envelope used. Valid values are:

E   UNB/UNZ

I   ICS

T   STX/END

U   BG/EG

X   ISA/IEA

0   Envelopes with no interchange header and trailer

For sending and receiving, indicates the type of receive issued. Valid values are:

E   EDIFACT

I   ICS or non-EDI file

T   UN/TDI

U   UCS

X   X12 (default)

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND SEND
RECEIVE AND TRANSLATE
RECVFILE AND SEND
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
```

ENVTYPE

```
SEND
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

EPURDATE

The date on which the transaction will be purged from the Transaction Store. The translator sets the default purge date when it adds the transaction to the Transaction Store. You can override the default length of time that the transaction can stay in the Transaction Store before being purged by using the PURGINT keyword when translating the transaction. The maximum length is 10.

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

EXTENDC

Indicates whether the extended C record format is used when translating to application data with C and D records. Valid values are:

Y   Uses the extended C record format

N   Does not use the extended C record format (default)

This keyword is used with the following commands:

```
DEENVELOPE AND TRANSLATE
RECEIVE AND TRANSLATE
RETRANSLATE TO APPLICATION
TRANSLATE TO APPLICATION
```

EXTRACTED

To select previously extracted records. When used with command SAP STATUS EXTRACT, default is N. When used with command SAP STATUS REMOVE, default is Y.

Y extract only records already marked as extract

N extract only records NOT marked as extracted

A extract ALL records

FADELAY

Indicates whether functional acknowledgments are immediately enveloped and queued for sending, or are placed in the Transaction Store for enveloping and sending later. Valid values are:

Y Puts functional acknowledgments in the Transaction Store but does not envelope them

N Puts functional acknowledgments in the Transaction Store and also envelopes them to one of the following (default) files:
- The file specified by the `FUNACKFILE` keyword, if present
- The TD queue from the network profile
- QDATA, QDATAU, or QDATAE (depending on envelope type)

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
```

FILEID

Specifies the ddname of a file used to:
- Write data during an ENVELOPE operation
- Read data during a DEENVELOPE operation
- Send data during a SEND operation
- Write data during a RECEIVE operation

For outbound processing, if you do not specify this keyword, the value from the `Transaction data queue` field in the network profile is used as the ddname for the envelope file.

For inbound processing, if you do not specify this keyword, the value from the `Receive file name` field from the mailbox (requestor) profile is used as the ddname for the envelope file.

The maximum length is eight.

FILEID

Notes:

1. On ENVELOPE and DEENVELOPE commands, all transactions are placed in this file. You should also specify the `NETID` keyword to make sure all transactions you select are for the same network.

2. `FILEID` contains the name of a TS queue. You should include this keyword to make sure that different applications running in the same CICS region do not envelope transactions to the same TS queue.

3. In CICS, WebSphere Data Interchange does not change the `FILEID` value to uppercase.

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND SEND
ENVELOPE
ENVELOPE AND SEND
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND SEND
RECEIVE AND TRANSLATE
RECONSTRUCT
RECONSTRUCT AND SEND
RECVFILE AND SEND
REENVELOPE
REENVELOPE AND SEND
SEND
SENDFILE
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

FILTERMSGS

For data transformation processing. Filters messages in the list, if severity is less than 8. Maximum length is 80. Up to 11 individual messages may be filtered. See also IGNOREWARN and IGNOREINFO keywords.

This keyword is used with the following commands:
```
TRANSFORM
```

FIXEDFILEID

Specifies the ddname of the file used for output during fixed-to-fixed translation processing. Data is written to the file during an ENVELOPE operation for fixed-to-fixed translation. The maximum length is eight.

If you do not specify this keyword, the ddname (based on the EDI standard ID) is the same as the `Application file name` in the target DF definition. To create a unique envelope file for each trading partner, specify the `File Suffix` field in the trading partner profile.

Notes:

1. On ENVELOPE commands, all transactions are placed in this file. You should also specify the `NETID` keyword to make sure all transactions you select are for the same network.

2. `FIXEDFILEID` contains the name of a TS queue. You should include this keyword to make sure that different applications running in the same CICS region do not envelope transactions to the same TS queue.

3. In CICS, WebSphere Data Interchange does not change the `FIXEDFILEID` value to uppercase.

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

FORCETEST

Indicates whether the deenvelope or translate-to-application processes should be forced to select a test usage/rule, regardless of the test indicator value in the envelope. This keyword is most useful when receiving test envelopes that do not have a test indicator (such as the UCS BG). In this case, you can use this keyword to force the translator to consider the envelopes for testing and only look for test usages/rules. Valid values are:

Y   Forces the process to test mode and select only a test usage/rule if one is defined. If a test usage/rule is not found, an error is generated and the transaction is rejected.

If `FORCETEST`(**Y**) is used with the DEENVELOPE command, it must also be used on the TRANSLATE TO APPLICATION or RETRANSLATE TO APPLICATION commands to select the deenveloped transactions.

N   Uses the test indicator from the envelope to determine which usage/rule to select (default). An envelope without a test indicator is always considered a production envelope.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
TRANSLATE TO APPLICATION
RETRANSLATE TO APPLICATION
```

FORMAT

The ID of the data format associated with the transaction or transactions you want to select. The maximum length is 16.

This keyword is used with the following commands:

FORMAT

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

FUNACKFILE

The ddname of the file that you want to use for returning functional acknowledgments for the deenveloped transactions. You can use this keyword if you do not want to use the TD queue specified in the network profile. The maximum length is eight.

Note: In CICS, WebSphere Data Interchange does not change the `FUNACKFILE` value to uppercase and `FUNACKFILE` contains the name of a TS queue.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
TRANSFORM
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
```

FUNACKP

Indicates whether transactions with functional acknowledgments pending should be selected. Valid values are:

Y   Selects transactions for which a functional acknowledgment was requested but not received

N   Selects transactions for which a functional acknowledgment was not requested or has already been received

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
```

```
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RECEIVE
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
UNPURGE
```

FUNACKREQ

Indicates whether the functional acknowledgment envelope file is required. WebSphere Data Interchange will produce an error if unable to open it. Valid values are:

Y   Requires the functional acknowledgment envelope file

N   Does not require the functional acknowledgment envelope file (default)

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
```

GROUP

Indicates whether group data records are written to the EDIQUERY file. Valid values are:

Y   Writes group data records

N   Discards group data records (default)

For the format of these records, "Group data extract record layout" on page 305.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

GRPCTLNO

Specifies the group control number assigned by the sender to identify the functional group. The maximum length is 14.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
```

```
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION DATA EXTRACT
UNPURGE
```

HANDLE

Specifies the ID assigned by the system to a transaction when it is placed in the Transaction Store. To ensure uniqueness, the ID is a concatenation of the date, time, and a sequence number in format: YYYYMMDDHHMMSSnnnnnn

You can use this keyword to envelope a specific EDI document or all the documents whose time stamp falls within a given range. The system left-justifies and pads your entries. The FROM value is padded with 0s and the TO value is padded with 9s. To select transactions for the current date, use an asterisk (*).

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

HOLDFILE

The event log hold file name. The UNLOAD LOG ENTRIES command copies the non-archived event log entries into this file. The LOAD LOG ENTRIES command loads the records from this file into the event log. The maximum length is eight.

This keyword is used with the following commands:

```
LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES
```

HOLDTYPE

Indicates the event log hold file type. Used only for CICS and MQ (which is supported in both z/OS and CICS). Valid values are:

MQ   WebSphere Data Interchange MQSeries queue profile member name

TD   Transient data queue

TM   Temporary storage queue - main storage

TS   Temporary storage queue - auxiliary storage (default for CICS)

In z/OS, the default is the ddname of the sequential file.

This keyword is used with the following commands:

```
LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES
```

IACCESS

Indicates how the interchange should be presented to the IEXIT program. Valid values are:

F   Gives the interchange to the exit in a file. The interchange is written to the TD queue file, and then the IEXIT program is started.

M   Gives the interchange to the exit in virtual storage. Applies only when `ITYPE` is **UE** (user exit).

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

IAREA

Specifies up to 16 bytes of information, the address of which is provided to the IEXIT program. Applies only to z/OS programs. In CICS, the address provided to the IEXIT program is the address of the utility control block. The maximum length is 16.

IAREA

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

ID

The name of a WebSphere Data Interchange profile such as Mailbox (REQPROF), Trading Partner (TPPROF), or Network (NETPROF). This must be a valid WebSphere Data Interchange profile ID. The maximum length is eight.

This keyword is used with the following commands:

```
QUERY PROFILE
DELETE PROFILE
```

IEXIT

The name of the program to receive control as each interchange is processed. See the `ITYPE` field description on 156 for the types of program you can use.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

IFCC

Specifies the condition codes that you want to override. You can specify up to 10 utility condition codes, separated by commas. The codes are checked by the WebSphere Data Interchange Utility and overridden with values from the `SETCC` keyword on a one-to-one basis. If you specify this keyword, you must specify the SETCC keyword.

This keyword can be used with any utility PERFORM command.

IGNOREINFO

For data transformation processing. Filters all informational messages. Default is (N).

This keyword is used with the following commands:

```
TRANSFORM
```

IGNOREWARN

For data transformation. Filters all warning messages. Default is (N).

This keyword is used with the following commands:
```
TRANSFORM
```

IMAGE

Indicates whether `image data records` are written to the EDIQUERY file. Valid values are:

Y   Writes image data records

N   Discards image data records (default)

Images are always written as separate records. For the format of these records, "Transaction/Acknowledgment image data extract record layout" on page 310.

This keyword is used with the following commands:
```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

INFILE

The input file containing the data. For z/OS, this is the ddname or MQSeries queue name. For CICS this is TS or TD queue, MQSeries queue, or VSAM data set.

This keyword is used only with the TRANSFORM command.

INMEMTRANS

Specifies the number of transactions held in memory before the database updates are attempted. This field is valid only if envelope level recovery (`RECOVERY`(**E**)) is in effect. Keeping transactions in storage delays the time when the database lock is obtained and reduces the length of time that the database lock is held. The more transactions kept in storage, the higher concurrency rate WebSphere Data Interchange can achieve. The amount of storage used for each `transaction` is approximately 2K. Valid values are **1** to **65535**. The default is **100**.

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

INTCTLNO

Specifies the interchange control number assigned by the sender to identify the interchange data. The maximum length is 14.

This keyword is used with the following commands:
```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RECONSTRUCT
RECONSTRUCT AND SEND
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

INTERCHANGE

Indicates whether the interchange data record is written to the EDIQUERY file. Valid values are:

Y   Writes interchange data records

N   Discards interchange data records (default)

For the format of these records, "Interchange data extract record layout" on page 304.

This keyword is used with the following commands:
```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

INTID

The interchange sender/receiver ID of the trading partner as entered in the `Interchange ID` field of the trading partner profile.

This keyword is used only with the TRADING PARTNER PROFILE DATA EXTRACT command.

INTRECID

The interchange receiver ID (assigned by the receiver) that identifies the receiver to the sender. The maximum length is 35.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RECONSTRUCT
RECONSTRUCT AND SEND
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

INTSNDID

The interchange sender ID (assigned by the sender) that identifies the sender to the receiver. The maximum length is 35.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

INTYPE

Indicates the type of file specified with the `INFILE` keyword. Used only for CICS and MQ (which is supported in both z/OS and CICS). For CICS, when you specify the `INFILE` keyword, you must also specify this keyword. Valid values are:

MQ   WebSphere Data Interchange MQSeries queue profile member name

TD   Transient data queue

TM   Temporary storage queue - main storage

TS   Temporary storage queue - auxiliary storage (default for CICS)

VS   VSAM data base

This keyword is used only with the TRANSFORM command.

ITPBREAK

Indicates whether a new interchange envelope starts when the internal trading partner ID changes. Valid values are:

Y   Always starts new interchange envelope (default)

N   Does not necessarily start a new interchange envelope

This keyword is used with the following commands:
```
ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

ITYPE

Indicates the type of program specified in IEXIT. If you specify this keyword, you must also specify the IEXIT keyword. Valid values are:

PG   A program that should be linked to using the EXEC CICS LINK command in CICS

UE   A WebSphere Data Interchange user exit program defined in the User Exits profile

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

LASTTRXDATE

The date of the last transaction sent to or received from the trading partner.

This keyword is used only with the TRADING PARTNER PROFILE DATA EXTRACT command.

LEVEL

Specifies what information is dumped or how much detail is traced during processing.

This keyword is used with the following commands
```
GLB DUMP
GLB TRACE
```

For the GLB DUMP command, valid values are:

1   Dumps everything (default)

2   Dumps working storage

3   Dumps entire dataspace

10   Dumps common area

11   Dumps record index area

12   Dumps data area

For the GLB TRACE command, valid values are:

0   Ends trace (the default)

1   Starts trace functions only

2   Starts trace functions and subroutines

3   Starts trace functions, subroutines, and subroutine keypoints

LOGAEID

The associated entry ID in the event log. This keyword can be used with the T0 keyword to specify a range of associated entry IDs for selecting event log entries. The maximum length is 40.

This keyword is used with the following commands:
```
LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES
```

LOGDATE

The event log date. This keyword can be used with the T0 keyword to specify a range of dates for selecting event log entries. The maximum length is eight.

This keyword is used with the following commands:

LOGDATE

```
LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES
```

LOGFORM

The event log format ID. This keyword can be used with the `TO` keyword to specify a range of format IDs for selecting event log entries. The maximum length is 16.

This keyword is used with the following commands:

```
LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES
```

LOGTIME

The event log time. This keyword can be used with the `TO` keyword to specify a range of times for selecting event log entries. The length is six.

This keyword is used with the following commands:

```
LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES
```

LOGUSER

The event log user ID. This keyword can be used with the `TO` keyword to specify a range of user IDs for selecting event log entries. The maximum length is eight.

This keyword is used with the following commands:

```
LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES
```

MAPID

The map name. For the TRANSFORM command, the value in this field overrides the map name specified in data transformation rules.

This keyword is used with the following commands:

```
DELETE USAGE
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
TRANSFORM
```

MAXRUNTIME

The maximum time in minutes that the REMOVE TRANSACTIONS process is allowed to run. Once the specified time is reached, the REMOVE TRANSACTIONS process stops even if it has not completed. When the REMOVE TRANSACTIONS process is running, other WebSphere Data Interchange processes are prevented from running. Setting this keyword and value is useful when the REMOVE TRANSACTION process is to run standalone for a limited time. The default value is **0** (no maximum run time).

This keyword is used only with the REMOVE TRANSACTIONS command.

MEMBER

The name of a WebSphere Data Interchange profile member. The maximum length is dependent upon the type of profile but cannot exceed 35.

This keyword is used with the following commands:

```
DELETE PROFILE
QUERY PROFILE
REPORT CONTINUOUS RECEIVE STATUS
START CONTINUOUS RECEIVE
STOP CONTINUOUS RECEIVE
```

MERGED

Indicates whether the transaction image is printed with a new line for each segment, and merged with the functional acknowledgment image. Valid values are:

Y   Prints each segment on a new line

N   Does not print each segment on a new line

This keyword is used only with the PRINT TRANSACTION IMAGE command.

MRREQID

A requestor ID used to associate management reporting statistics with DEENVELOPE processing instead of RECEIVE processing. You can use this keyword if you receive interchanges without using WebSphere Data Interchange, and you want to keep management reporting receive statistics on those interchanges. Do not use this keyword if management reporting statistics were created when the interchange was received (default). Use this keyword only in exceptional situations.

This keyword is used with the following commands§

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
```

MSGUCLASS

An override message user class for a send or receive type command. If you specify this keyword, its value overrides the value specified in the mailbox (requestor) profile member identified by the `REQID` keyword. The maximum length is eight.

This keyword is used with the following commands:

```
ENVELOPE AND SEND
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND SEND
RECEIVE AND TRANSLATE
RECONSTRUCT AND SEND
RECVFILE AND SEND
```

MSGUCLASS

```
REENVELOPE AND SEND
SEND
SENDFILE
TRANSLATE AND SEND
```

MULTIDOCS

Indicates whether the XML input file contains multiple documents. Valid values are:

Y   Multiple documents. The input message must be in EBCDIC format and each document must begin with an XML declaration (`<?xml`...).

N   One document (default)

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND TRANSLATE
```

NETACKP

Indicates whether to select transactions for which network acknowledgments are pending. Valid values are:

Y   Selects transactions for which a network acknowledgment was requested but not received

N   Selects transactions for which a network acknowledgment is not pending or was not requested

This keyword is used with the following commands:
```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
UNPURGE
```

NETID

The network ID as entered in the network profile. The maximum length is eight.

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
NETWORK ACTIVITY DATA EXTRACT
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
UPDATE STATUS
```

NETNAME

The name of the network as entered in the `Network name` field of the network profile. This field is case sensitive. The maximum length is 30.

This keyword is used only with the NETWORK ACTIVITY DATA EXTRACT command.

NETSTAT

Indicates the network status of a transaction for which a send has been requested. Valid values are:

30	Enveloped
31	Envelope error
41	Sent with errors
42	Send request error
43	Not sent net error
46	Send started
48	Send requested
49	Sent to network
50	Accepted by network
51	Delivered by network
52	Purged by network

53   Recall requested

54   Recall request error

55   Recalled

For complete status information, refer to the WebSphere Data Interchange for z/OS Administration Guide.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RECEIVE AND SEND
RECVFILE AND SEND
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
SEND
TRANSACTION DATA EXTRACT
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD
UNPURGE
```

NEWAPPLID

The application ID used to change to another application when loading event log entries into an event log. The maximum length is eight.

This keyword is used only with the LOAD LOG ENTRIES command.

NOMSG

Indicates whether extraneous messages should appear in the print file. Applies only to EXPORT and IMPORT commands, and is normally used when large numbers of records are being exported or imported. For example, it may be used with export-all (exporting after building a control file using program EDIXPEA) or import-all (using a control file with category 9).

Y   Suppresses the following messages from appearing in the print file:

`EI0048, EI0049, EI0062, FF0512, FF0514`

Also suppresses certain error conditions. For example, if in your command file you specify that you want to export CONTRECV members, and no Continuous Receive (CONTRECV) profile exists on your system, this error is ignored. Also, if you specify certain associated objects that do not exist, this error is also ignored.

N   Does not suppress error messages or conditions (default).

This keyword is used with the following commands:
```
EXPORT
IMPORT
```

NUMDELS

Specifies the number of database deletions performed before WebSphere Data Interchange issues a database commit. DB2 includes a site-dependent value (in `NUMLKUS`) that controls the maximum number of page locks that a single REMOVE TRANSACTIONS process can apply. When the REMOVE TRANSACTIONS process is not running standalone, page locks are obtained against the database. If the `NUMDELS` value is too high, DB2 may stop the REMOVE TRANSACTIONS process because the value in `NUMLKUS` has been exceeded. If this happens, use a smaller number in `NUMDELS` so that WebSphere Data Interchange issues database commits more frequently, which will release page locks more quickly. The default is **100** and the maximum is **1000**. If a value greater than 1000 is entered, the default of 100 is used.

This keyword is used with the following commands:
```
REMOVE LOG ENTRIES
REMOVE STATISTICS
REMOVE RANSACTIONS
RESET STATISTICS
UNLOAD LOG ENTRIES
```

NUMUPDTS

Specifies the number of database updates performed before WebSphere Data Interchange issues a database commit. If you are experiencing timeouts using a command that includes this parameter, you can reduce the amount of time that the command holds page locks by forcing more frequent COMMITs. The default for NUMUPDTS is **50**. If you want more frequent COMMITS performed, specify a number smaller than 50.

If you are trying to improve the performance of a command that includes this parameter, you can specify a value larger than 50. This forces WebSphere Data Interchange to COMMIT less frequently and should speed up processing.

This keyword is used only with the UPDATE STATISTICS command.

ONELOGICAPP

Indicates whether the `APPFILE` in the current WHERE clause and all `APPFILE`s in proceeding WHERE clauses are considered one logical file. When chosen, envelope breaks are avoided between `APPFILE` processing. You can use this keyword to avoid `APPFILE` switching that causes envelope breaks when processing multiple raw data files by placing multiple raw data files in the same envelope without delaying enveloping. If the `FILEID` keyword is specified in combination with this keyword in multiple WHERE clauses, the value in the first `FILEID` found is used to envelope the data. You can also use this option with C and D record formats. Valid values are:

Y   Treats multiple application files as one logical file

N   Processes each file independently; envelope breaks occur at the end of each application file (default)

This keyword is used with the following commands:

```
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

ONEMSG

Indicates whether all MQSeries messages are read from an MQSeries queue or only one message is read from the queue. The queue can be either a receive file or an application send file. Applies only to MQ. An MQSeries message is defined as a logical set of MQ records with the same MSGID. This keyword also controls MQ message descriptor propagation when you can set it on the TRANSLATE AND SEND and RECEIVE AND TRANSLATE commands where the application file and the send or receive file are both MQ queues. If you want MQ message descriptors to propagate during processing, set this keyword to **Y**.

Indicates whether all messages/documents in the output data file(s) are written as a single MQSeries queue message or if they are separated into different MQSeries queue messages by a PERFORM SEND/SENDFILE. Set this keyword to Y to direct WDI to add a separator between logical messages/documents and N(Default value) to not include separators. Because output data file(s) will have a separator record inserted when ONEMSG(Y) is used, it is not advisable to use it if the outfile will not be processed by a SEND/SENDFILE command.

When used in conjunction with ONEMSG(Y) in ″PERFORM TRANSFORM″ command, and when the value Y is set in this keyword, the logical messages/documents in the output file is written as different MQSeries queue messages. When the value is N the output file is written as a single MQSeries queue message. Any separator records are bypassed.

Valid values are:

Y   Apply ONEMSG Rules for Reads and Writes from an MQSeries queue

N   Do not Apply ONEMSG Rules for Reads and Writes from an MQSeries queue (default)

This keyword is used with the following commands:

```
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
SEND
SENDFILE
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD
```

OPTRECS

Indicates which optional records are created during translation. **I** is valid only for the TRANSLATE TO STANDARD command (delayed enveloping). **Q** is not valid for the TRANSLATE or RETRANSLATE TO APPLICATION commands. The maximum length is 5. Valid values are:

E   Envelope header

G   Group header

I   Information

Q   Queuing

T   Transaction header

For more information on these records, "Optional records" on page 292.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE
REENVELOPE AND SEND
RETRANSLATE TO APPLICATION
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

OUTFILE

The ddname of an output file (or the name of a TS queue or TD queue). For CICS, when you specify this keyword, you must also specify the `OUTTYPE` keyword. For the TRANSFORM command, the value in this field overrides any output file names generated during processing, such as the filename from the trading partner profile. The maximum length is eight.

This keyword is used with the following commands:

```
QUERY PROFILE
SAP STATUS EXTRACT
TRANSFORM
```

OUTFORMAT

Indicates the format in which the output is written. Valid values are:

F   Fixed

N   Native

T   Tagged

This keyword is used only with the QUERY PROFILE command.

OUTLEN

Specifies the maximum record length for the output data.

This keyword is used only with the TRANSFORM command.

OUTTYPE

Indicates the file type of `OUTFILE`. Applies only for CICS and MQ (which is supported in both CICS and z/OS). For CICS, when you specify the `OUTFILE` keyword, you must specify this keyword. Valid values are:

MQ   WebSphere Data Interchange MQSeries queue profile member name

TD   Transient data queue

TM   Temporary storage queue - main storage

TS   Temporary storage queue - auxiliary storage (default for CICS)

VS   VSAM data set

This keyword is used with the following commands:
```
QUERY PROFILE
SAP STATUS EXTRACT
TRANSFORM
```

For z/OS, if you do not specify this keyword, this field is ignored and the ddname of a sequential file is used. For CICS, the default is **TS**.

PAGE

Indicates whether pageable translation should be enabled. For more information about pageable translation, see "Pageable translation work file (EDIVAX)" on page 202, and "Pageable translation" on page 473.

Valid values for Pageable Translation are:

Y   Enables Pageable Translation

N   Disables Pageable Translation

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
RECONSTRUCT AND SEND
REENVELOPE
REENVELOPE AND SEND
RETRANSLATE TO APPLICATION
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

PRIORTO

Specifies the date before which all statistics will be deleted.

This keyword is used with the following commands:

```
REMOVE STATISTICS
RESET STATISTICS
SAP STATUS REMOVE
```

PURGINT

Specifies the number of days that a transaction remains in the Transaction Store before being marked for purging. If you do not specify this keyword and value, or if you specify a value of **0**, the WebSphere Data Interchange Utility uses **30** days as the default. The maximum value is **9999**. You can use a negative value to indicate that a transaction's store time expired on a date in the past. For example, if you specify PURGINT with a value of **-2**, the purge date is set at two days ago. The minimum value is **-999**. If a functional or network acknowledgment is pending when the store time expires, the transaction retains its current store status until the acknowledgment is no longer pending.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
RETRANSLATE TO APPLICATION
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

RAWDATA

Indicates whether you want the data translated from EDI standard format to raw format, or to C and D records. Valid values are:

Y   Raw data format

N   C and D record format

For details about these record formats, "Application file" on page 195. For an explanation of the file to which data will be written, "Envelope file" on page 196.

This keyword has this meaning when used with the following commands:
```
DEENVELOPE AND TRANSLATE
RECEIVE AND TRANSLATE
RETRANSLATE TO APPLICATION
SENDFILE
TRANSLATE TO APPLICATION
```

For Fixed-to-Fixed translation, this keyword indicates whether the output data should be written in raw data format or in the C and D record format. For a description of C and D records, "WebSphere Data Interchange Utility records format" on page 276

This keyword has this meaning when used with the following commands:
```
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND SEND
RECONSTRUCT
RECONSTRUCT AND SEND
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD
```

RAWFMTID

Applies only for raw data. Specifies the format ID of the application data file. The maximum length is 16.

This keyword is used with the following commands:
```
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD
```

RAWTEST

Applies only to raw data. Indicates to the receiver whether the transaction is for testing. Valid values are:

Y   Test transactions. All the transactions are test transactions, and a test usage/rule should be used, if it exists. If a test usage/rule does not exist, the production usage/rule should be used. Same as `RAWUSAGE`(**T**).

N   Production transactions (default). All the transactions are production transactions and only the production usage/rule should be used. Same as `RAWUSAGE`(**P**).

U   Either test or production transactions. WebSphere Data Interchange determines the status of the transaction based on the presence of an active

test usage/rule. If an active test usage/rule exists, the transaction is considered a test transaction. If an active test usage/rule does not exist, the transaction is considered a production transaction. Same as `RAWUSAGE`(**U**).

Note: This keyword is being replaced by `RAWUSAGE`. The `RAWUSAGE` value overrides `RAWTEST` if both values are specified. `RAWTEST` is provided for transitional purposes only, and will be removed in a future release.

The translator uses this value to set the test indicator in the interchange header (0035 for EDIFACT, I14 for X12). For C and D records, the `TESTIND` field of the control record serves the same purpose as this keyword.

This keyword is used with the following commands:
```
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD
```

RAWUSAGE

Applies only for raw data. Indicates to the receiver whether the transaction is for production, testing, or information. Valid values are:

P   Production transactions (default). All of the transactions are production transactions and only the production usage/rule should be used.

T   Test transactions. All of the transactions are test transactions, and a test usage/rule should be used. If a test usage/rule does not exist, the production usage/rule should be used.

I   Information transaction. An information usage/rule is used if one exists. If one does not exist, a production usage/rule should be used. If a production usage/rule does not exist, an error occurs.

U   Either test or production transactions. WebSphere Data Interchange determines the status of the transaction based on the presence of an active test usage/rule. If an active test usage/rule exists, the transaction is considered a test transaction. If an active test usage/rule does not exist, the transaction is considered a production transaction.

The translator uses this value to set the test indicator in the interchange header (0035 for EDIFACT, I14 for X12). For C and D records, the `TESTIND` field of the control record serves the same purpose as this keyword.

This keyword is used with the following commands:
```
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD
```

RECEIVEACKDATA

Indicates whether detailed receive acknowledgment data is written to the EDIQUERY file. Valid values are:

RECEIVEACKDATA

Y   Writes detailed acknowledgment data

N   Discards detailed acknowledgment data (default)

Detailed acknowledgment data includes interchange, group, and transaction data for each acknowledgment transaction. This information is concatenated with the group and transaction records to which it applies. For the format of these records, "Transaction/Acknowledgment image data extract record layout" on page 310.

This keyword is used with the following commands:
```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

RECEIVEACKIMAGE

Indicates whether the receive acknowledgment image is written to the EDIQUERY file. Valid values are:

Y   Writes receive acknowledgments

N   Discards receive acknowledgments (default)

Images are always written as separate records. For the format of these records, "Transaction/Acknowledgment image data extract record layout" on page 310.

This keyword is used with the following commands:
```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

RECOVBAD

Indicates whether the translation process will try to recover from a bad EDI standard data flag. Valid values are:

Y   The translation process tries to recover from bad EDI standard data. The translator checks for the standard interchange headers when a segment terminator is missing from a particular standard segment. The translator attempts to reset the delimiters and check the current segment/record for the segment terminator.

N   The translation process does not try to recover (default).

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
```

RECOVERY

Indicates the unit of work. When this keyword is used on the PERFORM TRANSFORM command, RECOVERY(E) will cause all transactions in an interchange to be discarded if any of the transactions fails. If RECOVERY(T) is specified, the successful

transactions from the interchange will be processed and only the invalid transactions will be discarded. The default is environmentally dependent. Valid values are:

E   Issues a database commit after each envelope (default for CICS)

T   Issues a database commit after each transaction (default for z/OS)

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

REQID

The requestor ID as entered in the mailbox (requestor) profile. Since each WHERE clause can contain only one requestor ID, you must add a WHERE clause for each requestor ID. A send is issued for each requestor ID for which data was queued. The maximum length is 16.

This keyword is used with the following commands:

```
CLOSE MAILBOX
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE AND SEND
NETWORK ACTIVITY DATA EXTRACT
PROCESS NETWORK ACKS
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND SEND
RECEIVE AND TRANSLATE
RECVFILE AND SEND
REENVELOPE AND SEND
SEND
SENDFILE
TRANSLATE AND SEND
UPDATE STATUS
```

REQTP

The trading partner associated with a given requestor ID for use with direct connection networks such as point-to-point. This keyword is used to eliminate the need to specify a separate requestor ID for each trading partner.

This keyword is used only with the ENVELOPE AND SEND command.

RESET

Indicates whether the output file is reset to receive new data or the new data is appended to the data already in the output file. Valid values are:

Y   Resets the output file

N   Appends data to the output file

This keyword is used with the following commands

```
GLB DUMP
GLB TRACE
```

SAPFILE

The ddname of an output file (or the name of a TS queue or TD queue). For CICS, when you specify this keyword, you must also specify the SAPTYPE keyword. For the TRANSFORM command the system will write the SAP status data to the file specified by the SAPFILE/SAPTYPE keywords for CICS or to the file specified by the SAPFILE keyword for other platforms. The system will also write the data to the DB; each record will be marked as extracted. The maximum length is eight.

This keyword is used with the following commands:

```
TRANSFORM
```

SAPTYPE

Indicates the file type of SAPFILE. Applies only for CICS and MQ (which is supported in both CICS and z/OS). For CICS, when you specify the SAPFILE keyword, you must specify this keyword. Valid values are:

MQ WebSphere Data Interchange MQSeries queue profile member name
TD Transient data queue
TM Temporary storage queue - main storage
TS Temporary storage queue - auxiliary storage (default for CICS)
VS VSAM data set

This keyword is used with the following commands:

```
TRANSFORM
```

SAPSTAT

Specifies the SAP status value to extract, or when used with the TO keyword, the range of values to extract. Valid values are 04-22. The default is all status.

This keyword is used with the following commands:

```
RECEIVE AND SEND
SAP STATUS EXTRACT
SAP STATUS REMOVE
```

SAPUPDT

Indicates whether SAP status tracking is desired. Valid values are:

Y   Tracks SAP status and writes status records

N   Does not track SAP status or write status records (default)

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND SEND
RECVFILE AND SEND
REENVELOPE
REENVELOPE AND SEND
SEND
TRANSFORM
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD
```

SCRIPT

Specifies the value that can be used by communication software to identify a set of instructions to follow when processing requests for services. This set of instructions would be part of the communication software package and not part of WebSphere Data Interchange. The maximum length is eight.

This keyword is used with the following commands:
```
ENVELOPE AND SEND
RECEIVE AND SEND
RECONSTRUCT AND SEND
RECVFILE AND SEND
REENVELOPE AND SEND
RESTART SEND
SEND
SENDFILE
TRANSLATE AND SEND
```

SEGMENTED

Indicates whether the image is printed with each segment starting on a new line. Valid values are:

Y   Starts a new line for each segment

N   Does not start a new line for each segment

This keyword is used with the following commands:
```
PRINT ACKNOWLEDGMENT IMAGE
PRINT TRANSACTION IMAGE
```

SENDACKDATA

Indicates whether detailed send acknowledgment data is written to the EDIQUERY file. Valid values are:

SENDACKDATA

Y   Writes detailed acknowledgment data

N   Discards detailed acknowledgment data (default)

Detailed acknowledgment data includes interchange, group, and transaction data for each acknowledgment transaction. This information is concatenated with the group and transaction record to which it applies. `This` keyword is ignored unless `GROUP` or `TRANSACTION` is set to **Y**. For the format of these records, see "Transaction/Acknowledgment image data extract record layout" on page 310.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

SENDACKIMAGE

Indicates whether the send acknowledgment image is written to the EDIQUERY file. Valid values are:

Y   Writes the generated acknowledgment record

N   Discards the generated acknowledgment record (default)

Images are always written as separate records. For the format of these records, see "Transaction/Acknowledgment image data extract record layout" on page 310.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

SEQNUM

Indicates whether network profile member sequence numbers should be incremented during send processing. This is an optional keyword. Specifying a value of **N** turns off network sequence numbers and saves associated overhead. Valid values are:

Y   Increments network profile member sequence numbers (default)

N   Does not increment network profile member sequence numbers.

This keyword is used with the following commands:

```
ENVELOPE AND SEND
REENVELOPE AND SEND
RESTART SEND
SEND
SENDFILE
TRANSLATE AND SEND
```

SERVICESEGVAL

Indicates at which level the service segments should be validated. The service segments are the segments used when a transaction is enveloped (ISA, GS, ST, UNB, UNH, UNT, and so on). If you do not specify this keyword, no validation occurs. Valid values are:

1	Validates the service segments for syntax only. This includes checking for mandatory data that is missing, as well as data elements that are too large or too small.
2	In addition to level 1 checking, validates the service segment date and time data elements according to their types, and if a validation table has been specified, also checks the value of the data element.

Validation errors that occur during send processing will terminate processing. Validation errors that occur during receive processing cause the interchange/group/transaction with the error to be skipped (not processed) but processing continues.

For Data Transformation processing the service segment validation is accomplished using WDI Client validation mapping. The validation maps are executed in addition to any inbound validation maps specified on the data transformation mapping rule. The keyword SERVICESEGVAL on the PERFORM TRANSFORM command will identify the level of service segment validation.

```
$WDI_E99AENV_VAL - UN/EDIFACT based on E99A service segments and code lists

$WDI_UCSENV_VAL - UCS based on UCS 4050 service segments and code lists

$WDI_X44ENV_VAL - X12 based on X12 4040 service segments and code lists
```

The validation map names cannot change with WDI 3.2 and will be selected based on the inbound interchange or envelope type. Although the shipped validation maps may be modified, it is recommended that customers who want to modify the mapping, make a copy of the WDI validation map and update the WDI mapping to add a DIMAPSWITCH command to switch to the copied validation mapping.

The following translation tables are used within the mapping:

```
$WDI_E99AENV_VAL
ETEXT00A - For functional acknowledgment error code translation to error
message text included with WDI error message UT0033

$WDI_UCSENV_VAL
A1DEC - For error code translation to functional acknowledgment error code
A1TEXT - For functional acknowledgment error code translation to error
message text included with WDI error message UT0033

$WDI_X44ENV_VAL
TA1INC - For error code translation to TA1 note code
TA1MT - For TA1 note code translation to error message text included with
WDI error message UT0033.
```

A new data transformation property SERVSEGVAL has also been added to be used with mapping commands GetProperty().

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
```

SERVICESEGVAL

```
REENVELOPE AND TRANSLATE
REENVELOPE
REENVELOPE AND SEND
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

SETCC

Specifies the condition codes used by the WebSphere Data Interchange Utility to override the utility condition codes specified in the IFCC keyword. You can have up to 10 override condition codes separated by commas. If you specify the IFCC keyword, you must specify this keyword. If this keyword is omitted, all condition codes specified on the IFCC keyword are overridden to zero (0). If a particular code is omitted in the SETCC keyword, the related condition code in the IFCC keyword is overridden to zero (0). For more information, see "Overriding utility condition codes" on page 23.

This keyword can be used with any utility PERFORM command.

SNDDATE

The date of the previous send request for the transaction. The maximum length is 10.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
UNPURGE
```

SNDTIME

The time of the previous send request for the transaction. The maximum length is eight.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
```

```
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
UNPURGE
```

STANDALONE

Indicates whether the Remove Transactions process should operate in contention with other WebSphere Data Interchange processes. In the DB2 environment, this allows WebSphere Data Interchange to obtain exclusive high-level table locks which improves performance of the Remove Transactions process since WebSphere Data Interchange does not have to obtain lower-level page locks. Valid values are:

Y The Remove Transactions process runs alone. Does not contend with any other WebSphere Data Interchange process.Run the Remove Transaction process in this mode if possible.

N The Remove Transactions process does not run alone. Contends with other WebSphere Data Interchange processes. High-level table locks are not obtained (default).

This keyword is used only with the REMOVE TRANSACTIONS command.

STDDESC

The description of the EDI standard as entered in the standards database. This field is case sensitive.

This keyword is used with the following commands:
```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

STDID

The EDI standard ID as entered in the standards database.

This keyword is used with the following commands:
```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

STDLV

The level of the EDI standard, for example, R1 for release 1.

This keyword is used with the following commands:
```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

STDTRID

The transaction ID of the EDI standard as specified by the standard, such as 850 for an X12 purchase order. The maximum length is eight.

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

STDVR

The version of the EDI standard, for example, V3 for version 3.

This keyword is used with the following commands

```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

STSTAT

Indicates the status of the transaction in the Transaction Store. Valid values are:

0   Active

1   Held

3   Purge-date expired

4   Purge-user request

For additional status information, refer to the WebSphere Data Interchange for z/OS Administration Guide.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
RELEASE
REMOVE TRANSACTIONS
TRANSACTION DATA EXTRACT
UNPURGE
```

SYNTAX

Specifies the syntax type for the input data. This is a required field. Valid values are:

D Application data

E EDI data

X XML data

This keyword is used only with the TRANSFORM command.

TESTMODE

Indicates whether the transactions are test, information, or production transactions. Valid values are:

Y Test or information transactions

N Production transactions

This keyword is used with the following commands:
```
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

TPID

The internal trading partner ID used by an application as entered in the trading partner send usage/rule. The maximum length is 35.

This keyword has this meaning with the following commands:
```
DELETE USAGE
ENVELOPE
ENVELOPE AND SEND
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
```

TPID

```
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

This keyword provides the default internal trading partner ID value, if the data format does not define a field that contains this value, or if a field is defined but contains all blanks. This also becomes the default ID value if the internal trading partner ID is blank in the C record.

This keyword has this meaning with the following commands:

```
TRANSLATE TO STANDARD
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
```

TPNICKN

The trading partner nickname for a trading partner profile member. The maximum length is 16. TPNICKN has two distinct uses:

- This keyword specifies a trading partner to receive data from, or a trading partner nickname to use when selecting Transaction Store or management reporting data.
- For the SEND and TRANSLATE AND SEND commands, this keyword is used the same as the keyword TPNICKNESEND. This means that the fields in the specified trading partner profile member are used for network override options (such as network charges), and will override the same fields in the mailbox (requestor) profile.

   Note: When this keyword is used with the SEND and TRANSLATE AND SEND commands to send EDI data, the receiver's mailbox is not specified because that information is contained within the data being sent.

This keyword is used with the following commands:

```
DELETE USAGE
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
```

```
PURGE
QUERY
RECEIVE
RECEIVE AND DEENVELOPE
RECEIVE AND SEND
RECONSTRUCT
RECONSTRUCT AND SEND
RECVFILE AND SEND
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
SEND
SENDFILE
TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

TPNICKNESEND

Specifies a trading partner profile member to use for network override options (such as network charges), that will override the same fields in the mailbox (requestor) profile.

Note: When this keyword is used with the SEND and TRANSLATE AND SEND commands to send EDI data, the receiver's mailbox is not specified because that information is contained within the data being sent.

This keyword is used with the following commands:

```
ENVELOPE AND SEND
REENVELOPE AND SEND
```

TRACELEVEL

Indicates the level of tracing done during the transform process. For z/OS, trace data will be written to ddname **EDIDTTRC**. For CICS, trace data will be written to the TD queue defined for EDI standard output. If required you can change the TD queue to a TS queue. For AIX and Windows platforms the trace data will be written to the file defined by the environment variable **EDIDTTRC**. You can set this using `export` command on AIX platforms or the `set` command on Windows platforms. For example:

```
export EDIDTTRC=trace.out
```

or

```
set EDIDTTRC=trace.out
```

The value consists of a series of C*n* values which represent the component and trace level for the component. The valid values for the component ID are:

A   All nodes

D   Deenveloper node

TRACELEVEL

E   Enveloper node

M   Message broker

P   Parsers

R   Rules node

T   Transformation node

V   Validation node

The valid values for the component tracing level are:

0   All trace messages are ignored.

1   Normal tracing. Only the first 256 byes of data in the buffer are written to the trace file.

2   Extended tracing. The entire contents of the buffer is written to the trace file.

3   Utility function tracing. Includes all the tracing done at level **2** plus additional tracing for some frequently called internal utility functions.

For example, a value of D1 V2 R2 would mean the deenveloper node (**D**) is set for normal tracing (**1**), the validation (**V**) and rules (**R**) nodes are set for extended tracing (**2**). Tracing is normally turned off except during problem determination. Activating tracing may negatively impact performance.

This keyword is used only with the **TRANSFORM** command.

TRANSACTION

Indicates whether the transaction data record is written to the EDIQUERY file. Valid values are:

Y   Writes transaction data records

N   Discards transaction data records (default)

For the format of these records, see "Transaction/Acknowledgment image data extract record layout" on page 310.

This keyword is used with the following commands:
```
ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT
```

TRERLVL

Indicates the maximum translation error level for the transactions you want to select. You can use this keyword to envelope only EDI documents that are error-free. Valid values are:

0   No errors

1   Data element errors

2   Data element and segment errors

3   Severe errors

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
UNPURGE
```

TRKFILE

Allows a tracking file to be used. If you specify **TRKFILE(Y)** and specify a tracking file in your command file your optional records will go into that tracking file instead of the application file.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND SEND
```

TRXCTLNO

The transaction set control number assigned by the sender to identify the transaction set to the sender. When combined with the sender ID, it identifies the transaction set to the receiver. The maximum length is 14.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
```

TRXCTLNO

```
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

TRXDATE

The date the transaction was added to the Transaction Store. This is also the date on which the transaction was translated to EDI standard format. The maximum length is 10.

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

TRXSTAT

Indicates the processing status of the transaction.

For ENVELOPE AND SEND, the default value is **21**. Valid value is:

21   Send translated

For REENVELOPE, the default values are **31**, **41**, **42**, and **43**. Valid values are:

29   Transaction detached -- send

30   Enveloped

31   Envelope error

41   Sent with errors

42	Send request error
43	Not sent network error
46	Send started status
48	Send requested
49	Sent to network
50	Accepted by network
51	Delivered by network
52	Purged by network
53	Recall requested
54	Recall request error
55	Recalled
61	Transaction accepted
62	Transaction rejected
63	Transaction accepted with errors

For TRANSLATE, the default value is **70**. For RETRANSLATE TO APPLICATION, the default value is **73**. Valid values for both are:

70	Received
72	Receive translated (RETRANSLATE TO APPLICATION only)
73	Receive translate error (RETRANSLATE TO APPLICATION only)

For UNPURGE TRANSACTIONS and REMOVE TRANSACTIONS, if no value is specified, all values are included in the selection criteria. Valid values are:

20	Send translate error
21	Send translated
29	Transaction detached -- send
30	Enveloped
31	Envelope error
41	Sent with errors
42	Send request error
43	Not sent network error
46	Send started status
48	Send requested
49	Sent to network
50	Accepted by network

51	Delivered by network
52	Purged by network
53	Recall requested
54	Recall request error
55	Recalled
61	Transaction accepted
62	Transaction rejected
63	Transaction accepted with errors
70	Received
71	Receive syntax error
72	Receive translated
73	Receive transaction error
74	Transaction detached -- recv

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

TRXTIME

The time the transaction was added to the Transaction Store. This is also the time at which the transaction was translated to EDI standard format. The maximum length is eight.

This keyword is used with the following commands:

```
ENVELOPE
ENVELOPE AND SEND
ENVELOPE DATA EXTRACT
HOLD
PRINT ACKNOWLEDGMENT IMAGE
PRINT ACTIVITY SUMMARY
PRINT EVENT LOG
PRINT STATUS SUMMARY
PRINT STATUS SUMMARY2
PRINT TRANSACTION DETAILS
PRINT TRANSACTION IMAGE
PURGE
QUERY
REENVELOPE
REENVELOPE AND SEND
RELEASE
REMOVE TRANSACTIONS
RETRANSLATE TO APPLICATION
TRANSACTION DATA EXTRACT
TRANSLATE TO APPLICATION
UNPURGE
```

USERID

The network user ID of the trading partner as entered in the `User ID` field of the trading partner profile. The maximum length is 32.

This keyword is used with the following commands:

```
NETWORK ACTIVITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
```

USERPGM

Specifies the program that WebSphere Data Interchange links to just before writing a record during data extract processing. Your program should return a code to WebSphere Data Interchange indicating whether the record is written to the EDIQUERY file or discarded. If you do not specify this keyword, WebSphere Data Interchange writes the requested records to the EDIQUERY file. For more information, see Chapter 1, "Using The Utility," on page 1 The maximum length is eight.

This keyword is used with the following commands:

```
ENVELOPE DATA EXTRACT
NETWORK ACTIVITY DATA EXTRACT
TRANSACTION DATA EXTRACT
TRADING PARTNER CAPABILITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
```

VERIFY

Indicates whether the status of a transaction is verified before the transaction is put into an envelope. Valid values are:

Y   Checks for correct status before an ENVELOPE or REENVELOPE operation

VERIFY

(other)

> Does not check status (default)

This keyword is used with the following commands:
```
ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND
```

VERSION

The version specified on the send or receive map usage. The maximum length is 8. Default is blank.

This keyword is used with the following commands:
```
DELETE USAGE
```

WRTCTLNO

Indicates whether trading partner control numbers are included when importing a trading partner profile member. Valid values are:

Y   Imports control numbers with the trading partner (default).

N   Does not import control numbers. If this is a new trading partner, the control number defaults to zeros. If this trading partner already exists, the existing control numbers are not overwritten by the control numbers in the import file.

This keyword is used only with the IMPORT command.

XML

Indicates whether XML processing is required for the input data. Valid values are:

Y   Requires XML processing

N   Does not require XML processing (default)

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
TRANSLATE AND ENVELOPE
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

XMLDICT

Identifies the PDS or HFS path for the XML dictionary files generated by the DTD conversion utility. This field is required for XML processing. The maximum length is 64.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
TRANSLATE AND ENVELOPE
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

XMLDTDS

Identifies the PDS or HFS path for the XML DTD members. This field is required for XML DTD processing. The maximum length is 64.

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
TRANSFORM
TRANSLATE AND ENVELOPE
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

For more information about XML DTD resolution, see "XML special considerations" on page 625.

XMLEBCDIC

Indicates whether the incoming XML data should be interpreted as EBCDIC data. Applies only to z/OS. Valid values are:

Y　　　　Interprets incoming XML data as EBCDIC data, regardless of the encoding on the XML declaration

N　　　　Uses the encoding on the XML declaration

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
TRANSLATE TO APPLICATION
TRANSFORM
```

XMLNS

Indicates if namespace processing for input XML data will be performed. If you are doing schema validation, or if the input document is based on an XML schema that uses namespaces, you should set XMLNS(Y). Default is N.

Y　　　　Do namespace processing for input XML data.

N　　　　Do not do namespace processing for input XML data. This is the default.

This keyword is used with the following commands:

```
TRANSFORM
```

XMLSCHEMAVAL

Indicates if validation of the data against an XML schemal will be performed. The XML schema to validate against should be in the directory specified by the XMLDTDS keyword.

Y   Validate the data against an XML schema.

N   Do not validate the data against an XML schema. This is the default.

A   Validate the data against an XML schema only if one is specified. If no schema is specified, do not attempt to validate against a schema.

This keyword is used with the following commands:
```
TRANSFORM
```

XMLSEGINP

Indicates whether record boundaries in the input file should be treated as line breaks. This keyword is ignored when XMLEBCDIC equals **N**.

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
TRANSLATE AND ENVELOPE
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

XMLSTDID

The EDI standard ID created with the DTD conversion utility. If you specify this keyword, this standard ID is used to convert each XML document to a corresponding EDI standard envelope and transaction. If you specify this keyword, the first eight alphanumeric characters of the root element name are used as the standard ID for each XML document. The maximum length is eight.

This keyword is used with the following commands:
```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
TRANSLATE AND ENVELOPE
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

XMLVALIDATE

Indicates the level of validation applied to the incoming XML data. Valid values are:

0   Ignores external DTD references

1   Uses the declared DTD for processing default attributes, entity references, and so on, but does not validate the data against the DTD (default)

2   Fully validates the data against the DTD

This keyword is used with the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
TRANSLATE TO APPLICATION
TRANSFORM
```

XMLSPLIT

For data transformation and XML source message processing. Overrides the XML DTD or Schema definition retrieval and processes the XML input message using default processing. Default is (Y).

This keyword is used with the following commands:

```
TRANSFORM (using SYNTAX(X))
```

XRECS

For data transformation and application data C&D data format as the target message. Indicates that XRECS should not be created in the output file for records exceeding 32000 bytes in length. Default is (Y).

This keyword is used with the following commands:

```
TRANSFORM
```

Chapter 3. File formats and WebSphere Data Interchange Utility records

This chapter describes the files used by the WebSphere Data Interchange Utility and the format of data within each file. "Using sample JCL" on page 407 also contains information about the WebSphere Data Interchange Utility files, and "Required utility data sets" on page 421 shows the required files for each PERFORM command.

Dynamically allocated application files

If no application file is specified within a command file, WebSphere Data Interchange will select a unique file name for the application file and assign this unique physical name to the logical file name specified on the PERFORM command. The new file will be located in the default temporary directory (/tmp for AIX and a user specified directory on Windows) unless the TMPDIR environment is set to a valid directory.

Transaction Store input and output files

This section describes the input and output files used by the WebSphere Data Interchange Utility. To indicate to WebSphere Data Interchange that a record is a comment, place an asterisk (*) in column one of a record.

Command file (EDISYSIN or SYSIN)

The command file contains the input WebSphere Data Interchange Utility commands that you want executed. The command language syntax is fairly free-form and, except for values associated with the following keywords, is not case sensitive:

- ACFIELD
- ADDRLN1
- ADDRLN2
- CMMTLN1
- CMMTLN2
- CMPYNM
- CNCTNM
- CNCTPH
- NETNAME
- STDDESC

For information about PERFORM commands, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

The WebSphere Data Interchange Utility first verifies that EDISYSIN is allocated. If so, logical name EDISYSIN is used. If EDISYSIN does not exist, logical name SYSIN is used. Prior to Release 4 of WebSphere Data Interchange, the command file was always allocated to logical name SYSIN. The command file is opened for read processing only, so it can be allocated as an inline data set in your WebSphere Data Interchange Utility JCL. File specifications are:

Transaction Store files

Use Input file containing WebSphere Data Interchange Utility PERFORM
commands.

Specified
Does not apply.

logical name
EDISYSIN or SYSIN. EDISYSIN takes precedence and is used if allocated.
You can override these logical names by specifying a WebSphere Data
Interchange MQSeries queue profile member to use with the `MQSYSIN`
parameter.

In CICS, the command file name and type are specified in the WebSphere
Data Interchange Utility control information. For detailed information, see
"WebSphere Data Interchange Utility control information" on page 368.

Suggested format
Record format: Fixed or variable. Record length: 80 bytes, but any length is
acceptable.

Remarks
The processing of this file adjusts to the file attributes. You can define this file
to best fit your requirements.

WebSphere Data Interchange DB2 command file (EDITSIN)

This optional file is used to contain keywords that determine whether WebSphere Data
Interchange should attach and/or detach DB2. For more information, see "WebSphere
Data Interchange and DB2 attachment" on page 428. This file applies only to z/OS DB2
installations; it does not apply to CICS installations. File specifications are:

Use Input file containing information that tells the WebSphere Data Interchange
Utility whether or not to attach and/or detach DB2.

Specified
Does not apply.

logical name
EDITSIN.

Suggested format
Record format: Fixed or variable. Record length: 80 bytes

Remarks
Typically, this is an in-stream JCL file that you can define to best meet your
requirements.You can invoke the WebSphere Data Interchange Utility using
EXEC PGM=EDIFFUT or EXEC PGM=IKJEFT01. For the most part, if EXEC
PGM=EDIFFUT is used, EDITSIN will contain the DB2 control information. If
EXEC PGM=IKJEFT01 is used, SYSTSIN will contain the DB2 control
information.

Example 1
The following example uses DB2 command file EXEC PGM=EDIFFUT.

```
//RUNDI  EXEC PGM=EDIFFUT,DYNAMNBR=20,REGION=6144K
//    PARM='SYSID=DIENU APPLID=EDIFFS LANGID=ENU'
//EDITSIN DD *
SYSTEM(DB93) PLAN(EDIENU32) OPEN(Y) CLOSE(Y) CAF(Y)
```

Example 2

The following example uses DB2 command file EXEC PGM=IKJEFT01.

```
//RUNDI  EXEC PGM=IKJEFT01,DYNAMNBR=20,REGION=6144K
//SYSTSIN DD *
DSN SYSTEM(DB93)  RUN PROG(EDIFFUT)    PARM('SYSID=DIENU APPLID=EDIFFS
LANGID=ENU SYSTEM=DB93 PLAN=EDIENU32')  PLAN(EDIENU32)  END
/*
```

Network commands file (NETOP)

The network commands file contains the commands that you want WebSphere Data Interchange to pass to the network. WebSphere Data Interchange reads the commands from a member of this partitioned data set (PDS) and writes the commands to the Network input file specified in the network profile member. This interface to networks can be used instead of using various Network Commands (NETOP) profile member commands; however, not all networks are supported through this interface. File specifications are:

Use   Holds the network commands that the user wants WebSphere Data Interchange to pass to the network.

Specified
Specific members are specified in the `Network cmds file` field of the trading partner or mailbox (requestor) profiles.

logical name
EDINTCMD.

Suggested format
Record format: Fixed or variable. Record length: 80 bytes.

Remarks
Network commands can contain variables that are resolved by WebSphere Data Interchange before the commands are passed to the network. If the records exceed the file record length, they will be truncated.

Application file

The application file contains the input records that WebSphere Data Interchange uses to translate application data into an EDI standard format during the send process, and the output records WebSphere Data Interchange creates when translating data from an EDI standard format to an application format during the receive process. For translating to an application format, this file must be able to handle the largest data record you expect to receive and the largest information record the WebSphere Data Interchange Utility might return. If you request raw data records, the WebSphere Data Interchange Utility writes the information and other optional records to the exception file (FFSEXCP). The WebSphere Data Interchange Utility opens the file for output when processing the

first received transaction and opens the file again to append the data (EXTEND). Use the JCL DISP options to control whether the file is cleared or appended to during the first use.

When translating to an EDI standard format, the value of the `RAWFMTID` keyword indicates if the file contains raw data records. If `RAWFMTID` is omitted, the file is expected to contain C and D records. When translating to an application format, both the data format and the value of the `RAWDATA` keyword indicate whether the file is written in C and D record format or raw data record format. If the `RAWDATA` keyword is set to **Y** and the data format does not have a record ID position specified, C and D records are written. File specifications are:

Use   Input file for translating to an EDI standard; output file for translating to application.

Specified
Sending: APPFILE keyword.

Receiving: Data format setting on the Transaction Usage Override panel.

logical name
User-defined.

Suggested format
Record format: Fixed or variable. Record length:

- For raw data, use maximum structure size.
- For C and D records, use maximum structure size plus 17 bytes, or 1024 bytes, whichever is greater.
- For I records, use 483 bytes.

Remarks
Records are truncated if record length is not large enough.

Envelope file

For outbound documents, the envelope file is either the file specified as the `Trans data queue` in the network profile member, or an override file specified in a command that requests enveloping. For inbound documents, this is either the `Receive file name` specified in the mailbox (requestor) profile member, or an override file specified in a command that requests receiving. File specifications are:

Use   Sending: holds complete envelopes when enveloping takes place.

Receiving: contains envelopes to be deenveloped and/or translated.

Specified
Sending: TD queue in network profile or overridden by the `FILEID` specified on the command.

Receiving: receive file in mailbox (req7uestor) profile or overridden by the `FILEID` specified on the command.

logical name
User-defined. Default value of **QDATA**, **QDATAE**, or **QDATAU** during enveloping.

Suggested format

Record format: fixed or variable. Record length: 80 bytes or greater.

Remarks

For enveloping on point-to-point networks, the envelope file is dynamically allocated with the DS name constructed of the current user ID and trading partner nickname.

Which envelope file is used for fixed-to-fixed translations is based on the value in the `Standard ID` field. For fixed-to-fixed translations with a target data format, the standard ID is the same as the `Application file name` in the data format. The `File suffix` field in the trading partner profile (TPPROF) member is used as a suffix to the standard ID to create a unique envelope file for each trading partner. Data can be written to these files in either C and D record format or a raw data format based on the `RAWDATA` keyword setting used in the PERFORM command. "WebSphere Data Interchange Utility records format" on page 276 for a description of C and D records and the RAWDATA format.

Note: You can override the envelope file name created by the above concatenation by specifying the `FIXEDFILEID` keyword in the PERFORM command.

Exception file (FFSEXCP)

When translating to application format, WebSphere Data Interchange will write translated transactions to the exception file if it cannot open the file intended to receive them, or if a file name is not provided. When translating to EDI standard format, WebSphere Data Interchange will write the transactions to this file that were not translated successfully. Optional records are also written to this file if the tracking file (FFSTRAK) does not exist.

This file must be large enough to contain the largest data record you are sending or receiving, and the largest information record the translator might return. WebSphere Data Interchange opens the file for output when processing the first transaction and then opens the file for EXTEND. Use the JCL DISP options to control whether the file is cleared or appended to during the first use. File specifications are:

Use   Sending: Holds transactions that were not translated successfully and the unidentified and optional records. For more information, see "Tracking file (FFSTRAK)" on page 198

Receiving: Holds transactions that could not be written to the application file, and optional records if the application file contains raw data. For more information, see "Optional records" on page 292

Specified

Does not apply.

logical name

FFSEXCP. FFSEXCP takes precedence and is used if allocated. You can override this logical name and use an MQSeries queue instead. You can use the MQEXCP parameter to specify a WebSphere Data Interchange MQSeries queue profile member to use instead of a sequential file.

Transaction Store files

In CICS, the exception file name and type are specified in the WebSphere Data Interchange Utility control information. For more information, see "WebSphere Data Interchange Utility control information field descriptions" on page 370

Suggested format

Record format: Fixed or variable. Record length:

- For raw data, use maximum structure size
- For C and D records, use maximum structure size plus 17 bytes, or 1024 bytes, whichever is greater
- For I records, use 483 bytes

Remarks

Records are truncated if record length is not large enough.

Tracking file (FFSTRAK)

When translating to EDI standard format, WebSphere Data Interchange writes the optional records to the tracking file if it exists. This file must be large enough to contain the largest information record that WebSphere Data Interchange might return. File specifications are:

Use   Sending: Holds optional records. For more information, see "Optional records" on page 292

Receiving: Does not apply.

Specified

Does not apply.

logical name

FFSTRAK (optional). FFSTRAK takes precedence and is used if allocated. You can override this logical name and use an MQSeries queue instead. You can specify a WebSphere Data Interchange MQSeries queue profile member to use instead of a sequential file with the MQTRAK parameter.

In CICS, the tracking file name and type are specified in the WebSphere Data Interchange Utility control information. "WebSphere Data Interchange Utility records format" on page 276.

Suggested format

Record format: Fixed or variable. Record length:
- For C and D records, use 1024 bytes.
- For I records, use 483 bytes.

Remarks

Records are truncated if the record length is not large enough. If this file is not supplied and your application is using C and D records, WebSphere Data Interchange writes the optional records to the exception file (FFSEXCP). You cannot mix optional records with raw data in the exception file, so if `this file` is not supplied and your application uses raw data, WebSphere Data Interchange does not write the optional records.

Print file (PRTFILE)

The print file must allow for a minimum record size of 132 bytes. WebSphere Data Interchange opens this file for output. Use the JCL DISP options to control whether the file is cleared or appended to. File specifications are:

Use   Contains an audit report from the WebSphere Data Interchange Utility showing the results of processing.

Specified
Does not apply.

logical name
PRTFILE. PRTFILE takes precedence and is used if allocated. You can override this logical name and use an MQSeries queue instead. You can specify a WebSphere Data Interchange MQSeries queue profile member to use instead of a sequential file with the MQPRT parameter.

In CICS, the print file name and type are specified in the WebSphere Data Interchange Utility control information. For more information, see "WebSphere Data Interchange Utility records format" on page 276.

Suggested format
Record format: FBA or VBA. Record length: 132 bytes.

Remarks

Status of processing and any errors WebSphere Data Interchange encounters are written to this file during WebSphere Data Interchange Utility processing.

Figure 14 on page 200 shows a sample PRTFILE.

```
Audit Trail Report  -WebSphere Data Interchange Utility-  Date: 01/10/13  Time: 11:10:22
Interchange Control Number = 00000000000057
FF0010 Transaction number 1 to 1 translated successfully
FF0013 Transactions with Interchange Control Number 00000000000057 were successfully queued
Message: TR0004 Severity: 04
Code in ID type field not found in validation table.
Internal Trading Partner ID and Application Format = DLGTYPES - DLGTYPES.
Transaction handle, code, mode, and function = 20011013111038000000 - DL1 - PRODUCTION - SEND.
Interchange, group, and transaction control numbers = 000000056  - 56 - 0155.
Current Loop-ID and repetitions = 110000 - 1 - 1.
Standard segment and field ID = SEG3(003) - 3 - 1 - 11. Application field ID = SEG3 - 61.
Data type and value = Z -123456. Validation table name = DLGP2V.
Interchange Control Number = 00000000000056
FF0011 Transaction number 2 translated with errors
Message: TR0004 Severity: 04
Code in ID type field not found in validation table. Internal
Trading Partner ID and Application Format = DLGTYPES - DLGTYPES.
Transaction handle, code, mode, and function = 20011013111039000000 - DL1 - PRODUCTION - SEND.
Interchange, group, and transaction control numbers = 000000056 - 56 - 0156.
Current Loop-ID and repetitions = 110000 - 1 - 1.
Standard segment and field ID = SEG3(003) - 3 - 1 - 11. Application field ID = SEG3 - 61.
Data type and value = Z -123456. Validation table name = DLGP2V.
```

Figure 14. Sample PRTFILE

Report file (RPTFILE)

WebSphere Data Interchange opens the report file for output for the first report and as EXTEND for successive reports. Use the JCL DISP options to control whether the file is cleared or appended to during the first use. File specifications are:

Use   Contains reports requested during Transaction Store processing.

Specified
Does not apply.

logical name
RPTFILE. RPTFILE takes precedence and is used if allocated. You can override this logical name and use an MQSeries queue instead. You can specify a WebSphere Data Interchange MQSeries queue profile member to use instead of a sequential file with the MQRPT parameter.

In CICS, the report file name and type are specified in the WebSphere Data Interchange Utility control information. For more information, see "WebSphere Data Interchange Utility records format" on page 276.

Suggested format
Record format: FBA or VBA. Record length: 132 bytes.

Remarks
Reports requested from the Transaction Store Facility or WebSphere Data Interchange Utility are written to this file.

Query file (EDIQUERY)

The query file is opened for output for the first PERFORM command executed and opened for extend for all PERFORM commands issued thereafter during a single execution of the WebSphere Data Interchange Utility. Use the JCL DISP options to control whether the file is cleared or appended to during the first use. The query file is the output file for the commands shown in Table 6.

Table 6. EDIQUERY file layout for commands

PERFORM command	Record layout on page:
QUERY	68
ENVELOPE DATA EXTRACT	303
NETWORK ACTIVITY DATA EXTRACT	297
TRADING PARTNER CAPABILITY DATA EXTRACT	297
TRADING PARTNER PROFILE DATA EXTRACT	297
TRANSACTION ACTIVITY DATA EXTRACT	297
TRANSACTION DATA EXTRACT	303

File specifications are:

Use Output file for the QUERY command and the data extract commands listed above.

Specified
Does not apply.

logical name
EDIQUERY (required for commands that generate output). You can override this logical name and use an MQSeries queue instead. You can use the MQQUERY parameter specify a WebSphere Data Interchange MQSeries queue profile member to be used instead of a sequential file.

In CICS, the query file name and type are specified in the WebSphere Data Interchange Utility control information. For more information, see "WebSphere Data Interchange Utility records format" on page 276.

Suggested format
Record format: Fixed or variable. Record length: 32756 to ensure the largest expected record is not truncated.

Remarks
This file is used for the output of many different types of records. Therefore, you should allocate the file as variable length record format with a maximum record length of 32756. If you use this file for a specific set of records that do not require this maximum record length, you can allocate the file to meet your requirements. Records are truncated if the record length supplied is not large enough.

Transaction Store files

Work file (FFSWORK)

The work file is an internal work file used by WebSphere Data Interchange during send and translation processing. WebSphere Data Interchange opens this file for output only, and it should always be empty.

Note: In the CICS environment, this file is handled internally by WebSphere Data Interchange.

File specifications are:

Use Sending: Holds the current transaction for transfer to the exception file if translation is not successful.

Receiving: Does not apply.

Specified
Does not apply.

logical name
FFSWORK (required).

External specification is not made in CICS. WebSphere Data Interchange takes care of this file internally.

Suggested format
Record format: Variable blocked (VB). Record length: 32756 bytes.

Remarks
A temporary file used only during send and translation processing. This file is used extensively as a temporary file during translate-to-EDI-standard operations and is a prime candidate for a virtual input/output data set.

Pageable translation work file (EDIVAX)

The pageable translation work file is a temporary work file used by WebSphere Data Interchange when pageable translation is enabled and virtual storage usage for EDI or application data reaches 28 MB. Pageable translation is enabled by using the PAGE keyword on Utility PERFORM commands and, for API applications, by setting the VAXFLAG field in the TRCB to **X**. Enabling pageable translation will ensure that the virtual storage used for EDI and application data does not exceed 28 MB by paging any excess data to the EDIVAX file.

The amount of space allocated to this file depends on the maximum amount of data to be translated. You can calculate the amount of needed space by adding the following four values together:
1. Number of bytes in largest interchange
2. Number of bytes in largest application transaction image
3. 4 MB overhead
4. Number of structures in largest interchange multiplied by 120 bytes

The number of structures in the largest interchange includes structures that are passed separately (records) and substructures that are not passed separately but which contain data during translation. Pageable translation deals with the first two components, and

ensures that the amount of virtual storage required for them does not exceed 28 MB. The other components are not addressed by pageable translation. The maximum amount of data that WebSphere Data Interchange can page with pageable translation is approximately five gigabytes (specifically, 150,000 multiplied by 28,632 bytes). File specifications are:

Use Holds the paged EDI or application data when the virtual storage used to hold this data reaches 28 MB.

Specified
 Does not apply.

logical name
 EDIVAX (required for pageable translation).

Suggested format
 DCB statement should not be specified. (This is under WebSphere Data Interchange control.)

Remarks
 Omit the data set name to allow z/OS to assign a temporary data set name.

Enveloping options file for functional acknowledgments (FAENV)

The enveloping options file is an optional QSAM file that you can use if you need more flexibility in the enveloping of functional acknowledgments.

Note: For this file, a functional acknowledgment is an ANSI X12 997 transaction, a UCS 999 transaction, or an EDIFACT CONTRL message.

With this file you can specify:
- Interchange and group envelope overrides for functional acknowledgments
- A standard profile member to fill envelope data elements other than the sender and receiver IDs

To use the file, specify logical name FAENV in the JCL for any request that includes the DEENVELOPE, RECEIVE AND DEENVELOPE, or RECEIVE AND TRANSLATE commands.

File specifications are:

Use Sending: Does not apply.

 Receiving: Provides flexibility in determining the data used in the enveloping segments when functional acknowledgments are generated.

Specified
 Does not apply.

logical name
 FAENV (optional) for z/OS. EDIFAENV (optional) for CICS.

Suggested format

Record format: Fixed or variable. Record length: As large as the longest record in the file.

Remarks

An optional file used only during deenveloping to control the data used to build the enveloping (service) segments, such as ISA and GS segments, for the functional acknowledgment being returned to your trading partner.

The search key for an entry is ISID, IRID, GSID, and GRID, corresponding to the inbound envelope that is being received or deenveloped. The remaining fields of the entry specify the overrides you want to use for the outbound envelope containing the functional acknowledgments. If the FAENV file exists, WebSphere Data Interchange searches it for a matching entry. You do not have to include all of the key values in an entry or any overrides. If you do not specify overrides, an envelope profile member name is expected. You can use overrides in conjunction with a profile member. The applicable fields taken from the profile member are the same as those normally used during translation to send. If the member name and overrides are both present, the overrides are used. Entries on the right side of the equal sign override entries from the envelope profile member. For more information, refer to the WebSphere Data Interchange User's Guide.

FAENV field descriptions

The fields listed in Table 7 are optional.

Table 7. Fields in the Enveloping Options File for Functional Acknowledgments

Field	Maximum length	Description
ISID	35	Interchange sender ID from the inbound envelope
IRID	35	Interchange receiver ID from the inbound envelope
GSID	35	Group sender ID from the inbound envelope
GRID	35	Group receiver ID from the inbound envelope
ISIDFA	35	Interchange sender ID override for outbound functional acknowledgment envelope
IRIDFA	35	Interchange receiver ID override for outbound functional acknowledgment envelope
GSIDFA	35	Group sender ID override for outbound functional acknowledgment envelope
GRIDFA	35	Group receiver ID override for outbound functional acknowledgment envelope
EPM	8	Standard envelope profile member name

FAENV file format

The following is the format of an entry in the FAENV file:

```
ISID,IRID,GSID,GRID = ISIDFA,IRIDFA,GSIDFA,GRIDFA>EPM
```

Where:

This character:
      Separates:

, (comma)

      Envelope fields and indicates that a value is not listed

= (equals)
      Key fields from override fields

> (greater than)
      Override fields from an envelope profile name

Sample entries

Each of the following is a valid entry in FAENV:

```
ISID1,IRID1,=ISID1X,IRID1X,GSID1X,GRID1X

ISID2,,GSID2=ISID2X>EPM2

,IRID3,GSID3,GRID3=,,,GRID3X>EPM3

ISID4,IRID4,,GRID4=ISID4X,,,GRID4X>EPM4

ISID5,IRID5,GSID5=ISID5X,IRID5X,,GRID5X>EPM5

ISID6,IRID6=ISID6,IRID6X,GSID6X>PM6

ISID7=>PM7
```

A practical example

Two inbound X12 interchanges and one EDIFACT interchange contain transactions: these are defined in Table 8.

Table 8. Interchange example

Interchange	Transaction and Value
Interchange 1 (X12):	ISA06 (ISID) TPDUNS# ISA08 (IRID) MYDUNS#DIV1 GS02 (GSID) PDUNS# GS03 (GRID) MYDUNS#DIV1
Interchange 2 (X12):	ISA06 (ISID) TPDUNS# ISA08 (IRID) MYDUNS#DIV2 GS02 (GSID) TPDUNS# GS03 (GRID) YDUNS#DIV2

Table 8. Interchange example (continued)

Interchange	Transaction and Value
Interchange 3 (EDIFACT):	UNB03 (ISID) ACCX ACCX01 UNB06 (IRID) ACCY ACCY03 UNG02 (GSID) ACCOUNTING UNG04 (GRID) BOOKING

Without the FAENV file, the interchange and group envelopes created for the functional acknowledgments would be the same for Interchange 1 as for Interchange 2. In addition, if they are deenveloped one after the other, both functional acknowledgments are placed in the same outbound group and interchange envelopes. This happens because there is no distinction made between the different inbound interchange receiver IDs. The envelopes for the outbound acknowledgments are produced using the FA interchange values defined in Table 9.

Table 9. Envelopes for FA interchange

Envelope	Description
ISA06 or UNB03	First envelope profile member value
ISA08 or UNB06	Account number and user ID from trading partner profile member value
GS02 or UNG02	First envelope profile member value
GS03 or UNG04	First envelope profile member value

The first envelope profile member is obtained from the receive usage/rule of the first transaction set in the interchange. If both inbound interchanges are processed one after the other, the first envelope profile member is the one obtained for the first transaction set for the first interchange.

To distinguish between sender IDs for outbound functional acknowledgments, change the FAENV file to modify the normal procedure.

For the previous examples, the following entries must be present in FAENV:

```
TPDUNS#,MYDUNS#DIV1=MYDUNS#DIV1,TPDUNS#,MYDUNS#DIV1,TPDUNS#
TPDUNS#,MYDUNS#DIV2=MYDUNS#DIV2,TPDUNS#,MYDUNS#DIV2,TPDUNS#
ACCX ACCX01,ACCY ACCY03,ACCOUNTING=ACCY ACCY03,ACCX ACCX01,BOOKING,ACCOUNTING
```

The key specifies only the ISID and IRID. A key value is not required. Therefore, any GSID and GRID values meet the conditions, and the overrides are used. As a result, the functional acknowledgments that are produced have the separate envelopes shown in Table 10 on page 208

Transaction Store files

Table 10. Interchange example acknowledgements

Interchange	Transaction and Value
Interchange 1 (X12):	ISA06 (ISID) MYDUNS#DIV1 ISA08 (IRID) TPDUNS# GS02 (GSID) MYDUNS#DIV1 GS03 (GRID) TPDUNS#
Interchange 2 (X12):	ISA06 (ISID) MYDUNS#DIV2 ISA08 (IRID) TPDUNS# GS02 (GSID) MYDUNS#DIV2 GS03 (GRID) TPDUNS#
Interchange 3 (EDIFACT):	UNB03 (ISID) ACCY ACCY03 UNB06 (IRID) ACCX ACCX01 UNG02 (GSID) BOOKING UNG04 (GRID) ACCOUNTING

Using the FAENV file allows you to separate the functional acknowledgments into interchange and group envelopes with the values you specify.

Export/Import utility function

WebSphere Data Interchange provides an Export/Import utility function for updating WebSphere Data Interchange's databases in a batch environment. The Export/Import batch function uses three files:

1. Batch Control File (CTLFILE) containing the control information that describes the data being exported or imported.
2. Export/Import Files (E/I File) containing the data that is being imported or exported (in tagged or fixed format).
3. Print File (PRTFILE) containing a report on Export/Import activity.

The Export/Import utility allows you to choose one, any combination of, or all the following categories of WebSphere Data Interchange data for extracting (exporting) or loading (importing) EDI data:
1. EDI standards/standard transactions
2. Data formats
3. Maps
4. Control strings
5. Profiles
6. Tables

Each category is a complete set with or without the associated objects, depending on your choice. For example, for export/import of an EDI standard, the complete set includes:
* Standard definition
* Transaction definitions
* Transaction details
* Segment definitions
* Segment details

• Data element definitions

The associated objects are:
• Envelope standards
• Validation tables (if validation is required)

The categories and their associated objects are described on the following pages.

Sample programs included in the WebSphere Data Interchange product enable you to convert a fixed format (flat) file into a tagged import file for importing trading partner profile (TPPROF) members, send transaction usages/rules, and receive transaction usages/rules.

These sample programs and JCL include extensive documentation that describes how they work and how they can be used. These programs are:
• A COBOL program named EDIXF2T that is the main program executed.
• An Assembler program named EDIXTAGF, that is link edited with EDIXTAGF, and is a formatting service to create the tags.
• Sample JCL named EDIXF2T used to define the fixed flat file and to execute the conversion program EDIXF2T.

These programs and JCL are provided as source and can be modified, compiled, and link edited to suit your individual needs.

Export/Import control file (CTLFILE)

The export/import control file describes what data is being exported or imported. The control file can process multiple export or import requests.

For example, to export seven different data formats would require seven records in the control file.

The control file is specified with the CTLFILE keyword and is used with EXPORT and IMPORT commands. The suggested record format can be fixed or variable, but the record length should not exceed 120 bytes. You can use the CTLTYPE keyword to specify the type of control file. The following table describes the labels in the export/import control file.

Table 11. Export/Import control file labels

Label	Position	Length	Type	Description
CATEGORY	1	1	Char	Transaction category
REPLACE	2	1	Char	Replace named object
KEYID	3-32	30	Char	Object ID
ASSOBJ	33-54	22	Char	Associated objects
USAGETID	55-72	16	Char	Map name for usage import

Table 11. Export/Import control file labels (continued)

Label	Position	Length	Type	Description
MBRNAME	85-126	30	Char	Member name for profile import, or data format name for data format import

Export/Import control file label descriptions

CATEGORY
Specifies the category of the transaction and is required for both export and import.
Valid values are:

1	EDI standards/transaction sets
2	Data format dictionaries/data formats
3	Maps
4	Control strings
7	Profiles
8	Tables
9	All categories (import only)
A	XML dictionaries/DTDs
B	Global variables

REPLACE
For import only. Indicates whether to replace the same named object in the database.
This field does not apply to usages/rules or to profile members. It does apply to the
parent categories of entire maps and profiles. This field applies only to the primary
object identified by the category code. All associated objects that are imported will
overwrite duplicate entries currently in WebSphere Data Interchange without issuing a
warning message. Valid values are:

1　　　　Replaces the named object

(other) Does not replace the named object (default)

KEYID
Specifies the ID of the object to be exported or imported. The value must be
left-justified. For exporting EDI standard transactions, the standard dictionary name is
left-justified in the first 8 bytes. For importing specific EDI standard transaction sets, the
import file must contain an EDI complete standard with all transaction sets included. If
no key value is present for data format dictionaries, maps, or standards, all records are
exported for the specified category.

This keyword is required for importing usages/rules. For importing other items, if you do
not specify this keyword, all records for the category (specified in position 1) are
processed.

This field specifies the profile type when performing a profile handline profiles. The profile type is one of the values listed under "Additional profile layouts" on page 263. Use the MBRNAME field of the control record to limit action to a specific profile entry. KEYID values are:

REQPROF Mailbox Profile
SECUPROF Network Security Profile
NETPROF Network Profile
NETOP Network Commands
ACTLOG Activity Log
APPDEF Application Default Profile
ADAMCTL User Exits
LANGPROF Language Profile
CONTRECV Continuous Receive Profile
MQSERIES MQSeries Queue Profile
TPPROF Trading Partner Profile
SYSPROF CICS Performance

ASSOBJ

For export only. Identifies an array specifying the associated objects. The array must contain either **1** (Yes) or any other value (No) for each associated object in the following order:

For EDI standards:
1. Validation tables
2. Envelope profiles
3. Envelope standards

For maps and control strings:
 1. Usages/Rules
 2. Control strings
 3. Standard transactions (for DT maps source document definition)
 4. Data formats (for DT maps target document definition)
 5. Validation tables
 6. Translation tables
 7. User exit routines
 8. Trading partner profiles
 9. Translation exit routines
10. Network security profiles
11. Network profiles
12. Network command profiles
13. Envelope profiles
14. Envelope standards
15. Maps
16. Conversion of prior-release objects
17. Global variables
18. Validation maps
19. FA maps
20. Embedded maps

> **Note:** There are no associated objects for XML DTDs or mapping global variables.

To export a usage/rule without its map, specify **1** for usages/rules (item 1) and **0** for maps (item 15).

To export validation maps and functional acknowledgment maps, specify **1** for usage/rules (item 1) and **1** for validation maps (item 18) and **1** for functional acknowledgment maps (item 19).

For import only. The array must contain a **1** (Yes) and any other value (No) for each associated object in the following order:

For maps:

1. Usages/Rules only (the `KEYID` value must not be blank.)
16. Conversion of prior-release objects

For data formats:

16. Conversion of prior-release objects

For EDI standards:

16. Conversion of prior-release objects

For Trading Partner Profiles, to export Send usages, Receive usages, and Rules. Send usages and Receive usages are identified by the TP Nickname in the member field of the export control record. All rows with a TPNICKNM or APPLTPID are selected and formatted as 7T5 and 7T6 export records. Rules are selected by the RECVTPID column equal to the TP nickname specified in the export control records. These are formatted as 7T7 export records:

1. Usages/Rules

All usages and rules with a Sending or Receiving trading partner equal to MBRNAME field specified in the export control record will be exported.

Note: If MBRNAME field of the export control record is blank, all Trading Partner Profiles and associated Usages/Rules will be exported.

USAGETID
For import only. Specifies the map name under which to import the usages/rules. If blank, the imported map name is used.

MBRNAME
Specifies the profile member name to export or import, or a transaction ID when exporting a specific EDI standard transaction. For data format dictionaries, specifies the

dictionary name to be used. If the MBRNAME field of the export control record is blank, and TPPROF is used as the KEYID then all Trading Partner Profiles will be exported.

Export/Import files

The export/import files are used for output when you export WebSphere Data Interchange data, or are used for input when you import WebSphere Data Interchange data. Export/Import files are sequential, contain variable length records, and must be allocated with a record format of V, LRECL=8152, and BLKSIZE=8156. There are several export/import files, each associated with the category of data they hold. The logical names are listed in Table 12.

Table 12. Export/Import file logical names

logical name	Category
EDIEISTD	EDI standards and standard transactions
EDIEIADF	Data formats
EDIEITBL	Tables
EDIEITPT	Maps
EDIEICST	Control strings
EDIEIPRF	Profiles

You can maintain export files and the data in those files by using the following z/OS/TSO ISPF utility functions:

DATASET

Allocate, rename, delete, and display export data set information

COPY

Copy export data set

DSLIST

Print, rename, delete, browse, and display export data set information

Note: In CICS, if you want to maintain the export files, define them as QSAM extrapartitioned TD queues at WebSphere Data Interchange system generation. To perform maintenance, close the queues in CICS, and use the ISPF utility functions in the z/OS/TSO environment. Then run batch JCL to execute the utility functions in the z/OS/TSO environment. CICS has the export/import files allocated and the data set names cannot be shared with ISPF utilities. When export queue maintenance is complete, open the queues in CICS. If the logical name associated with the queue specifies DISP=**OLD**, the queue is cleared when opened. If the logical name associated with the queue specifies DISP=**MOD**, the queue is not cleared when opened, and new export data is appended to the end of the queue.

WebSphere Data Interchange requires that the records contained in each Export/Import file follow a certain sequence:

Export/Import files

- The first record of the Export/Import file must be the Common Control Record
  (**0C1** or **0C2**). This record contains information such as the date, time and system
  release level.
- The last record of each group must be the Common End of Group Record **(000)**.
- The data records are placed between the **0C1**/**0C2** record and the first **000** record,
  and also between each **000** record and any subsequent **000** records. These records
  vary in purpose and length, and they must be written in the format described in
  Table 13 and Table 14 on page 215.

Figure 15 shows the set of records written to the EDIEIPRF logical name by this export.

```
Export/Import common control record          0C1
First trading partner profile header record  7P1
First trading partner profile detail record  7P2
Second trading partner profile detail record 7P2
Third trading partner profile detail record  7P2
Fourth trading partner profile detail record 7P2
Fifth trading partner profile detail record  7P2
Sixth trading partner profile detail record  7P2
Last trading partner profile detail record   7P2
Export/Import common end of group record     000
```

Figure 15. Example of records written to EDIEIPROF by export

The user exported seven trading partner profiles in tagged format. This resulted in
WebSphere Data Interchange writing out the Common Control Record (0C1), and a
Trading Partner Profile Header Record (7P1) that specifies that the following records
are part of a trading partner profile export. This is followed by seven Trading Partner
Profile Detail Records (7P2), one for each trading partner. The file ends with a Common
End of Group Record (000).

Export/Import common control record (0C1/0C2)

The Common Control Record defines the format of the records (tagged or fixed), the
user ID, date, time, and WebSphere Data Interchange version/release number. This
record is required and it must be the first record in the Export/Import record data set.

Note: The first 3 bytes of this record are **0C1** or **0C2**, starting in column 1. The
0C1/0C2 record occurs only once in an export/import data set and is always the
first record.

Table 13. Export/Import Common Control Record (0C1/0C2) fields

Field Name	Position	Length	Type	Field Description
Category	1-1	1	Char	Export/Import record category = **0**
Rectype	2-3	2	Char	Export/Import record type code Record type = **C1** (tagged format) Record type = **C2** (fixed format)

Table 13. Export/Import Common Control Record (0C1/0C2) fields (continued)

Field Name	Position	Length	Type	Field Description
User ID	4-11	8	Char	User ID of person creating the export/import record data set
Date	12-17	6	Char	Creation date of export/import data set (YYMMDD)
Time	18-23	6	Char	Creation time of export/import data set (HHMMSS)
Language	24-26	3	Char	WebSphere Data Interchange language English (United States) = **ENU**
DIVersRel	27-38	12	Char	WebSphere Data Interchange version and release Vers 3 Rel 1 = 010301000000 Vers 4 Rel 1 = 010401000000

The following example of a completed 0C1 record depicts the user ID as SMITH, a creation date of **010419**, a creation time of **161616**, English as the language, and Version 4 Release 1 (**010401000000**) as the WebSphere Data Interchange version and release.

```
0C1 SMITH   010419161616ENU010401000000
```

Export/Import common end of group record (000)

The Common End of Group Record indicates the end of an export/import group of records. This record is required and must be the last record of an export/import group. Table 14 describes the fields of the common end of group record.

Table 14. Export/Import Common End of Group Record (000) Fields

Field Name	Position	Length	Type	Field Description
Category	1-1	1	Char	Export/Import record category = **0**
Rectype	2-3	2	Char	Export/Import record type = **00**

The Common End of Group Record contains only **000**, starting in column 1.

Export/Import file data area

The content data area varies depending upon which category of data you are importing or exporting. The following categories are supported:
• EDI standards and standard transactions
• Data formats
• Maps
• Control strings
• Profiles
• Tables

Export/Import files

Regardless of the category chosen, the data portion (tagged or fixed) of an export/import file starts in column 4 of the record. Columns 1-3 are reserved for the Export/Import record ID (RECID). As discussed earlier, the Common Control Record (0C1/0C2) and Common End of Group Record (000) are fixed records. All records contained in the export/import file data area might be tagged or fixed format records, except for the control string (G1) and document layout (L1), which are native database format.

Note: When fixed format record layouts are changed due to design changes in WebSphere Data Interchange, new fields are added to the end of the records.

The following tables define the order in which the fields are exported (position of fixed format fields), the length of fixed format fields (maximum length for tagged fields), and the type of data. Data with type **Char** can have any valid characters. Data with type **DEC** must have only decimal characters (0-9). When exported, data with type **HEX** is expanded to two hex representation characters (0-9, A-F) for each hex byte.

Exporting and importing trading partner profiles

The trading partner profile contains information about the companies that exchange business transactions with you. There must be one record for each trading partner that you wish to define.

The Export/Import format for defining trading partners includes the record types defined in Table 15.

Table 15. Trading partner record types

Record type	Description
7P1	One trading partner profile header record
7P2	One or more trading partner profile records
7Z1	Zero or more contact records
7P3	Zero or more trading partner contact records
7P4	One or more trading partner control number records
7A1	At most, one trading partner comment record

The trading partner profile record group must be followed by the Export/Import Common End of Group Record (000). Figure 16 on page 217 illustrates a valid sequence of records written to logical name EDIEIPRF when exporting multiple trading partner profiles.

```
0C1 Export/Import common control record
7P1 Trading partner header record
7P2 Trading partner profile record
7P4 Trading partner control number record
7A1 Trading partner comment record
7P2 Trading partner profile record
7Z1 Contact record
7Z1 Contact record
7P3 Trading partner contact record
7P3 Trading partner contact record
7P3 Trading partner contact record
7P4 Trading partner control number record
7A1 Trading partner comment record
7P2 Trading partner profile record
7Z1 Contact record
7P3 Trading partner contact record
7P4 Trading partner control number record
7P2 Trading partner profile record
7P4 Trading partner control number record
7P2 Trading partner profile record
7P4 Trading partner control number record
000 Export/Import common end of group record
```

Figure 16. Example of multiple trading partner profiles

Importing new definitions to WebSphere Data Interchange

After the Export/Import file data set has been updated, you can submit the updates using the batch utility or with the WebSphere Data Interchange Import function.

Export/Import trading partner profile header record (7P1)

The trading partner profile header record is required and must be the first record of a trading partner profile record group. A trading partner profile group is defined as one trading partner profile header record (7P1) followed by one or more trading partner profile records (7P2).

The trading partner profile header record contains the trading partner profile indicator and profile description. It is described in Table 16.

Table 16. Trading partner profile header record 7P1

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID =**TPPROF**
PROFDESC	12-46	35	Char	Profile/table description

Export/Import trading partner profile record (7P2)

The trading partner profile record defines the descriptive and control data that identify an individual trading partner and its processing options. Multiple 7P2 records can be created (one record per trading partner) to support the creation of multiple trading partner profiles.

For fixed format records, the fields must be in the position specified and for the length specified in the table.

For tagged files, the tags for the trading partner profile record (7P2) can be placed in any column or any order you choose. When importing data, all incoming variables must be delimited by parentheses. When exporting, outgoing variable data is placed between parentheses. Profile member tags and their associated maximum record lengths are listed in Table 17.

Table 17. Trading partner profile members

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID
TPNICKN	12-27	16	Char	Trading partner nickname
NETID	28-35	8	Char	Network ID
SYSQUAL	36-36	1	Char	System qualifier
SYSID	37-44	8	Char	System ID
ACCTID	45-76	32	Char	Account number
USERID	77-108	32	Char	User ID
INTQUAL	109-112	4	Char	Interchange ID qualifier
INTID	113-147	35	Char	Interchange ID
CONAME	148-187	40	Char	Company name
ADDR1	188-227	40	Char	Company address line 1
ADDR2	228-267	40	Char	Company address line 2
PHONE	268-292	25	Char	Contact phone
CONTACT	293-322	30	Char	Contact name
SNDPASS	323-336	14	Char	Interchange send password
RCVPASS	337-350	14	Char	Interchange receive password
SECPROF	351-358	8	Char	Network security profile ID
NETCLS	359-359	1	Char	Network message class
NETCHG	360-360	1	Char	Network charges code
NETACK	361-361	1	Char	Network acknowledgment
NETVCHK	362-362	1	Char	Destination verification code
NETRETN	363-365	3	Char	Retention period
NETEDIO	366-366	1	Char	EDI receive option
NETEDIP	367-367	1	Char	EDI processing override
STGFMTOV	368-368	1	Char	Storage format override

Table 17. Trading partner profile members (continued)

Tag	Position	Length	Type	Description
MACHTYPE	369-369	1	Char	Machine type
STGFMT	370-370	1	Char	Storage format
EOTID	371-371	1	Char	End of text/message delimiter
LOGENV	372-372	1	Char	Log standard data
FNCGRP	373-373	1	Char	Functional group code
SEDLM	374-375	2	Char	Subelement delimiter
DEDLM	376-377	2	Char	Data element delimiter
SEGDLM	378-379	2	Char	Segment delimiter
SEGSEP	380-381	2	Char	Segment ID separator
DECNOT	382-382	1	Char	Decimal notation
RLSCHAR	383-384	2	Char	Release character
INTCTLNO	385-393	9	Char	Interchange mask
GRPCTLNO	394-402	9	Char	Group mask
TRXCTLNO	403-411	9	Char	Transaction mask
COMMENT1	412-451	40	Char	Comment line 1
COMMENT2	452-491	40	Char	Comment line 2
NETCMDS	492-499	8	Char	Network commands file
TPDATALINE	500-531	32	Char	Trading partner data phone number
TIMEOUT	532-535	4	Dec	Data line communication timeout
SEGMENTED	536-536	1	Char	Segmented output option
SUFFIX	537-538	2	Char	File suffix
TPENVSUF	539-540	2	Char	Envelope profile suffix
TPGENRCV	541-541	1	Char	Allow generic receive
TPCMPRES	542-542	1	Char	Compression
TPSUPAD3	543-582	40	Char	Address line 3
TPSUPCTY	583-612	30	Char	City name
TPSUPST	613-614	2	Char	State code
TPSUPPST	615-629	15	Char	Postal code
TPSUPCON	630-659	30	Char	Country
TPSUPFAX	660-684	25	Char	Fax number
TPSUPU3	685-724	40	Char	Comment line 3
TPSUPU4	725-764	40	Char	Comment line 4
TPSUPU5	765-804	40	Char	Comment line 5
TPSUPU6	805-844	40	Char	Comment line 6
TPSUPU7	845-884	40	Char	Comment line 7
TPSUPU8	885-924	40	Char	Comment line 8

Exporting and importing trading partner profiles

Table 17. Trading partner profile members (continued)

Tag	Position	Length	Type	Description
TPSUPU9	925-964	40	Char	Comment line 9
TPSUPU10	965-1004	40	Char	Comment line 10
DESCRIPT	1005-1034	30	Char	Description
TPTYPE	1035-1035	1	Char	Trading partner type
PRIORITY	1036-1036	1	Char	Expedite priority
PROCESS	1037-1076	40	Char	Process ID
DESEP	1077-1078	2	Char	Repeating data element separator

The following example shows part of an export/import file that contains records for exporting/importing three trading partner profiles.

```
0C1SMITH     0140419161616ENU010103000000

7P1PROFID(TPPROF) PROFDESC(Trading partner profile)
7P2PROFID(TPPROF) TPNICKN(IBM)  NETID(IINR4)  SYSQUAL(4) NETACK(N)PASSWORD(N WPASS)
7P4TPNICKN(IBM) APPRECID(JONES) APPRECQ(01) INTCTLNO(000010000)
   GRPCTLNO(000015000) TRXCTLNO(000020000)
7P2PROFID(TPPROF) TPNICKN(TPT40) NETACK(Y)  SYSQUAL(4) NETID(IINR5) PHONE(555-1234)
7P2PROFID(TPPROF) TPNICKN(VAN04PR) CONTACT(John Doe) SNDPASS(Y) RCVPASS(Y)
7P4TPNICKN(VAN04PR) APPRECID(SMITH) APPRECQ(02) APPDOCID(850) INTCTLNO(000030001)
GRPCTLNO(000040000) TRXCTLNO(000045000)
000
```

Figure 17. Example of exporting trading partner profiles

Figure 17 shows how the 0C1 record is the first record in the Export/Import file, followed by profile data records with a 000 record as the last record. The 7P1 and 7P2 fields are the record IDs (RECIDs). They must be the first 3 bytes of the data record. In this example, the **7** indicates the category code, the **P** indicates the object being exported/imported (in this case, Profile), and the **1**, **2**, and **4** are internal WebSphere Data Interchange indicators.

Trading partner profile member field descriptions

Category code: Must contain **7**.

Record type: Must contain **P2**.

Profile ID (PROFID): Must contain **TPPROF**.

Trading partner nickname (TPNICKN): The name you use to refer to this trading partner. Use the same name throughout WebSphere Data Interchange to refer to this trading partner by nickname.

Network ID (NETID):   The name that identifies the network used to communicate with this trading partner. It must match the name of a member in the network profile.

System qualifier (SYSQUAL):   For AT&T Global Network users, enter an **I** if intersystem addressing is required for this trading partner (Network reference: DTBLTYP). Enter the ID of the other system in the next field.

System ID (SYSID):   For intersystem addressing, the ID of the system responsible for the receiver's account. For AT&T Global Network users, the ID is limited to three characters (Network reference: DTBLID).

Account number (ACCTID):   The account number assigned to your trading partner by the network. The entry must be left-justified. For sending and receiving EDI in ISA/IEA envelopes, the last position must be blank. The combination of this field and the User ID field must be unique for all members in the profile.

If the interchange ID is blank, the account number and user ID make up the receiver ID in the interchange envelope. UCS (BG/EG) envelopes are an exception, where the phone number field contains the receiver ID.

User ID (USERID):   The user ID assigned to your trading partner by the network. The entry must be left-justified. The combination of this field and the account number field must be unique for all members in the profile.

If the interchange ID field is blank, the account number and user ID make up the receiver ID in the interchange envelope. UCS (BG/EG) envelopes are an exception, where the phone number field contains the receiver ID.

Interchange ID qualifier (INTQUAL):   The type of interchange ID entered in the interchange ID field such as a Dun and Bradstreet number (DUNS). The EDI standard defines these codes. If the interchange ID is blank, the enveloper takes the qualifier from the envelope profile member. The combination of this field and the interchange ID field must be unique for each member unless the fields are left blank.

Interchange ID (INTID):   The ID used as the interchange receiver ID (recipient) when you send to this partner and as the interchange sender ID when you receive from this partner. If this field is blank, the enveloper uses the account number and user ID (or phone number for BG/EG interchanges). The combination of this field and the interchange ID qualifier field must be unique for each member unless the fields are left blank.

Company name (CONAME):   The name of the trading partner's company.

Company address line 1 (ADDR1):   The first line of the trading partner's address.

Company address line 2 (ADDR2):   The second line of the trading partner's address.

Contact phone (PHONE):   The trading partner's phone number (free-form). For UCS (BG/EG) enveloping, if the interchange ID is blank, the enveloper uses the phone number as the interchange ID.

Exporting and importing trading partner profiles

Contact name (CONTACT): The name of the person you communicate with when dealing with this trading partner.

Interchange send password (SNDPASS): The password agreed upon by you and your trading partner for sending to this trading partner. This value maps to the `password data type` in the interchange envelope.

Interchange receive password (RCVPASS): The password agreed upon by you and your trading partner for receiving from this trading partner. If this value matches the interchange password (password data type) that was received, translation occurs.

Network security profile ID (SECPROF): The ID of the default network security profile member that specifies the encryption and authentication processes that apply to EDI data for this trading partner. This profile member is always used when receiving from this partner. It is used by default when sending to this partner unless the send usage/rule for the map specifies a different member in the `Group security profile name` field or the `Trans security profile name` field.

Network message class (NETCLS): Applies only to sending. Indicates any special status (for example, test) of the data being sent. For AT&T Global Network users (Network reference: MSGNCLS), **T** indicates test status, and a blank indicates normal status.

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Network charges code (NETCHG): Indicates how charges are shared between sender and receiver. For AT&T Global Network users (Network reference: MSGCHRG), valid values are:

1   Receiver pays all charges.

2   Receiver pays all charges if agreed to; otherwise, charges are split between sender and receiver.

3   Receiver pays all charges if agreed to; if receiver does not agree to pay all charges, charges are split between sender and receiver if agreed to; otherwise, the sender pays all charges.

4   Charges are split between sender and receiver if agreed to; otherwise, the sender pays all charges.

5   Charges are split between sender and receiver.

6   Sender pays all charges.

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Network acknowledgment (NETACK): Indicates which network acknowledgments (receipt, delivery, purge) you want to receive when sending to this partner. For AT&T Global Network users (Network reference: MSGRCPT), valid values are:

(blank) No acknowledgments

R Receipt only

D Delivery only

B Both receipt and delivery

A Purge only

C Both receipt and purge

E Either purge or delivery

F Receipt and either delivery or purge

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Destination verification code (NETVCHK): Indicates whether the destination is verified before the data is sent. For AT&T Global Network users (Network reference: MSGVCHK), valid values are:

N Does not request verification (default)

Y Requires verification

F Requests verification, but sends even if the destination is not verified (useful for intersystem addressing)

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Retention period (NETRETN): The number of days that the data is kept in the network mailbox before being purged, if it is not received.

If you are using more than one network, contact a representative from the network you are using or refer to an appropriate manual to determine the correct values to use for this field. The following pertains to the AT&T Global Network and to Expedite Base 4.2 and higher.

Valid values are **0** through **180**. For installations in the U.S., the default retention period is **30** days. For installations outside the U.S., contact your marketing representative for your default value.

If you specify **0** or blank, Information Exchange retains the data for the default period. In addition, if you specify a value larger than the maximum for your system, Information Exchange retains the data for the default period.

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Note: The size of this field for Expedite/CICS is two digits, meaning the valid values for Expedite/CICS are **0-99**. The two left-most digits entered in this field are used by

Expedite/CICS as the retention period. For example: if you enter 180, the
retention period value sent to Expedite/CICS will be **18**.

EDI receive option (NETEDIO): Indicates whether you want EDI segments stored in
the file as separate records. For AT&T Global Network users (Network reference:
EDIOPT), valid values are:

Y   Ends records at the segment delimiter (default)

N   Does not end records at the segment delimiter

EDI processing override (NETEDIP): Indicates whether you want the EDI data you
receive to have special EDI processing, which breaks records by the segment delimiter.
For AT&T Global Network users (Network reference: EDIPROC), valid values are:

Y   Performs EDI processing if the common data header indicates that the data is
               in EDI standard format (default)

N   Does not perform EDI processing, regardless of the values in the common
               data header

If your request does not specify a trading partner, this information is taken from the
mailbox (requestor) profile.

Storage format override (STGFMTOV): If you are using more than one network,
contact a representative from the network you are using to determine the available
options.

Information Exchange uses common data header control information that allows users
to send or receive information electronically. The control header contains information
such as the type of record, original record format, sending system type, type of data
being sent, whether the data is in an EDI format, and a unique record number for
tracking.

If you are using IBM Expedite Base/MVS (IEBASE), refer to the `DLMOVERRIDE` command
option keyword for an up-to-date list of acceptable values. Valid values are:

Y   Formats the data according to the `DELIMITED` parameter, even if the common
               data header (CDH) indicates a delimiter type

N   Ignores the value in the `DELIMITED` parameter, and formats the data according
               to the record type (if any) in the CDH (default)

If there is no common data header, the format indicated in the `Storage format` field is
used. If your request does not specify a trading partner, the information is taken from
the mailbox (requestor) profile.

Machine type (MACHTYPE): This field is not currently supported.

Storage format (STGFMT): Indicates to the network how data is stored for free-form
messages and files. When determining what option to select, you should consider the

type of data you want to send and how the file will be received. If you are using more than one network, contact a representative from the network you are using to determine the available options. Valid values are:

C   Stores each record with a carriage return and line-feed character (CRLF) and uses the end of file (EOF) character to mark the end of file. These characters are represented and stored as hex values 0D0A and 1A respectively. This option is generally used to send files containing program source code with variable length records. Output records will include data up to the carriage return and line feed characters (CRLF).

L   Precedes each record with a 2-byte hexadecimal record length. Select this option for sending data in fixed format or for sending binary data. The output record length is determined by the value in the 2 bytes containing the record length.

N   Stores the data as it is received (default). Output records are built based on the record length of the data set allocated to receive the data.

If you are using Expedite/CICS, refer to the DTYPE field for an up-to-date list of valid values:

A   Stores each record with a carriage return and line-feed character (CRLF) and uses the end of file (EOF) character to mark the end of file. These characters are represented and stored as hex values 0D0A and 1A respectively. This option is generally used to send files containing program source code with variable length records. Output records will include data up to the carriage return and line feed characters (CRLF).

B   Precedes each record with a 2-byte hexadecimal record length. Select this option for sending data in fixed format or for sending binary data. The output record length is determined by the value in the 2 bytes containing the record length.

E   Stores each record based on the delimiters found in the EDI standard data.

O   Stores data as it is received (free-form).

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

End of text or message delimiter (EOTID):   The character that signifies to the network the end of the data. It applies to free-form messages and data files, not to EDI standard transactions. (Network reference: EORCHAR, EOMIND).

Log standard data (LOGENV):   Indicates whether the translator should include EDI standard (untranslated) data when logging an image of a transaction. For EDI data with errors, the EDI standard data is logged even if you enter **N**.

This field can be overridden by the `Log standard data` field of the Application Defaults (APPDEFS) profile. Valid values are:

Y   Includes EDI standard (untranslated) data when logging transaction image

N   Does not include EDI standard (untranslated) data when logging transaction
image

Functional group code (FNCGRP):   Indicates whether the enveloper should create
functional groups for transactions with type **E** envelopes. (Functional groups are always
created for types **I**, **U**, and **X**; they are never created for type **T**). If you choose not to
have functional groups, the different message types are enveloped together. Valid
values are:

Y   Creates functional groups

N   Does not create functional groups

Subelement delimiter (SEDLM):   The character (in hex notation) that separates
subelements (also called component data elements) in a transaction set. An entry here
(other than 00 or 40) overrides the character specified by the EDI standard.

If entered, this character must be different from the characters specified for the data
element delimiter, segment delimiter, segment ID separator, decimal notation, and
release character fields, with this exception: the subelement delimiter and the data
element delimiter can be the same for envelopes with types **I**, **U**, and **X**.

Data element delimiter (DEDLM):   The character (in hex notation) that separates the
data elements in a transaction set. An entry here (other than 00 or 40) overrides the
character specified by the EDI standard.

If entered, this character must be different from the characters specified for the
subelement delimiter, segment delimiter, segment ID separator, decimal notation, and
release character fields, with this exception: the subelement delimiter and the data
element delimiter can be the same for envelopes with types **I**, **U**, and **X**.

Segment delimiter (SEGDLM):   The character (in hex notation) that marks the end of
each segment in a transaction set. An entry here (other than 00 or 40) overrides the
character specified by the EDI standard.

If entered, this character must be different from the characters specified for the
subelement delimiter, data element delimiter, segment ID separator, decimal notation,
and release character fields.

Segment ID separator (SEGSEP):   The character (in hex notation) that separates the
segment ID and the first data element in a segment. An entry here (other than
00 or 40) overrides the character specified by the EDI standard.

If entered, this character must be different from the characters specified for the
subelement delimiter, data element delimiter, segment delimiter, decimal notation, and
release character fields.

Decimal notation (DECNOT):   The character that represents the decimal point in a
transaction set. For type **E** envelopes, an entry here overrides the character specified
by the EDI standard. For all other types, a period represents the decimal point.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, segment ID separator, and release character fields.

Release character (RLSCHAR): For type **E** and **T** envelopes, the character (in hex notation) that is used to indicate where a delimiter is being used as part of the data. An entry here (other than 00 or 40) overrides the character specified by the EDI standard.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, segment ID separator, and decimal notation fields.

Interchange mask (INTCTLNO): The initial reference number that the enveloper places in the **CN** `data type` of the interchange header and trailer. This value is used as the base value for each trading partner/receiver ID combination. This field does not represent the current control number for this trading partner. Use the `Control Number` option on the Member List panel to request control number information.

Group mask (GRPCTLNO): The initial reference number or special codes that the enveloper places in the **CN** `data type` of the functional group header and trailer. This value is used as the base value for each trading partner/receiver ID combination. Use caution when updating this field. This field does not represent the current control number for this trading partner. Use the `Control Number` option on the Member List panel to request control number information.

Transaction mask (TRXCTLNO): The initial reference number or special codes that the enveloper places in the `CN data type` of the transaction set header and trailer. This value is used as the base value for each trading partner/receiver ID combination. Use caution when updating this field. This field does not represent the current control number for this trading partner. Use the `Control Number` option on the Member List panel to request control number information.

Comment line 1 (COMMENT1): Free-form notes about the trading partner.

Comment line 2 (COMMENT2): Free-form notes about the trading partner.

Network commands file (NETCMDS): The name of a PDS member to be allocated the z/OS logical name EDINTCMD. This member will contain the commands that you want to pass to the network.

WebSphere Data Interchange reads the commands from the PDS member and writes the commands to the network input file specified in the network profile member after all substitutable variable tags are resolved by WebSphere Data Interchange.

TP data phone number (TPDATALINE): The phone number to dial to connect to your trading partner directly. This is the number used by your computer to talk directly to your trading partner's computer. The Contact phone field contains the voice phone number to call a human (not a computer) at your trading partner's site.

Exporting and importing trading partner profiles

Data line communication timeout (TIMEOUT): Specifies the maximum allowable time that the communications line can be idle without dropping the connection. If the data line is idle for a time greater than the value entered, the line is dropped. If you specify a trading partner when requesting network activity, the value for this field is taken from the trading partner profile. Otherwise, the value for this field is taken from the mailbox (requestor) profile.

Segmented output option (SEGMENTED): Indicates whether you want EDI segments stored in the output file as separate records. Valid values are:

Y   Ends records at the segment delimiter

N   Does not end records at the segment delimiter (default)

File suffix (SUFFIX): A 2-character suffix for the logical name used to store the results of a fixed-to-fixed translation. The logical name is taken from the `Application file name` field of the target data format.

This 2-character suffix is appended to the `Application file name` or, if the file name is longer than 6 characters, overwrites the last two characters of the `Application file name`. Use this suffix to separate the results of a fixed-to-fixed translation by trading partner. If you do not want to separate results by trading partner, either leave the suffix blank or assign all the logical names to the same physical file.

For CICS, you can use the suffix to identify a unique TS queue for each trading partner.

Envelope profile suffix (TPENVSUF): A 2-character suffix for a generic envelope profile member name. A generic envelope profile name has an ampersand (&) as the first character of the name followed by a 1 to 6-character common name.

This 2-character suffix is used to generate the envelope profile member name for enveloping transactions. Use this suffix to use different envelope profile members for different trading partners. To use the same envelope profile member for several trading partners, enter the same suffix in each of the trading partner profiles, or leave the suffix blank to use a common envelope profile member (with the same name as the common name in the usage/rule). You must define an envelope profile member for each combination of common name/suffix value specified in a trading partner profile.

The envelope profile member name is specified on the Send and Receive Trading Partner Usage panels. The generic form of the name is most useful for generic usages/rules but can be used for any usage/rule.

Allow generic receive (TPGENRCV): Indicates whether transactions received from this trading partner can be translated using generic receive trading partner usages/rules. Valid values are:

Y   Allows translation via generic receive trading partner usages/rules

N   Does not allow translation via generic usages/rules (default)

Compression (TPCMPRES): Indicates whether Expedite Base/MVS or Expedite/CICS should compress your data prior to transmission. Valid values are:

N	Does not compress data (default).
Y	Compresses data prior to transmission. If the file sent contains multiple EDI envelopes, all envelopes are sent compressed.
T	Conditionally compresses data prior to transmission. For Expedite Base/MVS, compression will occur if the `SENDER`, `RECEIVER`, and `COMPRESS` parameters are listed in the CPLOOKUP EXPFILE file. This file defines a series of paired receivers and senders, and for each pair, indicates whether compression should be performed. For Expedite/CICS, compression will occur if the sender/receiver pair is found in the lookup table and `COMPRESS` equals **Y** in the lookup table.

For more information, refer to the or to .

If a trading partner profile member is specified on the send request, the compression value is taken from the trading partner profile. Otherwise, this information is taken from the mailbox (requestor) profile.

Address line 3 (TPSUPAD3):   The third line of the trading partner's address.

City name (TPSUPCTY):   The trading partner's city.

State code (TPSUPST):   The trading partner's state.

Postal code (TPSUPPST):   The trading partner's zip code.

Country (TPSUPCON):   The trading partner's country.

Fax number (TPSUPFAX):   The trading partner's fax number.

Comment line 3 (TPSUPU3):   Free-form remarks.

Comment line 4 (TPSUPU4):   Free-form remarks.

Comment line 5 (TPSUPU5):   Free-form remarks.

Comment line 6 (TPSUPU6):   Free-form remarks.

Comment line 7 (TPSUPU7):   Free-form remarks.

Comment line 8 (TPSUPU8):   Free-form remarks.

Comment line 9 (TPSUPU9):   Free-form remarks.

Comment line 10 (TPSUPU10):   Free-form remarks.

Description (DESCRIPT):   A free-form description of the member.

Trading partner type (TPTYPE):   Indicates the type of trading partner this definition represents. Valid values are:

Exporting and importing trading partner profiles

E   EDI trading partner (default). Considered to be external to your business organization. The interchange control numbers are generated using the EDI trading partner ID.

A   Application trading partner. Considered to be internal to your business organization. For example, divisions or departments might be internal trading partners. Use this setting when no specific EDI trading partner is defined, or in combination with an EDI trading partner. Interchange control numbers are generated using the application and EDI trading partner combination. Use this trading partner type with centralized EDI when multiple application trading partners do business with the same EDI trading partner.

B   Both EDI trading partner and application trading partner. Considered to be both external (EDI) and internal (application). Use this setting when your organization provides EDI translation services to customers who are trading partners. The value in this field is directly related to the usage/rule setup.

For more information, refer to the WebSphere Data Interchange User's Guide.

Priority (PRIORITY): Indicates whether messages are sent through Expedite Base/MVS or Expedite/CICS with normal delivery or priority delivery. Valid values are:

(blank)   Normal delivery

P   High priority

For more information, refer to the SEND commands in the Expedite Base/MVS Programming Guide or to Customizing and Developing Applications with Expedite/CICS.

If a trading partner profile member is specified on the send request, this value is taken from the trading partner profile. Otherwise, this information is taken from the mailbox (requestor) profile.

Process: The process name used with data transformation to associate with this trading partner, such as `PRODUCTION_PURCHSG_V1R1`. You can create data transformation rules for a group of maps (creating a process) and then associate trading partners with the process. If you change the map rules for the process, all trading partners associated with the process use the changed maps automatically.

In the same manner, you can use the process name to create classes of trading partners such as production, non-production, financial institutions, XML trading partners, or EDI trading partners. You can create data transformation map rules to associate with each trading partner class.

Repeating data element separator (DESEP): The hex character separator placed between repeating data elements for all transactions sent to this trading partner. A value in this field (other than 00 or 40) overrides the separator character defined in the EDI standard. This character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, segment ID separator, decimal notation, and release characters. This hex character should not be found within the EDI standard data.

Trading partner contact definition (7P3)

Trading partner contact information is exported or imported automatically when a trading partner record is exported or imported. Each trading partner contact record specifies a contact that is related to a trading partner. The fields are described in Table 18..

Table 18. Trading partner contacts definition

Tag	Position	Length	Type	Description
TPNICKN	4-19	16	Char	Trading partner nickname
TPCONNM	20-59	40	Char	Trading partner contact name

Trading partner control numbers (7P4)

Trading partner control numbers are exported or imported automatically when a trading partner is exported or imported. Each trading partner control number specifies a control number that is related to a trading partner. The fields are described in Table 19.

Table 19. Trading partner control numbers

Tag	Position	Length	Type	Description
TPNICKN	4-19	16	Char	Trading partner nickname
APPRECID	20-54	35	Char	Receiver ID
APPRECQ	55-58	4	Char	Receiver qualifier
APPDOCID	59-66	8	Char	Transaction ID, message ID, or document definition name
INTCTLNO	67-75	9	Char	Interchange control number
GRPCTLNO	76-84	9	Char	Group control number
TRXCTLNO	85-93	9	Char	Transaction control number

Comments definition (7A1)

Comments are exported or imported automatically when a trading partner is exported or imported. The comment record specifies textual data about a trading partner. The fields are described in Table 20.

Table 20. Comments definition

Tag	Position	Length	Type	Description
CMTTYPE	4-11	8	Char	Comments entity type
CMTKEY	12-46	35	Char	Comments entity type key
CMTTEXT	47-2094	2048	Char	Trading partner contact comments

Exporting and importing trading partner profiles

Contact definition (7Z1)

Contact information is exported or imported automatically when a trading partner record is exported or imported. The contact record specifies data about a contact. The fields are described in Table 21.

Table 21. Contacts definition

Tag	Position	Length	Type	Description
TPCONNM	4-43	40	Char	Trading partner contact name
TPCONTLE	44-83	40	Char	Trading partner contact job title
TPCONAD1	84-123	40	Char	Trading partner contact address line 1
TPCONAD2	124-163	40	Char	Trading partner contact address line 2
TPCONAD3	164-203	40	Char	Trading partner contact address line 3
TPCONCTY	204-233	30	Char	Trading partner contact city
TPCONST	234-235	2	Char	Trading partner contact state
TPCONPST	236-250	15	Char	Trading partner contact postal code
TPCONCON	251-280	30	Char	Trading partner contact country
TPCONPHN	281-305	25	Char	Trading partner contact phone
TPCONFAX	306-330	25	Char	Trading partner contact fax number
TPCONOTH	331-355	25	Char	Trading partner contact other phone
TPCONEML	356-395	40	Char	Trading partner contact e-mail address
CMTTEXT	396-2443	2048	Char	Trading partner contact comments

Export and importing EDI standard records

EDI standards can be exported/imported using tagged or fixed format files. The EDI standard/transaction set includes:

1Y1 Dictionary record

1Y2 Transaction header record

1Y3 Transaction detail record

1Y4 Segment header record

1Y5 Segment detail record

1Y6 Data element header record

1Y7 Data element detail record

1Y8 Segment notes record

1Y9 Transaction notes record

1YA Composite data element notes record

1YB   Envelope detail record

Three types of associated objects can be exported using tags:
1. Validation tables
2. Envelope profile
3. Envelope standard

In Table 22, you can export all of the data elements in a particular segment. Since a segment is made up of many data element usages/rules and definitions, multiple Y6 and Y7 records are shown here.

Table 22. Exporting all data elements in a particular segment

Record Type	Record Name
0C1	Export/Import common control record
1Y1	EDI standard dictionary record
1Y2	EDI standard transaction header record
1Y3	EDI standard transaction detail record
1Y4	EDI standard segment header record
1Y5	EDI standard segment detail record
1Y6	EDI standard data element header record
1Y7	EDI standard data element detail record
1Y6	EDI standard data element header record
1Y7	EDI standard data element detail record
1Y8	EDI standard segment notes record
1Y9	EDI standard transaction notes record
1YA	EDI standard composite data element notes record
1YB	EDI standard envelope details record
000	Export/Import common end of group record

The following example shows part of an export/import file data set that contains records for exporting/importing one transaction of an EDI standard. The transaction has one segment with three data elements.

```
0C1ADMIN    011116151224ENU010401000000

1Y1 STDID(AARV4R4) AGENCY(X) VERSION(40) RELEASE(40) INDUSTRY(RAIL) DICTENVFLAG(D)
        INTENV(X) STDDESC(Rail Carrier 4040)  ALLTRXS(N)
1Y2 STDID(AARV4R4) STDTRID(426) FNGRPID(SR) STDDESC(Rail Revenue Waybill) STDPURP(Purpose)
1Y3 STDID(AARV4R4) STDTRID(426) TABLENO(1) POSNO(20) SEGID(ZR) SEGREQ(M) SEGMAX(1) UNLIMMAX(0)
UNLPREP(0) 1Y4 STDID(AARV4R4) SEGID(ZR) STDDESC(Waybill Reference Identification)
STDPURP(To transmit identity and reference information of the waybill)
1Y5 STDID(AARV4R4) SEGID(ZR) POSNO(1) DEID(762) DEREQ(M) 1Y6 STDID(AARV4R4) DEID(762)
DETYPE(ID) DEMIN(1) DEMAX(1) CODELIST(762R440) STDDESC(Waybill Response Code)
STDPURP(Code indicating a waybill response)
```

Exporting and importing trading partner profiles

```
1Y9 STDID(AARV4R4) STDTRID(426) TABLENO(1) POSNO(241) NOTETYPE(N) NOTEPARA(1) RELATION(CM)
    STDNOTE(One R2B segment is needed for each line haul carrier)
1Y8 STDID(AARV4R4) SEGID(ZR) POSNO(4) NOTETYPE(N) NOTEPARA(1) RELATION(P) REL1(4) REL2(5)
000
```

EDI standard dictionary record (1Y1)

This record defines the EDI standard being used. The fields are described in Table 23.

Table 23. EDI standard dictionary record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
AGENCY	34-41	8	Char	Agency issuing the standard
VERSION	42-49	8	Char	Version of the standard
RELEASE	50-57	8	Char	Release level of the standard
INDUSTRY	58-65	8	Char	Industry code of the standard
DICTENVFLAG	66-66	1	Char	Standard type flag (dictionary or envelope)
INTENV	67-96	30	Char	Interchange envelope name
STDDESC	97-176	80	Char	Description of the standard
STDPURP	177-688	512	Char	Purpose of the standard
CMTTEXT	689-2736	2048	Char	Comments
ALLTRXS	2737-2737	1	Char	All transactions flag

EDI standard transaction header record (1Y2)

This record defines the transaction header fields being used. The fields are described in Table 24.

Table 24. EDI standard transaction header record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
STDTRID	34-41	8	Char	Transaction ID
FNGRPID	42-49	8	Char	Function group ID
STDDESC	50-129	80	Char	Description of the standard
STDPURP	130-2177	2048	Char	Purpose of the standard
CMTTEXT	2178-3977	1800	Char	Comments

EDI standard transaction detail record (1Y3)

This record defines the detail fields being used. If you are importing a transaction set, you can define multiple transaction detail records. The fields are described in Table 25 on page 235.

Table 25. EDI standard transaction detail record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
STDTRID	34-41	8	Char	Transaction ID
TABLENO	42-47	6	Char	Table ID
POSNO	48-53	6	Char	Position number
SEGID	54-61	8	Char	Segment ID
SEGREQ	62-62	1	Char	Segment required flag
SEGMAX	63-73	11	Char	Number of times segment can be repeated
UNLIMMAX	74-79	6	Char	No limit on the number of times segment can be repeated
NLOOPID	80-85	6	Char	Loop ID
LPREP	86-96	11	Char	Number of times loop can be repeated
UNLPREP	97-102	6	Char	No limit on the number of times loop can be repeated
LPLEV	103-108	6	Char	Loop level

EDI standard segment header record (1Y4)

This record defines the segment header fields being used. The fields are described in Table 26.

Table 26. EDI standard segment header record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
SEGID	34-41	8	Char	Segment ID
STDDESC	42-121	80	Char	Description of the standard
STDPURP	122-2169	2048	Char	Purpose of the standard
CMTTEXT	2170-3969	1800	Char	Comments

EDI standard segment detail record (1Y5)

This record defines the segment detail fields being used. The fields are described in Table 27.

Table 27. EDI standard segment detail record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
SEGID	34-41	8	Char	Segment ID
POSNO	42-47	6	Char	Position number

Table 27. EDI standard segment detail record (continued)

Tag	Position	Length	Type	Description
DEID	48-55	8	Char	Data element ID
DEREQ	56-56	1	Char	Data element required flag
SEGMAX	57-67	11	Char	Number of times segment can be repeated

EDI standard data element header record (1Y6)

This record defines the data element header fields being used. The fields are described in Table 28.

Table 28. EDI standard data element header record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
DEID	34-41	8	Char	Data element ID
DETYPE	42-43	2	Char	Type of data element
DEMIN	44-54	11	Char	Minimum length of data element
DEMAX	55-65	11	Char	Maximum length of data element
CODELIST	66-73	8	Char	Code list name
STDDESC	74-123	80	Char	Description of the standard
STDPURP	124-2171	2048	Char	Purpose of the standard
CMTTEXT	2172-3971	1800	Char	Comments

EDI standard data element detail record (1Y7)

This record defines the data element detail fields being used. The fields are described in Table 29.

Table 29. EDI standard data element detail record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
COMPID	34-41	8	Char	Composite data element ID
POSNO	42-47	6	Char	Position number
DEID	48-55	8	Char	Data element ID
DEREQ	56-56	1	Char	Data element required flag
SEGMAX	57-67	11	Char	Number of times segment can be repeated

EDI standard segment notes record (1Y8)

This record defines the segment note fields being used. The fields are described in Table 30.

Table 30. EDI standard segment notes record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
SEGID	34-41	8	Char	Segment ID
POSNO	42-47	6	Char	Position number
NOTETYPE	48-48	1	Char	Type of note
NOTEPARA	49-54	6	Char	Paragraph number
RELATION	55-56	2	Char	Relationship of one data element to another
REL1	57-62	6	Char	Related data element reference number
REL2	63-68	6	Char	Related data element reference number
REL3	69-74	6	Char	Related data element reference number
REL4	75-80	6	Char	Related data element reference number
REL5	81-86	6	Char	Related data element reference number
REL6	87-92	6	Char	Related data element reference number
REL7	92-98	6	Char	Related data element reference number
REL8	99-104	6	Char	Related data element reference number
REL9	105-110	6	Char	Related data element reference number
REL10	111-116	6	Char	Related data element reference number
STDNOTE	117-1140	1024	Char	Text of the note

EDI standard transaction notes record (1Y9)

This record defines the transaction note fields being used. The fields are described in Table 31.

Table 31. EDI standard transaction notes record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
STDTRID	34-41	8	Char	Transaction ID
TABLENO	42-47	6	Char	Table number
POSNO	48-53	6	Char	Position number
NOTETYPE	54-54	1	Char	Type of note
NOTEPARA	55-60	6	Char	Paragraph number
RELATION	61-62	2	Char	Relationship of one data element to another

Table 31. EDI standard transaction notes record (continued)

Tag	Position	Length	Type	Description
REL1	63-68	6	Char	Related data element reference number
REL2	69-74	6	Char	Related data element reference number
REL3	75-80	6	Char	Related data element reference number
REL4	81-86	6	Char	Related data element reference number
REL5	87-92	6	Char	Related data element reference number
STDNOTE	93-1116	1024	Char	Text of the note

EDI standard composite data element notes record (1YA)

This record defines the composite data element note fields being used. The fields are described in Table 32.

Table 32. EDI standard composite data element notes record

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
COMPID	34-41	8	Char	Composite data element ID
POSNO	42-47	6	Char	Position number
NOTETYPE	48-48	1	Char	Type of note
NOTEPARA	49-54	6	Char	Paragraph number
RELATION	55-56	2	Char	Relationship of one data element to another
REL1	57-62	6	Char	Related data element reference number
REL2	63-68	6	Char	Related data element reference number
REL3	69-74	6	Char	Related data element reference number
REL4	75-80	6	Char	Related data element reference number
REL5	81-86	6	Char	Related data element reference number
STDNOTE	87-1110	1024	Char	Text of the note

Exporting and importing data formats

Data formats can be exported/imported using either tagged or fixed format files. The data format set includes:

2W1   Data format dictionary

2W2   Data format record ID information

2W3   Data format leader record

2W4   Data format loop record

2W5   Data format record record

2W6	Data format structure record
2W7	Data format field record
2W8	Data format header details record
2W9	Data format loop details record
2WA	Data format record details record
2WB	Data format structure details record

There are no associated objects for data formats.

The following example shows part of an export/import file data set that contains records for exporting/importing one data format.

```
0C1DD5TST1 011025134023ENU010401000000

2W1  DICTIONARYNAME(MMTHL1_DICTIONARY)  DESC(MAUI MAP TEST - HL1 - SUW)  ALLADFS(N)

2W2  RECORDIDINFONAME(MMTHL1_RECORDID)  DESC(MAUI MAP TEST - HL1 - SUW)  ADFTYPE(0)
     RIDPOS(1)  RIDLENG(3)  RIDTYPE(AN)

2W3  DICTIONARYNAME(MMTHL1_DICTIONARY)  RECORDIDINFONAME(MMTHL1_RECORDID)  FORMAT(MMTHL1)
        DESC(MAUI MAP TEST - HL1 - SUW)  ADFTPIDFLD(TRADINGPART) ADFBEGINRECID(MMTHL1REC)

2W5  DICTIONARYNAME(MMTHL1_DICTIONARY)  RECORDIDINFONAME(MMTHL1_RECORDID)
ADFRECORDNAME(MMTHL1REC)
       DESC(MAUI MAP TEST - HL1 - SUW)  ADFRECORDID(001)

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(MMTHL1RECID)
DESC(MAUI MAP TEST - HL1 - SUW) DATATYPE(AN)  FLDLEN(3)  ADFSENDMAP(0) ADFRECEIVEMAP(0)

2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC) ADF_RMEM_SEQNUM(1)
        ADF_RMEM_MEMBERNAME(MMTHL1RECID) ADF_RMEM_MEMBERTYPE(FIELD)  ADF_RMEM_MAXUSE(1)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(TRADINGPART)
DESC(MAUI MAP TEST - HL1 - SUW) DATATYPE(AN)  FLDLEN(18)  ADFSENDMAP(0) ADFRECEIVEMAP(0)

2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC) ADF_RMEM_SEQNUM(2)
        ADF_RMEM_MEMBERNAME(TRADINGPART) ADF_RMEM_MEMBERTYPE(FIELD)  ADF_RMEM_MAXUSE(1)

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(ACFIELD) DESC(MAUI MAP TEST - HL1 - SUW)
        DATATYPE(AC)  FLDLEN(6)  ADFSENDMAP(0) ADFRECEIVEMAP(0)

2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC) ADF_RMEM_SEQNUM(3)
        ADF_RMEM_MEMBERNAME(ACFIELD) ADF_RMEM_MEMBERTYPE(FIELD)  ADF_RMEM_MAXUSE(1)

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(FILLER) DESC(MAUI MAP TEST - HL1 - SUW)
        DATATYPE(AN)  FLDLEN(1)  ADFSENDMAP(0) ADFRECEIVEMAP(0)

2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC) ADF_RMEM_SEQNUM(4)
        ADF_RMEM_MEMBERNAME(FILLER) ADF_RMEM_MEMBERTYPE(FIELD)  ADF_RMEM_MAXUSE(1)

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(ELIGBEGDATE)
DESC(MAUI MAP TEST - HL1 - SUW) DATATYPE(AN)  FLDLEN(8)  ADFSENDMAP(0) ADFRECEIVEMAP(0)
```

Exporting and importing data formats

```
2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC) ADF_RMEM_SEQNUM(5)
       ADF_RMEM_MEMBERNAME(ELIGBEGDATE) ADF_RMEM_MEMBERTYPE(FIELD)  ADF_RMEM_MAXUSE(1)

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(ELIGENDDATE) DESC(MAUI MAP TEST - HL1 - SUW)
       DATATYPE(AN)  FLDLEN(8)  ADFSENDMAP(0) ADFRECEIVEMAP(0)

2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC) ADF_RMEM_SEQNUM(6)
       ADF_RMEM_MEMBERNAME(ELIGENDDATE) ADF_RMEM_MEMBERTYPE(FIELD)  ADF_RMEM_MAXUSE(1)

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(SUBSCRIBER) DESC(MAUI MAP TEST - HL1 - SUW)
       DATATYPE(AN)  FLDLEN(18)  ADFSENDMAP(0) ADFRECEIVEMAP(0)

2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC) ADF_RMEM_SEQNUM(7)
       ADF_RMEM_MEMBERNAME(SUBSCRIBER) ADF_RMEM_MEMBERTYPE(FIELD)  ADF_RMEM_MAXUSE(1)

2W6  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFSTRUCTNAME(FAMILY) DESC(MAUI MAP TEST - HL1 - SUW)

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(DEPFIRSTNAME) DESC(MAUI MAP TEST - HL1 - SUW)
       DATATYPE(AN)  FLDLEN(9)  ADFSENDMAP(0) ADFRECEIVEMAP(0)

2WB  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_SMEM_STRUCTNAME(FAMILY) ADF_SMEM_SEQNUM(1)
       ADF_SMEM_MEMBERNAME(DEPFIRSTNAME) ADF_SMEM_MEMBERTYPE(FIELD)  ADF_SMEM_MAXUSE(1)

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(DEPLASTNAME) DESC(MAUI MAP TEST - HL1 - SUW)
       DATATYPE(AN)  FLDLEN(9)  ADFSENDMAP(0) ADFRECEIVEMAP(0)

2WB  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_SMEM_STRUCTNAME(FAMILY) ADF_SMEM_SEQNUM(2
       ADF_SMEM_MEMBERNAME(DEPLASTNAME) ADF_SMEM_MEMBERTYPE(FIELD)  ADF_SMEM_MAXUSE(1)

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(DEPDOB) DESC(MAUI MAP TEST - HL1 - SUW)
       DATATYPE(AN)  FLDLEN(8)  ADFSENDMAP(0) ADFRECEIVEMAP(0)

2WB  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_SMEM_STRUCTNAME(FAMILY) ADF_SMEM_SEQNUM(3)
       ADF_SMEM_MEMBERNAME(DEPDOB) ADF_SMEM_MEMBERTYPE(FIELD)  ADF_SMEM_MAXUSE(1)

2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC) ADF_RMEM_SEQNUM(8)
       ADF_RMEM_MEMBERNAME(FAMILY) ADF_RMEM_MEMBERTYPE(STRUCT)  ADF_RMEM_MAXUSE(15)

2W8  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_HMEM_ADFNAME(MMTHL1) ADF_HMEM_SEQNUM(1)
       ADF_HMEM_MEMBERNAME(MMTHL1REC) ADF_HMEM_MEMBERTYPE(REC)  ADF_HMEM_AREA(Detail)
       ADF_HMEM_MAXUSE(1)  ADF_HMEM_INFLOOP(0)

000
```

Data format dictionary record (2W1)

This record defines the data format dictionary being used. The fields are described in
Table 33.

Table 33. Data format dictionary record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
DESC	34-83	50	Char	Description

Table 33. Data format dictionary record (continued)

Tag	Position	Length	Type	Description
CMTTEXT	84-2131	2048	Char	Comments
ALLADFS	2132-2132	1	Char	Include all ADFs

Data format record ID record (2W2)

This record defines the data format record being used. The fields are described in Table 34.

Table 34. Data format record ID record

Tag	Position	Length	Type	Description
RECORDIDINFONAME	4-33	30	Char	Record ID information name
DESC	34-83	50	Char	Description
ADFTYPE	84-84	1	Char	Data format type
RIDPOS	85-90	6	Char	Record ID position
RIDLENG	91-101	11	Char	Record ID length
RIDTYPE	102-103	2	Char	Record ID type
CMTTEXT	104-2151	2048	Char	Comments

Data format header record (2W3)

This record defines the data format header being used. The fields are described in Table 35.

Table 35. Data format header record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
RECORDIDINFONAME	34-63	30	Char	Record ID information name
FORMAT	64-79	16	Char	Data format name
DESC	80-129	50	Char	Description
FILEID	130-137	8	Char	logical name
FILETYPE	138-139	2	Char	File type
ADFTPIDFLD	140-169	30	Char	Field containing trading partner ID
ADFBEGINRECID	170-199	30	Char	Field containing beginning record ID
ADFENDRECID	200-229	30	Char	Field containing ending record ID

Table 35. Data format header record (continued)

Tag	Position	Length	Type	Description
ADFGENRTCDE	230-259	30	Char	Field containing generic routing code
CMTTEXT	260-2307	2048	Char	Comments
INTSNDQUALFLD	2308-2337	30	Char	Field containing interchange sender ID qualifier
INTSNDIDFLD	2338-2367	30	Char	Field containing interchange sender ID
INTRCVQUALFLD	2368-2397	30	Char	Field containing interchange receiver ID qualifier
INTRCVFLD	2398-2427	30	Char	Field containing interchange receiver ID
APPLTPIDFLD	2428-2457	30	Char	Field containing application trading partner nickname
EDITPIDFLD	2458-2487	30	Char	Field containing EDI trading partner nickname
ADFCMDELIM	2488-2488	1	Char	Comma-separated value delimiter
ADFRECDELIM	2489-2490	2	Char	Record delimiter
ADFUNICODE	2491-2506	16	Char	Code page
ADFTXTDELIM	2507-2508	2	Char	Reserved for future use
ADFTXTQUAL	2509-2538	30	Char	Reserved for future use

Data format loop record (2W4)

This record defines the data format loop being used. The fields are described in Table 36.

Table 36. Data format loop record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
RECORDIDINFONAME	34-63	30	Char	Record ID information name
ADFLOOPNAME	64-93	30	Char	Loop name
DESC	94-143	50	Char	Description
CMTTEXT	144-2191	2048	Char	Comments

Data format record record (2W5)

This record defines the data format record being used. The fields are described in Table 37.

Table 37. Data format record record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
RECORDIDINFONAME	34-63	30	Char	Record ID information name
ADFRECORDNAME	64-93	30	Char	Record name
DESC	94-143	50	Char	Description
ADFRECORDID	144-159	16	Char	Record ID
CMTTEXT	160-2207	2048	Char	Comments

Data format structure record (2W6)

This record defines the data format structure being used. The fields are described in Table 38.

Table 38. Data format structure record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADSTRUCTNAME	34-63	30	Char	Structure name
DESC	64-113	50	Char	Description
CMTTEXT	114-2161	2048	Char	Comments

Data format field record (2W7)

This record defines the data format fields being used. The fields are described in Table 39.

Table 39. Data format field record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADFFIELDNAME	34-63	30	Char	Field name
DESC	64-113	50	Char	Description
DATATYPE	114-115	2	Char	Data type
FLDLEN	116-121	6	Char	Field length
ADFMAPCOMMANDS	122-201	80	Char	Mapping command line
ADFSENDMAP	202-202	1	Char	Send map
ADFRECEIVEMAP	203-203	1	Char	Receive map

Exporting and importing data formats

Table 39. Data format field record (continued)

Tag	Position	Length	Type	Description
ADFVALTBLNAME	204-211	8	Char	Code list name
CMTTEXT	212-2259	2048	Char	Comments

Data format header details record (2W8)

This record defines the data format header details being used. The fields are described in Table 40.

Table 40. Data format header details record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADF_HMEM_ADFNAME	34-49	16	Char	Data format name
ADF_HMEM_SEQNUM	50-59	10	Char	Sequence number
ADF_HMEM_MEMBERNAME	60-89	30	Char	Member name
ADF_HMEM_MEMBERTYPE	90-97	8	Char	Member type
ADF_HMEM_AREA	98-113	16	Char	Area value
ADF_HMEM_MAXUSE	114-118	5	Char	Maximum number of occurrences
ADF_HMEM_INFLOOP	119-119	1	Char	Infinite loop flag

Data format loop details record (2W9)

This record defines the data format loop details being used. The fields are described in Table 41.

Table 41. Data format loop details record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADF_LMEM_LOOPNAME	34-63	30	Char	Loop name
ADF_LMEM_SEQNUM	64-73	10	Char	Sequence number
ADF_LMEM_MEMBERNAME	74-103	30	Char	Member name
ADF_LMEM_MEMBERTYPE	104-111	8	Char	Member type
ADF_LMEM_MAXUSE	112-116	5	Char	Maximum number of occurrences
ADF_LMEM_INFLOOP	117-117	1	Char	Infinite loop flag

Data format record details record (2WA)

This record defines the data format record details being used. The fields are described in Table 42.

Table 42. Data format record details record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADF_RMEM_RECORDNAME	34-63	30	Char	Record name
ADF_RMEM_SEQNUM	64-73	10	Char	Sequence number
ADF_RMEM_MEMBERNAME	74-103	30	Char	Member name
ADF_RMEM_MEMBERTYPE	104-111	8	Char	Member type
ADF_RMEM_MAXUSE	112-116	5	Char	Maximum number of occurrences
ADF_RMEM_OCCURSDEPNAME	117-146	30	Char	The field that determines whether this structure should occur

Data format structure details record (2WB)

This record defines the data format structure details being used. The fields are described in Table 43.

Table 43. Data format structure details record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADF_SMEM_STRUCTNAME	34-63	30	Char	Structure name
ADF_SMEM_SEQNUM	64-73	10	Char	Sequence number
ADF_SMEM_MEMBERNAME	74-103	30	Char	Member name
ADF_SMEM_MEMBERTYPE	104-111	8	Char	Member type
ADF_SMEM_MAXUSE	112-116	5	Char	Maximum number of occurrences
ADF_SMEM_OCCUR SDEPNAME	117-146	30	Char	The field that determines whether this structure should occur

Exporting and importing maps

Maps can be exported/imported using tagged or fixed format files. The map set includes:

3T4   Trading partner

3T5   Trading partner send usages/rules

Exporting and importing maps

3T6	Trading partner receive usages/rules
3T7	Data transformation map usage/rule
3V1	Map header record
3V2	Map segment record
3V3	Map element record
3V9	Map application control record
3VA	Map syntax record
3VB	Map local variables record
3VC	Map reference record
3VD	Map nodes record
3VE	Map comments record
BK2	Global variables

The twenty associated objects for exporting maps are listed below. Either maps or usages/rules must be selected:

1. Transaction usages/rules
2. Control string
3. Standard transactions (for DT maps source document definition)
4. Data formats (for DT maps target document definition)umm in
5. Validation tables
6. Translation tables
7. User exit routines
8. Trading partner profile
9. Translation exit routines
10. Network security profile
11. Network profile
12. Network commands profile
13. Envelope profile
14. Envelope standard
15. Maps
16. Conversion of prior-release objects
17. Global variables
18. Validation maps
19. FA maps
20. Embedded maps

Note: In trading partner-to-trading partner translations, WebSphere Data Interchange stores control numbers by ISA Sender Qualifier, ISA Receiver Qualifier, and ISA Receiver, ensuring that the receiver gets a contiguous set of control numbers from every sender.

The following example shows an export/import file that contains tags for a send map and the associated send usage/rule.

```
0C1DD5TST1 011025135724ENU010401000000
3V1  TPTID(MMTHL1)  STDID(X12V3R7)  STDTRID(271)  FORMAT(MMTHL1) DIR(S)
GENREQD(N)
     DESC(MAUI MAP TEST - HL1 - SUW Outbound)
DICTIONARYNAME(MMTHL1_DICTIONARY)
     CMPCHK(N)  BASE(T)
3V2  TPTID(MMTHL1)  TABLE(1)  POS(20)  OCCUR(1)  GENFLAG(N)
3V2  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(1)  DEPTHSEQ(1)
PATH(MMTHL1REC)
 HLNODE(1)  HLPARNODE(1)  HL03(20)  GENFLAG(N)
3V2  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(2)  DEPTHSEQ(3)  HLNODE(2)
 HLPARNODE(1)  HL03(21)  PARHL03(20)  GENFLAG(N)
3V2  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(3)  DEPTHSEQ(5)  HLNODE(3)
 HLPARNODE(1)  HL03(22)  PARHL03(21)  GENFLAG(N)
3V2  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  DEPTHSEQ(6)
PATH(MMTHL1RECFAMILY)  HLNODE(4)  HLPARNODE(1)  HL03(23)  PARHL03(22)
GENFLAG(N)3V2  TPTID(MMTHL1)  TABLE(2)  POS(20)  OCCUR(1)  PARTABLE(2)
 PARPOS(10)
 PAROCCUR(2)  DEPTHSEQ(4)  QUALOCCUR(1)  GENFLAG(N)
3V2  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(1)  PARTABLE(2)  PARPOS(10)
 PAROCCUR(4)  DEPTHSEQ(7)  QUALOCCUR(1)  GENFLAG(N)
3V2  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(2)  PARTABLE(2)  PARPOS(10)
 PAROCCUR(1)  DEPTHSEQ(2)  QUALOCCUR(1)  GENFLAG(N)
3V3  TPTID(MMTHL1)  TABLE(1)  POS(20)  OCCUR(1)  ELEPOS(1)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(0022)
3V3  TPTID(MMTHL1)  TABLE(1)  POS(20)  OCCUR(1)  ELEPOS(2)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(11)
3V3  TPTID(MMTHL1)  TABLE(1)  POS(20)  OCCUR(1)  ELEPOS(3)  ELEOCCUR(1)
 PATH(MMTHL1REC)  ADFFIELDNAME(ACFIELD)  DATEEDIT(-4096)
3V3  TPTID(MMTHL1)  TABLE(1)  POS(20)  OCCUR(1)  ELEPOS(4)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&DATE)
3V3  TPTID(MMTHL1)  TABLE(1)  POS(20)  OCCUR(1)  ELEPOS(5)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&TIME)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(1)  ELEPOS(1)  ELEOCCUR(2)
 DATEEDIT(-4096)  LITVAL(&SET MAPIT 0)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(1)  ELEPOS(1)  ELEOCCUR(3)
 PATH(MMTHL1REC)  ADFFIELDNAME(SUBSCRIBER)  DATEEDIT(-4096)
LITVAL(&IFDATA &SET MAPIT 1)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(1)  ELEPOS(1)  ELEOCCUR(4)
 DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HLID)

3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(1)  ELEPOS(2)
 ELEOCCUR(1) DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HLPID)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(1)  ELEPOS(3)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HCODE)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(1)  ELEPOS(4)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HCHILD)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(2)  ELEPOS(1)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&HLID)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(2)  ELEPOS(2)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&HLPID)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(2)  ELEPOS(3)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&HCODE)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(2)  ELEPOS(4)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&HCHILD)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(3)  ELEPOS(1)  ELEOCCUR(1)
```

```
            DATEEDIT(-4096)  LITVAL(&HLID)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(3)  ELEPOS(2)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&HLPID)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(3)  ELEPOS(3)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&HCODE)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(3)  ELEPOS(4)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&HCHILD)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(1)  ELEOCCUR(2)
 DATEEDIT(-4096)  LITVAL(&SET MAPIT 0)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(1)  ELEOCCUR(3)
 PATH(MMTHL1RECFAMILY)  ADFFIELDNAME(DEPDOB)  DATEEDIT(-4096)
 LITVAL(&IFDATA &SET MAPIT 1)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(1)  ELEOCCUR(4)
 DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HLID)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(2)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HLPID)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(3)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HCODE)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(4)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HCHILD)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(20)  OCCUR(1)  ELEPOS(1)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(AC)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(20)  OCCUR(1)  ELEPOS(2)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(&ACFIELD)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(1)  ELEPOS(1)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(DD)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(1)  ELEPOS(2)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(D)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(1)  ELEPOS(3)  ELEOCCUR(1)
 PATH(MMTHL1RECFAMILY)  ADFFIELDNAME(DEPLASTNAME)  DATEEDIT(-4096)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(1)  ELEPOS(4)  ELEOCCUR(1)
 PATH(MMTHL1RECFAMILY)  ADFFIELDNAME(DEPFIRSTNAME)  DATEEDIT(-4096)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(2)  ELEPOS(1)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(SS)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(2)  ELEPOS(2)  ELEOCCUR(1)
 DATEEDIT(-4096)  LITVAL(S)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(2)  ELEPOS(3)  ELEOCCUR(1)
 PATH(MMTHL1REC)  ADFFIELDNAME(SUBSCRIBER)  DATEEDIT(-4096)
000
```

The following tables define the fields for exporting/importing maps.

Map header record (3V1)

This record defines the map header being used. The fields are described in Table 44.

Table 44. Map header record

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
STDID	20-49	30	Char	EDI standard dictionary name
STDTRID	50-57	8	Char	EDI standard transaction ID
FORMAT	58-73	16	Char	Data format ID

Table 44. Map header record (continued)

Tag	Position	Length	Type	Description
DIR	74-74	1	Char	Map type (send/receive/data transformation) flag
GENREQD	75-75	1	Char	Compile required
DESC	76-125	50	Char	Description
DICTIONARYNAME	126-155	30	Char	Data format dictionary name
CMPCHK	156-156	1	Char	Compliance check flag
BASE	157-157	1	Char	Future use

Map segment record (3V2)

This record defines the map segment being used. The fields are described in Table 45.

Table 45. Map segment record

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
TABLE	20-20	1	Char	Table
POS	21-25	5	Char	Position in table
OCCUR	26-30	5	Char	Occurrence
PARTABLE	31-31	1	Char	Parent table
PARPOS	32-36	5	Char	Parent position
PAROCCUR	37-41	5	Char	Parent occurrence
DEPTHSEQ	42-46	5	Char	Depth sequence number
QUALOCCUR	47-52	6	Char	Qualification occurrence number
QUALELEPOS	53-56	4	Char	Qualified element position
QUALELEREPEATNO	57-61	5	Char	Qualified element repeat number
QUALSUBELEPOS	62-65	4	Char	Qualified sub-element position
PATH	66-320	255	Char	Qualified path
QUALTEXT	321-365	45	Char	Element loop value
HLNODE	366-370	5	Char	Hierarchical loop node number
HLPARNODE	371-375	5	Char	Hierarchical loop parent node number
HL03	376-377	2	Char	HL03 value
PARHL03	378-379	2	Char	HL03 parent value
FLDHL03	380-409	30	Char	HL03 field value
PARFLDHL03	410-439	30	Char	HL03 field parent value
GENFLAG	440-440	1	Char	Generate segment
DOSNIPET	441-441	1	Char	Snipet text
BSNIPET	442-521	80	Char	Before snipet flag

Table 45. Map segment record (continued)

Tag	Position	Length	Type	Description
ASNIPET	522-601	80	Char	After snipet flag
CMTTEXT	602-2649	2048	Char	Segment mapping comments

Map element record (3V3)

This record defines the map element being used. The fields are described in Table 46.

Table 46. Map element record

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
TABLE	20-20	1	Char	Table ID
POS	21-25	5	Char	Position in table
OCCUR	26-30	5	Char	Table occurrence number
ELEPOS	31-35	5	Char	Element position number
SUBELEPOS	36-40	5	Char	Sub-element position number
ELEOCCUR	41-45	5	Char	Element occurrence number
PATH	46-300	255	Char	Path name
ADFFIELDNAME	301-330	30	Char	Data format field name
ELEQUALPOS	331-335	5	Char	Qualifying element's position number
SUBELEQUALPOS	336-340	5	Char	Qualifying sub-element's position number
ELEQUALOCCUR	341-345	5	Char	Qualifying element's occurrence number
QUALTEXT	346-386	45	Char	Element qualifier
TRNTAB	391-398	8	Char	Translation table
USEREXIT	699-406	8	Char	User exit ID
VALTAB	407-414	8	Char	Code list ID
DATEEDIT	415-416	2	Char	Date edit field
SUBPOS	417-422	6	Char	Substring or concatenation position
SUBLEN	423-428	6	Char	Substring or concatenation length
ACCUM1	429-430	2	Char	Accumlator1
ACTION1	431-432	2	Char	Accumlator1 action
ACCUM2	433-434	2	Char	Accumlator2
ACTION2	435-436	2	Char	Accumlator2 action
ACCUM3	437-438	2	Char	Accumlator3
ACTION3	439-440	2	Char	Accumlator3 action

Table 46. Map element record (continued)

Tag	Position	Length	Type	Description
ACCUM4	441-442	2	Char	Accumlator4
ACTION4	443-444	2	Char	Accumlator4 action
LITVAL	445-524	80	Char	Code Snippet
CMTTEXT	525-2572	2048	Char	Element comments
ELEREPEATNO	2573-2577	5	Char	Element repeat occurrence number
QUALPATH	2578-2832	255	Char	Qualifying path
UNIQUEID	2833-2838	6	Char	Mapping unique ID

Map application control record (3V9)

This record defines the map application control being used. The fields are described in Table 47.

Table 47. Map application control record

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-20	1	Char	Unique ID
SEQUENCE	21-21	1	Char	Sequence
PATH	22-276	255	Char	Path name
ADFFIELDNAME	277-306	30	Char	Data format field name
LENGTH	307-311	5	Char	Field length

Map syntax record (3VA)

This record defines the map syntax being used. The fields are described in Table 48.

Table 48. Map syntax record

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-24	5	Char	Unique ID
BASE	25-25	1	Char	Future use
SEQNO	26-30	6	Char	Sequence number
SYNTAX	31-31	1	Char	Syntax type
DICTIONARYNAME	32-61	30	Char	Dictionary name
DOCNAME	62-91	30	Char	Document definition name

Exporting and importing maps

Map local variables record (3VB)

This record defines the map local variables being used. The fields are described in Table 49.

Table 49. Map local variables record

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
VARNAME	20-49	30	Char	Variable name
DISPNAME	50-79	30	Char	Displayed variable name
DESC	80-129	50	Char	Description
DATATYPE	130-130	1	Char	Data type
SCOPE	131-131	1	Char	Scope
DEMAX	132-136	5	Char	Maximum length
INITVAL	137-168	32	Char	Initial value

Map reference record (3VC)

This record defines the map reference being used. The fields are described in Table 50.

Table 50. Map reference record

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-25	6	Char	Unique ID
TYPE	26-26	1	Char	Type
PARENTID	27-37	11	Char	Syntax ID
SYNTAX	38-38	1	Char	Syntax type
DICTIONARYNAME	39-68	30	Char	Dictionary name
FORMAT	69-98	30	Char	Document definition name
PATH	99-353	255	Char	Path

Map nodes record (3VD)

This record defines the map nodes being used. The fields are described in Table 51.

Table 51. Map nodes record

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-25	6	Char	Unique ID
PARENTID	26-36	11	Char	Parent's unique ID
SEQNO	37-47	11	Char	Sequence

Table 51. Map nodes record (continued)

Tag	Position	Length	Type	Description
TYPE	48-48	1	Char	Map type
SUBTYPE	49-49	1	Char	Map sub type
ASSOCFWD	50-55	6	Char	Associated forward
ASSOCBKD	56-61	6	Char	Associated backward
POSNO	62-70	9	Char	Position number
EDIT	71-71	1	Char	Map edit
REFID	72-77	6	Char	Map reference ID
REPEATS	78-78	1	Char	Node repeats
QUALIFIED	79-79	1	Char	Qualified flag
CMTTEXT	80-2128	2048	Char	Comment

Map commands record (3VE)

This record defines the map commands being used. The fields are described in Table 52.

Table 52. Map commands record

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-25	6	Char	Unique ID
PARENTID	26-36	11	Char	Node's unique ID
SEQNO	37-47	11	Char	Sequence
VERSION	48-53	6	Char	Version
COMMAND	54-2101	2048	Char	Map command

Global variables details record (BK2)

This record defines the global variables being used. The fields are described in Table 53.

Table 53. Global variables details record

Tag	Position	Length	Type	Description
VARNAME	4-33	30	Char	Variable name
DISPNAME	34-63	30	Char	Displayed variable name
DESC	64-113	50	Char	Description
DATATYPE	114-114	1	Char	Data type
SCOPE	115-115	1	Char	Scope
DEMAX	116-120	5	Char	Maximum length

Table 53. Global variables details record (continued)

INITVAL	121-152	32	Char	Initial value

Send usage record (3T5)

This record defines the send usage/rule being used. The fields are described in Table 54.

Table 54. Trading partner send usage record

Tag	Position	Length	Type	Description
INTPID	4-38	35	Char	Internal trading partner ID
FORMAT	39-54	16	Char	Data format ID
TPTID	55-70	16	Char	Trading partner mapping transaction ID
TPNICKN	71-86	16	Char	Trading partner nickname
ENVTYPE	87-87	1	Char	EDI standard envelope type
STDPROF	88-95	8	Char	EDI standard profile name
APPSNDID	96-130	35	Char	Application sender ID
APPRECID	131-165	35	Char	Application receiver ID
APPASSWD	166-179	14	Char	Application password
PSTTRXIT	180-187	8	Char	Post-translation exit routine name
ACTIVE	188-188	1	Char	Active usage for this INTPID/FORMAT combination
ACKREQ	189-189	1	Char	Acknowledgment expected flag
TESTIND	190-190	1	Char	Test indicator
LOGAPP	191-191	1	Char	Log application data
ENFORCEHIER	192-192	1	Char	Enforce structure hierarchy
STRUCTDATACHK	193-193	1	Char	Structure must produce data
TRVALVL	194-194	1	Char	Validation level
TRERLVL	195-195	1	Char	Acceptable error level
GRPSECPR	196-203	8	Char	Group network security profile name
GRPENCKY	204-219	16	Char	Group encryption key name
GRPAUTKY	220-235	16	Char	Group authentication key name
TRNSECPR	236-243	8	Char	Trans network security profile name
TRNENCKY	244-259	16	Char	Trans encryption key name
TRNAUTKY	260-275	16	Char	Trans authentication key name
APPLTPID	276-291	16	Char	Application trading partner nickname

Table 54. Trading partner send usage record (continued)

Tag	Position	Length	Type	Description
ALPHANUM	292-299	8	Char	Alternate table for ALPHANUM
CHARSET	300-307	8	Char	Alternate table for CHARSET
CNTRX	308-308	1	Char	Control numbers utilization flag

Receive usage record (3T6)

This record defines the receive usage/rule being used. The fields are described in Table 55.

Table 55. Trading partner receive usage record

Tag	Position	Length	Type	Description
TPNICKN	4-19	16	Char	Trading partner nickname
STDTRID	20-27	8	Char	EDI standard transaction ID
TPTID	28-43	16	Char	Map name
APSNDRCV	44-78	35	Char	Application sender/receiver ID
AGENCY	79-86	8	Char	Responsible agency code
VER	87-94	8	Char	Version
REL	95-102	8	Char	Release
SNDRCVFL	103-103	1	Char	Sender or receiver flag
INTPID	104-138	35	Char	Internal trading partner ID
STDPROF	139-146	8	Char	Standard profile name
APPASSWD	147-160	14	Char	Application password
PRETRXIT	161-168	8	Char	Pre-translation exit routine
FILEID	169-176	8	Char	Logical receiving data set name
FILETYPE	177-178	2	Char	Type of receive file
ACTIVE	179-179	1	Char	Active usage flag for this trading partner
ACKTRN	180-187	8	Char	Acknowledgment transaction ID
TESTIND	188-188	1	Char	Test indicator
LOGAPP	189-189	1	Char	Log application data
OVERLAYCHK	190-190	1	Char	Consider data overlay an error
UNEXPFLDCHK	191-191	1	Char	Consider unexpected field an error
UNEXPSEGCHK	192-192	1	Char	Consider unexpected segment an error
SWITCHROUTING	193-193	1	Char	Switch application routing IDs on functional acknowledgment
TRVALVL	194-194	1	Char	Validation level

Table 55. Trading partner receive usage record (continued)

Tag	Position	Length	Type	Description
TRERLVL	195-195	1	Char	Acceptable error level
INTFA	196-196	1	Char	Internal functional acknowledgment switch
APPLTPID	197-212	16	Char	Application trading partner nickname
ALPHANUM	213-220	8	Char	Alternate table for ALPHANUM
CHARSET	221-228	8	Char	Alternate table for CHARSET
CNTRX	229-229	1	Char	Control numbers utilization flag

Data transformation map rule record (3T7)

This record defines the data transformation map being used. The fields are described in Table 56.

Table 56. Data transformation map rule record

Tag	Position	Length	Type	Description
MAPID	4-20	16	Char	Map name
DOCID	21-49	30	Char	Document definition name
DICTIONARYNAME	50-79	30	Char	Dictionary name
DESCRIPT	80-109	30	Char	Description
PROCESS	110-149	40	Char	Process ID
SENDTPID	150-165	16	Char	Sending trading partner nickname
RECVTPID	166-181	16	Char	Receiving trading partner nicknamme
TESTIND	182-182	1	Char	Test indicator
ENVTYPE	183-183	1	Char	EDI standard envelope type
STDPROF	184-191	8	Char	EDI standard profile name
APPSNDID	192-226	35	Char	Application sender ID
APPRECID	227-261	35	Char	Application receiver ID
APPASSWD	262-275	14	Char	Application password
SYNTAX	276-278	3	Char	Syntax type
ACTIVE	279-279	1	Char	Active usage for this INTPID/FORMAT combination
ACKREQ	280-280	1	Char	Acknowledgment expected flag
INBVALLVL	281-281	1	Char	Inbound validation level
OUTBVALLVL	282-282	1	Char	Outbound validation level
FAENVTYPE	283-283	1	Char	Functional acknowledgment envelope type

Table 56. Data transformation map rule record (continued)

Tag	Position	Length	Type	Description
FAENVPROF	284-291	8	Char	Functional acknowledgment envelope profile name
GRPLVLFA	292-292	1	Char	Group level functional acknowledgment
TRERLVL	293-293	1	Char	Acceptable error level
INBALPHANUM	294-301	8	Char	Inbound alternate table for ALPHANUM
INBCHARSET	302-309	8	Char	Inbound alternate table for CHARSET
OUTBALPHANUM	310-317	8	Char	Outbound alternate table for ALPHANUM
OUTBCHARSET	318-325	8	Char	Outbound alternate table for CHARSET
CNTRX	326-326	1	Char	Control numbers utilization flag
FILEID	327-334	8	Char	Logical application data set name
FILETYPE	335-336	2	Char	Type of application file
ACKDOCID	337-366	30	Char	Acknowledgment document ID
INBVALMAP	367-396	30	Char	Inbound validation map ID
OUTBVALMAP	397-426	30	Char	Outbound validation map ID

Exporting and importing table definitions

Table definitions can be exported/imported using either tagged or fixed format files. The table definitions set includes:

B1   Table definitions

B2   Table entry

The following example shows an export/import file that contains tags for table definitions.

```
0C1SMITH    940419161616ENU010103000000

8B1TBLID(AAAAA)  DATECRT(940325)  TBLDSC(dd)  PUBAUTH(ALL)  FLAGS(00)

8B2TBLID(AAAAA)  VAR1(A)

000
```

The following tables define the fields for exporting/importing tables.

Export/Import table definition (B1)

Table 57 defines the fields for exporting and importing table definitions.

Table 57. Table definition fields for export and import

Tag	Position	Length	Type	Description
TBLID	4-11	8	Char	Table ID
DATECRT	12-17	6	Char	Date created
TBLDSC	18-52	35	Char	Table description
PUBAUTH	53-58	6	Char	Authority
OWNER	59-66	8	Char	Owner
FLAGS	67-68	2	Char	Flags (always **00**)
KEYID	69-86	18	Char	Key field ID
KEYLN	87-92	6	Char	Key field length
RECLN	93-98	6	Char	Record length
NUMFLDS	99-104	6	Char	Number of fields in a table row
TBLTYPE	105-105	1	Char	Table type
FLD1ID	106-123	18	Char	Field 1 ID
FLD1OFF	124-129	6	Char	Field 1 offset
FLD1LEN	130-132	3	Char	Field 1 length
FLD1TYPE	133-134	2	Char	Field 1 data type
FLD1DSC	135-169	35	Char	Field 1 description
FLD2ID	170-187	18	Char	Field 2 ID
FLD2OFF	188-193	6	Char	Field 2 offset
FLD2LEN	194-196	3	Char	Field 2 length
FLD2TYPE	197-198	2	Char	Field 2 data type
FLD2DSC	199-233	35	Char	Field 2 description
TAG	234-241	8	Char	Future index tag

Export/Import table definition (B1) field descriptions

Table ID (TBLID):   A name for referring to the table.

Date created (DATECRT):   The date this table was created.

Table description (TBLDSC):   A brief description of this table.

Authority (PUBAUTH):   For WebSphere Data Interchange use only.

Owner (OWNER):   The user ID of the owner of this table.

Flags (FLAGS):   Always set to 00.

Key ID (KEYID): The ID of the field used as the key for performing lookups. Validation and type **T** translation tables use the first (or only) field as the key field. Type **R** translation tables use the second field.

Key field length (KEYLN): The length of the key field.

Record length (RECLN): The length of a row in this table.

Number of fields (NUMFLDS): The total number of fields in a row in this table. Validation tables have one field. Translation tables have two.

Table type (TBLTYPE): The type of table. Valid values are:

T Translation table. Contains pairs of values: the value to be translated and the value to be substituted for it. For example, you can translate a purchaser's part number to a supplier's part number before sending a purchase order. The table is organized with the application value as the key value and the EDI standard or trading partner value as data. With this type of table, multiple application values can translate to the same EDI standard value, but each EDI standard value will translate to one and only one application value. This table gives the best performance when translating from an application to an EDI standard.

R Translation table. Serves the same purpose as the type **T** table but is organized with the EDI standard value as the key value and the application value as data. With this type of table, multiple EDI standard values can translate to the same application value, but each application value will translate to one and only one EDI standard value. This table gives the best performance when translating from an EDI standard to an application.

V Validation table. This table contains a list of acceptable values for a field. It can contain several values or only one value.

Field 1 ID (FLD1ID): The ID of the first column in a row of this table.

Field 1 offset (FLD1OFF): The offset into a row of this table where the first field begins.

Field 1 length (FLD1LEN): The maximum number of characters one entry in this column will contain. The maximum is **35**. For translation tables (data type T) the maximum length for Field 1 and FIeld 2 combined is 68 characters. Include decimal points and integer signs as part of the data length. For example, the value -12.34 requires a length of 6.

Field 1 data type (FLD1TYPE): The data type for entries in this column. Valid values are:

CH Character. Any combination of characters.

R Real. A numeric field that requires a decimal point for fractional values. The decimal point is optional for integers. A sign (+ or -) is optional.

Field 1 description (FLD1DSC): A brief description of this field.

Exporting and importing table definitions

Field 2 ID (FLD2ID): The ID of the second column in a row of this table (if there is one).

Field 2 offset (FLD2OFF): The offset into a row of this table where the second field begins.

Field 2 length (FLD2LEN): The maximum number of characters one entry in this column will contain. The maximum is **35**. For translation tables (data type T) the maximum length for Field 1 and FIeld 2 combined is 68 characters. Include decimal points and integer signs as part of the data length. For example, the value -12.34 requires a length of 6.

Field 2 data type (FLD2TYPE): The data type for entries in this column. Valid values are:

CH   Character. Any combination of characters.

R   Real. A numeric field that requires a decimal point for fractional values. The decimal point is optional for integers. A sign (+ or -) is optional.

Field 2 description (FLD2DSC): A brief description of this field.

Tag (TAG): Reserved for future use.

Export/Import table entry (B2)

Table 58 defines the tags and lengths for exporting/importing table entries.

Table 58. Table entry export import tags and lengths

Tag	Position	Length	Type	Description
TBLID	4-11	8	Char	Table ID
VAR1	12-Variable	Variable	Char	Translate FROM field value
VAR2	Variable	Variable	Char	Translate TO field value

Table entry (B2) field descriptions

Table ID (TBLID): A name for referring to the table.

VAR1: For validation tables, the valid value to be entered into the table. For translation tables, the application value (user-defined) to be associated with a value from the EDI standard.

VAR2: Applies only for translation tables. The EDI standard value to be associated with an application value (user-defined).

Exporting and importing XML records

XML data definitions can be exported/imported using either tagged or fixed format files. The XML definitions set includes:

AJ1	XML dictionary record
AJ2	XML DTD header record
AJ3	XML DTD detail record

The following example shows an export/import file that contains tags for XML definitions.

```
0C1DD5TST1 011025135847ENU010401000000
AJ1  DICTIONARYNAME(TESTS)  DESC(Test DTDs)  ALLDTDS(Y)
AJ2  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)
     DESC(Order with Sender/Receiver)
     DTDROOTELEM(OrderSR)  DTDSENDQUALELEM(OrderSRHeaderSenderQualifier)
     DTDSENDIDELEM(OrderSRHeaderSenderId)
     DTDRECVQUALELEM(OrderSRHeaderReceiverQualifier)
     DTDRECVIDELEM(OrderSRHeaderReceiverId)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(1) DTDDATA(<??xml
     version="1.0" encoding="ISO-8859-1"??>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(2)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(3) DTDDATA(<!-- @version: -->)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(4)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(5) DTDDATA(<!ELEMENT
     OrderSR (Header, DetailLoop*, Trailer?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(6)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(7) DTDDATA(<!ELEMENT Header
      (PONum, PODate, Sender, Receiver?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(8)
DTDDATA(<!ATTLIST Header typecode CDATA #REQUIRED>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(9)

AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(10)
DTDDATA(<!ELEMENT PONum (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(11) DTDDATA(<!ELEMENT PODate (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(12)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(13)
DTDDATA(<!ELEMENT Sender (Id, Qualifier?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(14)
DTDDATA(<!ELEMENT Receiver (Id, Qualifier?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(15)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(16) DTDDATA(<!ELEMENT Id (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(17) DTDDATA(<!ELEMENT Qualifier (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(18)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(19) DTDDATA(<!ELEMENT
     DetailLoop (ItemNumber, SubDetail+?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(20)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(21)
DTDDATA(<!ELEMENT ItemNumber (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(22)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(23) DTDDATA(<!ELEMENT SubDetail
     (Description, Quantity, UnitPrice?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(24)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(25)
DTDDATA(<!ELEMENT Description (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(26) DTDDATA(<!ELEMENT Quantity (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(27) DTDDATA(<!ELEMENT UnitPrice (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(28)
```

Exporting and importing XML records

```
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(29) DTDDATA(<!ELEMENT
     Trailer (ItemCount, TotalBucks?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(30)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(31) DTDDATA(<!ELEMENT ItemCount (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(32)
DTDDATA(<!ELEMENT TotalBucks (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS) DOCNAME(POXML5SR) SEQNO(33)
000
```

The following tables define the fields for exporting/importing XML definitions.

XML dictionary record (AJ1)

This record defines the XML dictionary being used. The fields are described in Table 59.

Table 59. XML dictionary record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	XML dictionary name
DESC	34-83	50	Char	Description
ALLDTDS	84-84	1	Char	All XML DTDs flag

XML DTD header record (AJ2)

This record defines the XML DTD header being used. The fields are described in Table 60.

Table 60. XML DTD header record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	XML dictionary name
DOCNAME	34-63	30	Char	XML DTD Name
DESC	64-113	50	Char	Description
DTDROOTELEM	114-177	64	Char	Root element
DTDSENDQUALELEM	178-432	255	Char	Element containing sender qualifier
DTDSENDIDELEM	433-687	255	Char	Element containing sender ID
DTDSENDALIASTBL	688-695	8	Char	Sender alias table
DTDRECVQUALELEM	696-950	255	Char	Element containing receiver qualifier
DTDRECVIDELEM	951-1205	255	Char	Element containing receiver ID
DTDRECVALIASTBL	1206-1213	8	Char	Receiver alias table
DTDLOC	1214-1363	150	Char	Source location of DTD

XML DTD details record (AJ3)

This record defines the XML DTD details being used. The fields are described in Table 61 on page 263.

Table 61. XML DTD details record

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	XML dictionary name
DOCNAME	34-63	30	Char	XML DTD Name
SEQNO	64-74	11	Char	Data sequence number
DTDCONT	75-75	1	Char	Continuation flag
DTDDATA	76-330	255	Char	DTD data

Additional profile layouts

The sixteen additional profile layouts defined in the following tables can be exported or imported using either tagged or fixed format files. These profile layouts are:
- "Mailbox (requestor) profile (REQPROF-P2)"
- "Network security profile (SECUPROF-P2)" on page 264
- "Network profile (NETPROF-P2)" on page 265
- "Network commands profile (NETOP-P2)" on page 266
- "Activity log profile (ACTLOGS-P2)" on page 266
- "Application defaults profile (APPDEFS-P2)" on page 266
- "User exit profile (ADAMCTL-P2)" on page 267
- "Language profile (LANGPROF-P2)" on page 268
- "EDIFACT (E envelope) profile (E-P2)" on page 269
- "ICS (I envelope) profile (I-P2)" on page 270
- "UN/TDI (T envelope) profile (T-P2)" on page 271
- "UCS (U envelope) profile (U-P2)" on page 273
- "X12 (X envelope) profile (X-P2)" on page 273
- "Continuous receive profile (CONTRECV-P2 for CICS only)" on page 275
- "CICS performance profile (SYSPROF-P2 for CICS only)" on page 276
- "MQSeries queue profile (MQSERIES-P2)" on page 276

Mailbox (requestor) profile (REQPROF-P2)

This record defines the mailbox (requestor) settings being used. The fields are described in Table 62.

Table 62. Mailbox (requestor) profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **REQPROF**
REQID	12-27	16	Char	Mailbox (requestor) ID
NETID	28-35	8	Char	Network ID
ACCTID	36-67	32	Char	Network account number
USERID	68-99	32	Char	Network user ID
NETPASS	100-115	16	Char	Network password
MSGCLS	116-123	8	Char	Network message user class
FILEID	124-131	8	Char	Transaction receive logical name
NETCLS	132-132	1	Char	Network classification

Additional profile layouts

Table 62. Mailbox (requestor) profile (continued)

Tag	Position	Length	Type	Description
NETCHG	133-133	1	Char	Network charges code
NETACK	134-134	1	Char	Network acknowledgment code
NETVCHK	135-135	1	Char	Validate destination
NETRETN	136-138	3	Char	Message retention
NETEDIO	139-139	1	Char	EDI processing option
NETEDIP	140-140	1	Char	EDI processing override
STGFMTOV	141-141	1	Char	Storage format override
STGFMT	142-142	1	Char	Storage format
NETCMDS	143-150	8	Char	Member name of network commands file
TIMEOUT	151-154	4	Dec	Command line timeout value
NETACKPG	155-162	8	Char	Program to handle network acknowledgments from remote network
ALTNETPH	163-194	32	Char	Alternate dial connection phone number
COMPRESS	195-195	1	Char	Compression
PRIORITY	196-196	1	Char	Delivery priority
DESCRIPT	197-226	30	Char	Member description

Network security profile (SECUPROF-P2)

This record defines the network security settings being used. The fields are described in Table 63.

Table 63. Network security profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **SECUPROF**
SECID	12-19	8	Char	Security ID
SECORIGN	20-35	16	Char	Security originator
SECRECIP	36-51	16	Char	Security recipient
AUTTYPE	52-52	1	Char	Authorization type
AUTCODE	53-53	1	Char	Authorization code
ENCTYPE	54-54	1	Char	Encryption code
FILTYPE	55-55	1	Char	Filtering type
ENCPROG	56-63	8	Char	Encryption program
AUTPROG	64-71	8	Char	Authentication program
COMPROG	72-79	8	Char	Compression program

Table 63. Network security profile (continued)

Tag	Position	Length	Type	Description
FILPROG	80-87	8	Char	Filtering program
BUFSIZE	88-92	5	Dec	Buffer size
DESCRIPT	93-122	30	Char	Description

Network profile (NETPROF-P2)

This record defines the network settings being used. The fields are described in Table 64.

Table 64. Network profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **NETPROF**
NETID	12-19	8	Char	Network ID
NETNAME	20-49	30	Char	Network name
COMMRTN	50-57	8	Char	Communication routine name
NETPROG	58-65	8	Char	Network send/receive program name
NETPARM	66-122	57	Char	Network program parameters
CMDINDD	123-130	8	Char	Network program command input file name
CMDRECLN	131-134	4	Dec	Network command record length
QFILEDD	135-142	8	Char	TD queue file name
QRECLEN	143-146	4	Dec	Transaction data record length
TIMEZONE	147-151	5	Char	Time zone
SYSTYPE	152-159	8	Char	System type
SYSLVL	160-163	4	Char	System level
MSGTXTH	164-164	1	Char	Message text header character
CMDOUTDD	165-172	8	Char	Network program command output file name
MSGHNDLR	173-180	8	Char	Program to process messages
NETSEQ	181-185	5	Char	Sequence number for network
NETAKFILE	186-193	8	Char	File for network acknowledgements
NETPHONE	194-225	32	Char	Dial connection phone number
SCRIPT	226-233	8	Char	Communication script name
DESCRIPT	234-263	30	Char	Description

Network commands profile (NETOP-P2)

This record defines the network commands being used. The fields are described in Table 65.

Table 65. Network operations profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **NETOP**
NETID	12-19	8	Char	Network ID
NETOP	20-27	8	Char	Network command
NETFS	28-35	8	Char	Operation sequence number
BLKNM	36-43	8	Char	Block name
BLKPOS	44-47	4	Dec	Block position
CMDSEQ	48-51	4	Dec	Command line sequence number
CMDPOS	52-55	4	Dec	Command field position
CMDLEN	56-59	4	Dec	Command field length
CMDVAL	60-117	58	Char	Command field value
DESCRIPT	118-147	30	Char	Description

Activity log profile (ACTLOGS-P2)

This record defines the activity log settings being used. The fields are described in Table 66.

Table 66. Activity log profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **ACTLOGS**
APPLID	12-19	8	Char	Application ID
LOGFLNM	20-27	8	Char	Logical name of file
PRGNAME	28-35	8	Char	File handler program name (EDIELAD)
LANG	36-36	1	Char	Language type of log handler (always C)
LOGFLAG	37-38	2	Char	Log profile activity
DESCRIPT	39-68	30	Char	Description

Application defaults profile (APPDEFS-P2)

This record defines the application default settings being used. The fields are described in Table 67 on page 267.

Table 67. Application defaults profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **APPDEFS**
APPLICID	12-19	8	Char	Application ID
MEMBER	20-27	8	Char	ACTLOGS profile name
MRFLAG	28-28	1	Char	Management reporting active
TSFLAG	29-29	1	Char	Transaction Store active
TIFLAG	30-30	1	Char	Transaction image wanted
MONITOR	31-38	8	Char	CICS performance monitor exit name
USFLAG	39-39	1	Char	Test transaction with production usage
LOGENV	40-40	1	Char	Log EDI standard data
FAIFLAG	41-41	1	Char	Functional acknowledgment image
ELFLAG	42-42	1	Char	Envelope log active
DESCRIPT	43-72	30	Char	Description
ALPHANUM	73-80	8	Char	Name of override ALPHANUM table
CHARSET	81-88	8	Char	Name of override CHARSET table
CTRLYY	89-90	2	Char	Century control year

User exit profile (ADAMCTL-P2)

This record defines the user exits being used. The fields are described in Table 68.

Table 68. User exit profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **ADAMCTL**
LOGPGNM	12-19	8	Char	Logical name of program
LOADMOD	20-27	8	Char	Load module name
LANG	28-28	1	Char	Language used in user program
ADUEFLD	29-29	1	Char	Field exit type
ADUEPST	30-30	1	Char	Post-translate exit type
ADUEPRE	31-31	1	Char	Pre-translate exit type
ADUEENC	32-32	1	Char	Encryption exit type
ADUEAUT	33-33	1	Char	Authentication exit type
ADUECMP	34-34	1	Char	Compression exit type
ADUEFLT	35-35	1	Char	Filtering exit type
ADUEMNT	36-36	1	Char	Monitor exit type
ADUECOM	37-37	1	Char	Communication exit type
ADUEMSG	38-38	1	Char	Message process exit type
ADUEPTP	39-39	1	Char	Point-to-point exit type
ADUEENV	40-40	1	Char	Envelope exit type

Additional profile layouts

Table 68. User exit profile (continued)

Tag	Position	Length	Type	Description
DESCRIPT	41-70	30	Char	Description

Language profile (LANGPROF-P2)

This record defines the language settings being used. The fields are described in Table 69.

Table 69. Language profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **LANGPROF**
LANGID	12-17	6	Char	Language ID
CPAGE	18-22	5	Char	Code page ID
DMASK	23-32	10	Char	Date edit/display mask
TMASK	33-40	8	Char	Time edit/display mask
DECNOT	41-41	1	Char	Edit/display decimal notation
SIGN	42-43	2	Char	Preferred negative sign (display)
FOLD	44-44	1	Char	Substitute for non-displayed character
DESCRIPT	45-74	30	Char	Description

EDIFACT (E envelope) profile (E-P2)

This record defines the EDIFACT envelope settings being used. The fields are described in Table 70.

Table 70. EDIFACT (E envelope) profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **E**
EDIFKEY	12-19	8	Char	Profile name
UNB01	20-23	4	Char	Syntax identifier
UNB02	24-24	1	Char	Syntax version number
UNB03	25-59	35	Char	Sender ID
UNB04	60-63	4	Char	Sender ID qualifier
UNB05	64-77	14	Char	Reverse routing address
UNB06	78-112	35	Char	Recipient ID
UNB07	113-116	4	Char	Recipient ID qualifier
UNB08	117-130	14	Char	Routing address
UNB09	131-136	6	Char	Interchange date
UNB10	137-140	4	Char	Interchange time
UNB11	141-154	14	Char	Interchange control reference
UNB12	155-168	14	Char	Password
UNB13	169-170	2	Char	Password qualifier
UNB14	171-184	14	Char	Application reference
UNB15	185-185	1	Char	Processing priority
UNB16	186-186	1	Char	Acknowledgment request
UNB17	187-221	35	Char	Communications agreement ID
UNB18	222-222	1	Char	Test indicator
UNZ01	223-228	6	Char	Interchange control count
UNZ02	229-242	14	Char	Interchange control reference
UNG01	243-248	6	Char	Functional group ID
UNG02	249-283	35	Char	Sender ID
UNG03	284-287	4	Char	Sender ID qualifier
UNG04	288-322	35	Char	Recipient ID
UNG05	323-326	4	Char	Recipient ID qualifier
UNG06	327-332	6	Char	Functional group date
UNG07	333-336	4	Char	Functional group time
UNG08	337-350	14	Char	Functional group reference number
UNG09	351-352	2	Char	Controlling agency
UNG10	353-355	3	Char	Message version
UNG11	356-358	3	Char	Message release

Additional profile layouts

Table 70. EDIFACT (E envelope) profile (continued)

Tag	Position	Length	Type	Description
UNG12	359-364	6	Char	Association assigned
UNG13	365-378	14	Char	Application password
UNE01	379-384	6	Char	Number of messages
UNE02	385-398	14	Char	Functional group reference number
UNH01	399-412	14	Char	Message reference number
UNH02	413-418	6	Char	Message type
UNH03	419-421	3	Char	Message version number
UNH04	422-424	3	Char	Message release number
UNH05	425-426	2	Char	Controlling agency
UNH06	427-432	6	Char	Association assigned
UNH07	433-467	35	Char	Common access reference
UNH08	468-469	2	Char	Sequence of transfer
UNH09	470-470	1	Char	First and last message in sequence
UNT01	471-476	6	Char	Number of segments in message
UNT02	477-490	14	Char	Message reference number
DESCRIPT	491-520	30	Char	Description

ICS (I envelope) profile (I-P2)

This record defines the ICS envelope settings being used. The fields are described in Table 71.

Table 71. ICS (I envelope) profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **I**
ICSKEY	12-19	8	Char	Profile name
ICS01	20-21	2	Char	Subelement separator
ICS02	22-25	4	Char	Control standards ID
ICS03	26-30	5	Char	Control version number
ICS04	31-32	2	Char	Sender ID qualifier
ICS05	33-47	15	Char	Information sender ID
ICS06	48-49	2	Char	Receiver ID qualifier
ICS07	50-64	15	Char	Information receiver ID
ICS08	65-70	6	Char	Interchange date
ICS09	71-74	4	Char	Interchange time
ICS10	75-83	9	Char	Interchange control number
ICE01	84-89	6	Char	Number of groups
ICE02	90-98	9	Char	Interchange control number

Table 71. ICS (I envelope) profile (continued)

Tag	Position	Length	Type	Description
GS01	99-100	2	Char	Functional group ID
GS02	101-115	15	Char	Application sender code
GS03	116-130	15	Char	Application receiver code
GS04	131-136	6	Char	Functional group date
GS05	137-144	8	Char	Functional group time
GS06	145-153	9	Char	Functional group control number
GS07	154-155	2	Char	Responsible agency code
GS08	156-167	12	Char	Version/release/industry ID
GE01	168-173	6	Char	Number of included sets
GE02	174-182	9	Char	Functional group control number
ST01	183-185	3	Char	Transaction set ID
ST02	186-194	9	Char	Transaction set control number
SE01	195-204	10	Char	Number of included segments
SE02	205-213	9	Char	Transaction set control number
DESCRIPT	214-243	30	Char	Description

UN/TDI (T envelope) profile (T-P2)

This record defines the UN/TDI envelope settings being used. The fields are described in Table 72.

Table 72. UN/TDI (T envelope) profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **T**
TDIKEY	12-19	8	Char	Profile name
STX01	20-23	4	Char	Syntax identifier
STX02	24-24	1	Char	Syntax version number
STX03	25-38	14	Char	Sender code
STX04	39-73	35	Char	Sender name
STX05	74-87	14	Char	Recipient code
STX06	88-122	35	Char	Recipient name
STX07	123-128	6	Char	Interchange date
STX08	129-134	6	Char	Interchange time
STX09	135-148	14	Char	Interchange control reference
STX10	149-162	14	Char	Recipient's reference/password
STX11	163-176	14	Char	Application reference
STX12	177-177	1	Char	Transmission priority code
END01	178-182	5	Char	Total number of messages

Additional profile layouts

Table 72. UN/TDI (T envelope) profile (continued)

Tag	Position	Length	Type	Description
MHD01	183-194	12	Char	Message reference
MHD02	195-200	6	Char	Message type
MHD03	201-201	1	Char	Message version
MHD04	202-236	35	Char	Common access reference
MHD05	237-238	2	Char	Sequence of transfers
MHD06	239-239	1	Char	First and last transfers
MTR01	240-243	4	Char	Number of segments in message
DESCRIPT	244-273	30	Char	Description

UCS (U envelope) profile (U-P2)

This record defines the UCS envelope settings being used. The fields are described in Table 73.

Table 73. UCS (U envelope) profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **U**
UCSKEY	12-19	8	Char	Profile name
BG01	20-29	10	Char	Communication ID
BG02	30-39	10	Char	Communication password
BG03	40-54	15	Char	Application sender ID
BG04	55-69	15	Char	Application receiver ID
BG05	70-75	6	Char	Interchange date
BG06	76-81	6	Char	Interchange time
BG07	82-86	5	Char	Transmission control number
EG01	87-91	5	Char	Transmission control number
EG02	92-96	5	Char	Number of included groups
EG03	97-102	6	Char	Number of included transaction sets
EG04	103-112	10	Char	Number of included segments
GS01	113-114	2	Char	Functional group ID
GS02	115-129	15	Char	Application sender's code
GS03	130-144	15	Char	Application receiver's code
GS04	145-150	6	Char	Functional group date
GS05	151-158	8	Char	Functional group time
GS06	159-167	9	Char	Functional group control number
GS07	168-169	2	Char	Responsible agency code
GS08	170-181	12	Char	Version/release/industry ID
GE01	182-187	6	Char	Number of included sets
GE02	188-196	9	Char	Functional group control number
ST01	197-199	3	Char	Transaction set ID
ST02	200-208	9	Char	Transaction set control number
SE01	209-218	10	Char	Number of included segments
SE02	219-227	9	Char	Transaction set control number
DESCRIPT	228-257	30	Char	Description

X12 (X envelope) profile (X-P2)

This record defines the X12 envelope settings being used. The fields are described in Table 74 on page 274.

Additional profile layouts

Table 74. X12 (X envelope) profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **X**
X12KEY	12-19	8	Char	Profile name
ISA01	20-21	2	Char	Authorization information qualifier
ISA02	22-31	10	Char	Authorization information
ISA03	32-33	2	Char	Security information qualifier
ISA04	34-43	10	Char	Security information
ISA05	44-45	2	Char	Interchange sender ID qualifier
ISA06	46-60	15	Char	Interchange sender ID
ISA07	61-62	2	Char	Interchange receiver ID qualifier
ISA08	63-77	15	Char	Interchange receiver ID
ISA09	78-83	6	Char	Interchange date
ISA10	84-87	4	Char	Interchange time
ISA11	88-98	1	Char	Interchange standard ID
ISA12	89-93	5	Char	Interchange version ID
ISA13	94-102	9	Char	Interchange control number
ISA14	103-103	1	Char	Acknowledgment requested
ISA15	104-104	1	Char	Test indicator
ISA16	105-106	2	Char	Subelement separator
IEA01	107-111	5	Char	Number of included groups
IEA02	112-120	9	Char	Interchange control number
GS01	121-122	2	Char	Functional group ID
GS02	123-137	15	Char	Application sender's code
GS03	138-152	15	Char	Application receiver's code
GS04	153-158	6	Char	Group date
GS05	159-166	8	Char	Group time
GS06	167-175	9	Char	Functional group control number
GS07	176-177	2	Char	Responsible agency code
GS08	178-189	12	Char	Version/release/industry ID
GE01	190-195	6	Char	Number of included transaction sets
GE02	196-204	9	Char	Functional group control number
ST01	205-207	3	Char	Transaction set ID
ST02	208-216	9	Char	Transaction set control number
SE01	217-226	10	Char	Number of included segments
SE02	227-235	9	Char	Transaction set control number
DESCRIPT	236-265	30	Char	Description

Continuous receive profile (CONTRECV-P2 for CICS only)

This record defines the continuous receive settings being used. The fields are described in Table 75.

Table 75. Continuous receive profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **CONTRECV**
CONTRCV	12-27	16	Char	Profile name
ACTIVE	28-28	1	Char	Profile active
REQID	29-44	16	Char	Mailbox (requestor) ID
TPNICKN	45-60	16	Char	Trading Partner nickname
MSGCLS	61-68	8	Char	Network message user class
TRANSLATE	69-69	1	Char	Deenvelope and translate
RAWDATA	70-70	1	Char	Generate raw data
PRINTNM	71-78	8	Char	Print file name
PRINTTYP	79-80	2	Char	Print file type
EXCPNM	81-88	8	Char	Exception file name
EXCPTP	89-90	2	Char	Exception file type
ADDLRECS	91-95	5	Char	Additional records
DENVONLY	96-96	1	Char	Deenvelope only
DELAYFA	97-97	1	Char	Delay functional acknowledgment enveloping
FATSQ	98-105	8	Char	Enveloped functional acknowledgment TS queue name
RESPNM	106-113	8	Char	Name of response application TD queue
RESPTP	114-115	2	Char	Type of response
USERFLD	116-131	16	Char	User field passed to response
APPLID	132-139	8	Char	Application ID
NLSID	140-145	6	Char	National language ID
SYNCPTS	146-146	1	Char	Allow syncpoints
DUPENVL	147-147	1	Char	Allow duplicate envelopes
NETAKONLY	148-148	1	Char	Process network acknowledgments
PURGEINT	149-152	4	Dec	Transaction purge interval
DESCRIPT	153-182	30	Char	Description
SAPUPDT	183-183	1	Char	SAP update
PAGE	184-184	1	Char	Pageable translation active flag

CICS performance profile (SYSPROF-P2 for CICS only)

This record defines the CICS performance settings being used. The fields are described in Table 76.

Table 76. CICS performance profile

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = **SYSPROF**
SYSID	12-19	8	Char	System ID
PERSACTV	20-20	1	Char	Persistent environment active
PERSIZE	21-24	4	Dec	Persistent environment size
PERSTHDS	25-26	2	Dec	Number persistent environment threads
DESCRIPT	27-56	30	Char	Description

MQSeries queue profile (MQSERIES-P2)

This record defines the MQSeries queue settings being used. The fields are described in Table 77.

Table 77. MQSeries queue profile

Tag	Position	Length	Type	Description
PROFID	4-011	8	Char	Profile ID = **MQSERIES**
QUEUEID	012-019	8	Char	Logical name associated with the queue name
REALNAME	20-67	48	Char	Actual MQSeries queue name
MGRNAME	68-115	48	Char	Override queue manager name
READFLAG	116-116	1	Char	Destructive read flag indicator
MQSYNCFLAG	117-117	1	Char	Syncpoint control
MAXMSGLN	118-125	8	Char	Maximum message length
DESCRIPT	126-155	30	Char	Description

WebSphere Data Interchange Utility records format

WebSphere Data Interchange needs a description of the data format for each business application that generates data that will be translated, or uses data that has been translated. An application's data must be described to WebSphere Data Interchange so that it can be used as a source or target for translation.

WebSphere Data Interchange can accept two types of data record formats: raw data records and control and data records, as described below.

With Raw data, each record in raw data format identifies itself by containing a unique record identifier (a record ID). Raw data can be either comma-separated or fixed-position. For fixed-position data, the identifier starts in the same position and extends for the same length in each record. In comma-separated data, each value is separated by a comma. Comma-separated data is used only with data transformation maps. Raw data that is fixed-position can be used in data transformation maps, send maps and receive maps. The Record ID is actually a field in the record. As long as records contain identifiable record IDs, you can use application data without modification.

Control and Data (C and D) records are used when no record identifier exists clearly specifying the type of information contained in the record.

Note: For Data Transformation processing, the data format definition used during WDI Client mapping must also be defined as C and D or Raw Data application record type on the Record ID Info tab.

This section describes the format of the WebSphere Data Interchange Utility records, which are:
* "Control (C) records"
* "Data (D) records" on page 288
* "End transaction and interchange (Z) records" on page 290
* "Raw data records" on page 291
* "Optional records" on page 292

Control (C) records

The C record format is used for both send and receive transactions. All fields are left-justified and case-sensitive. The expanded fields (position 67 and greater) are optional. Use these fields to provide the override values you want the translator or enveloper to use.

The advantages of using C and D records are:
* You can use multiple formats within a single file.
* You can provide overrides for fields within service segments (ISA, GS, UNB, UNH, and so on.

The C record must identify the trading partner and the Data Format. With send/receiving map processing the internal trading partner ID field is used for Send Processing. Data Transformation processing currently requires the application and EDI trading partner fields.

Table 78 describes the record layout for translating C record data to EDI standard format.

Table 78. Control (C) record for translating to standard format

Label	Position	Length	Type	Description
RECID	1-1	1	Char	Record ID = **C**

Utility records format

Table 78. Control (C) record for translating to standard format (continued)

Label	Position	Length	Type	Description
INTPID	2-36	35	Char	Internal trading partner ID
FORMATID	37-52	16	Char	Data format ID
TRANRC	53-56	4	Integer	Translator return code
TRANXRC	57-60	4	Integer	Translator extended return code
UTILRC	61-64	4	Integer	WebSphere Data Interchange Utility return code
TESTIND	65-65	1	Char	Usage indicator
MUWIND	66-66	1	Char	Multiple D records indicator
XPANDED	67-67	1	Char	For send only. Expanded control block indicator
ITYPE	68-68	1	Char	For send only. Return information record indicator
ETYPE	69-69	1	Char	For send only. Return envelope header record indicator
GTYPE	70-70	1	Char	For send only. Return group header record indicator
TTYPE	71-71	1	Char	For send only. Return transaction set header record indicator
QTYPE	72-72	1	Char	For send only. Return queuing totals record indicator
ISID	73-107	35	Char	Interchange sender ID
IRID	108-142	35	Char	Interchange receiver ID
IVERREL	143-147	5	Char	Interchange version and release number
ISPW	148-161	14	Char	Interchange password
IAPREF	162-175	14	Char	Interchange application reference
GSID	176-210	35	Char	Group application sender ID
GRID	211-245	35	Char	Group application receiver ID
GVER	246-257	12	Char	Group version
GREL	258-269	12	Char	Group release number
GAPW	270-283	14	Char	Group password
TVER	284-289	6	Char	Transaction version
TREL	290-295	6	Char	Transaction release number
HOLDFLAG	296-296	1	Char	Held status indicator
BNDLFLAG	297-297	1	Char	Bundle indicator
ROUTCODE	298-300	3	Char	Generic routing code
ISYNTAXID	301-304	4	Char	Interchange syntax ID for **E** and **T** envelopes

Table 78. Control (C) record for translating to standard format (continued)

Label	Position	Length	Type	Description
ISYNTAXVER	305-305	1	Char	Interchange syntax version for **E** and **T** envelopes
ISIDQUAL	306-309	4	Char	Interchange sender ID qualifier
ISENDNAME	310-323	14	Char	Interchange sender name for **T** envelope) Application sender code for **U** envelope
IREVROUT	324-337	14	Char	Interchange reverse routing for **E** envelope
IRIDQUAL	338-341	4	Char	Interchange receiver ID qualifier
IRECVNAME	342-355	14	Char	Interchange receiver name for **T** envelope) Application receiver code for **U** envelope
IROUTEADDR	356-369	14	Char	Interchange routing address for **E** envelope
ISTDID	370-373	4	Char	Interchange standard ID for **I** and **X** envelopes
IPRIOR	374-374	1	Char	Interchange priority for **E** and **T** envelopes
ICOMMAGREE	375-409	35	Char	Interchange communication agreement for **E** envelope
GSIDQ	410-413	4	Char	Group application sender ID qualifier
GRIDQ	414-417	4	Char	Group application receiver ID qualifier
GRESPAG	418-419	2	Char	Group responsible agency code
DUPTRANS	420-420	1	Char	For receive only. Duplicate transaction indicator
FORCEC	421-421	1	Char	For send only. Force C record format on send translation
APPLTPID	422-437	16	Char	Application trading partner
EDITPID	438-453	16	Char	EDI trading partner
MAPID	454-469	16	Char	Name of map to be used for translation
TRXSTATUS	470-471	1	Char	C Flag F (failed) or Y (successful)
RSRVD1	454-512	42	Char	Reserved
CUSERDATA	513-768	256	Char	User data area
RSRVD2	769-1024	256	Char	Reserved

Control record label descriptions

RECID: The transaction set control record with a value of **C**.

INTPID: The internal trading partner ID used to define the map.

FORMATID: The data format ID used to describe the application data.

TRANRC: The translation status of the transaction. The code is in binary format.

TRANXRC: Defines the translation status of the transaction. The code is in binary format.

UTILRC: For send translation, indicates whether a transaction is written to the exception file in the UTILRC field. The WebSphere Data Interchange Utility returns the binary value **8** in UTILRC if it writes the transaction to the exception file.

For receive translation, UTILRC is always a binary **0**.

TESTIND: Indicates the usage/rule used for the transaction. Valid values are:

I	Information transaction. Use an information usage/rule. If an information usage/rule is not found, use a production usage/rule. Even when a production usage/rule is used, the transaction is flagged as an information transaction.
P	Production transaction. Use only a production usage/rule (default).
T	Test transaction. Use a test usage/rule. If a test usage/rule is not found, use a production usage/rule. Even when a production usage/rule is used, the transaction is flagged as a test transaction.
U	The translator should determine if the transaction is test, information, or production based on the usage/rule found. If a test usage/rule is found, the transaction is a test transaction, and value of **T** is returned. If an information usage/rule is found, the transaction is an information transaction, and a value of **I** is returned. If only a production usage/rule is found, the transaction is a production transaction, and a value of **P** is returned.

MUWIND: Indicates whether the transaction contains a single D record or multiple D records are involved in the transaction. Valid values are:

Y	Multiple D records are involved in the transaction.
N	A single D record is involved in the translation.

XPANDED: For send translation only. Indicates the version of the C record format being used (which fields are contained in the record). Receive translation always writes all fields through column 1024. Valid values are:

This value:	Specifies the C record contains fields through:
Y	Column 297

N	Column 67
1	Column 437
2	Column 1024

ITYPE: Indicates whether an information record is written to the exception/tracking file. Generally used for send translation. On receive translation, the field will contain the same values as shown in the discussion of "DUPTRANS" on page 287. Valid values are:

Y	Writes an information record to the exception/tracking file for each translated transaction. For more information, see "Information (I) records" on page 293.
N	Does not write an information record.

ETYPE: For send translation only. Indicates whether an interchange header image is written to the exception/tracking file. Valid values are:

Y	Writes an image of the interchange header segment to the exception/tracking file for each interchange processed. For more information, see "Interchange header (E) records" on page 295.
N	Does not write an image of the interchange header segment.

GTYPE: For send translation only. Indicates whether a group header image is written to the exception/tracking file. Valid values are:

Y	Writes an image of the group header segment to the exception/tracking file for each group processed. For more information, see "Group header (G) records" on page 295.
N	Does not write an image of the group header segment.

TTYPE: For send translation only. Indicates whether a transaction header image is written to the exception/tracking file. Valid values are:

Y	Writes an image of the transaction header segment to the exception/tracking file for each transaction processed. For more information, see "Transaction set header (T) records" on page 295.
N	Does not write an image of the transaction header segment.

QTYPE: For send translation only. Indicates whether a record containing totals relative to an interchange is written to the exception/tracking file. Valid values are:

Y	Writes information relative to totals in an interchange to the exception/tracking file for each interchange processed. For more information see "Queuing totals (Q) records" on page 295.
N	Does not write interchange totals information.

Utility records format

ISID: Overrides the ID provided in the envelope profile member or the trading partner profile member. It maps to a type **IS** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB03	E
ICS05	I
STX03 or STX04	T
BG03	U
ISA06	X

IRID: Overrides the ID provided in the trading partner profile or envelope profile member. It maps to a type **IR** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB06	E
ICS07	I
STX05 or STX06	T
BG04	U
ISA08	X

IVERREL: Overrides the version and release provided in the envelope profile member. It maps to types **VR** and **LV** standard data elements. WebSphere Data Interchange provides the equivalent data on receive translation.

ISPW: Overrides the password that the trading partner profile member provides. It maps to a type **PW** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB12	E
STX10	T
ISA04	X

IAPREF: Overrides the application reference ID provided in the envelope profile member. WebSphere Data Interchange uses this reference as the message user class for EDIFACT and UN/TDI. This reference points to a type **AP** standard data element. As distributed by WebSphere Data Interchange, no fields contain the **AP** data type. However, the following fields are frequently customized to have a data type of **AP**. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB14	E
STX11	T

GSID: Overrides the data format ID provided in the map. If the map provides the name of an envelope profile member, this entry overrides the sender ID that the envelope profile member provides. The group application sender ID maps to a type **AS** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNG02	E
GS02	X
GS02	I
GS02	U

GRID: Overrides the trading partner application name provided in the map. If the map provides the name of an envelope profile member, this entry overrides the receiver ID that the envelope profile member provides. The group application ID maps to a type **AR** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNG04	E
GS03	X
GS03	I
GS03	U

GVER: Overrides the group version provided in the envelope profile member. The value in the group version maps to a type **VR** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation.

GREL: Overrides the group release number provided in the envelope profile member. The group release number maps to type **LV** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation.

GAPW: Overrides the group password provided in the trading partner usages/rules. This usage/rule overrides the password in the envelope profile member. The group password maps to a type **PW** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation. Valid value is:

This value:	Overrides this envelope type:
UNG13	E

TVER: Overrides the transaction version provided in the envelope profile member. The transaction version maps to a type **VR** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation.

TREL: Overrides the transaction release number provided in the envelope profile member. The transaction release number maps to a type **LV** standard data element. WebSphere Data Interchange provides the equivalent data on receive translation.

HOLDFLAG: Specifies if the transaction is in hold status. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

Y	Places the transaction in held status.
N	Does not hold the transaction. Transactions that are not on hold and do not have a date specified for enveloping are available immediately for enveloping and sending.

BNDLFLAG: Specifies if the transaction starts a bundle. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

Y	Starts a group of related transactions (a bundle). All transactions that follow are part of the bundle until the translator encounters another transaction with this field set to **Y** or **N**.
N	Ends the bundle without starting a new one.
(blank)	Continues processing as usual. The translator ends a bundle automatically if the data forces the start of a new group or interchange envelope. Any action that changes the transaction or store status of one member of the bundle (such as envelope, send, or hold) is applied to all members of the bundle.

Note: If the current trading partner is not using functional groups (value of **N** in the `Functional group` field of the trading partner profile), WebSphere Data Interchange ignores the changes in the data. A new group is created but does not end the bundle.

ROUTCODE: For send translation only. A three-character generic routing code provided by the application and used by WebSphere Data Interchange to select a generic send usage/rule. A blank specifies a default generic send usage/rule. WebSphere Data Interchange provides the equivalent data on receive translation.

ISYNTAXID: Overrides the interchange syntax ID provided in the envelope profile member. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB01	E
STX01	T

ISYNTAXVER: Overrides the interchange syntax version provided in the envelope profile member. Valid values are:

This value:	Overrides this envelope type:
UNB02	E
STX02	T

ISIDQ: Overrides the interchange sender ID qualifier provided in the envelope profile member or trading partner profile member. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB04	E
ICS04	I
ISA05	X

ISENDNAME: Overrides the interchange sender name provided in the envelope profile member. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
STX04	T
UCS03	U (If not IS data type)

IREVROUT: Overrides the interchange reverse routing provided in the envelope profile member. WebSphere Data Interchange provides the equivalent data on receive translation. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
UNB05	E

IRIDQ: Overrides the interchange receiver ID qualifier provided in the trading partner profile or envelope profile member. Valid values are:

This value:	Overrides this envelope type:
UNB06	E
ICS06	I
ISA07	X

IRECVNAME: Overrides the interchange receiver name provided in the envelope profile member. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
STX06	T
UCS04	U (If not **IR** data type)

IROUTADDR: Overrides the interchange routing address provided in the envelope profile member. This entry must be left-justified. Valid value is:

This value:	Overrides this envelope type:
UNB08	E

ISTDID: Overrides the interchange standard ID provided in the envelope profile member. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
ICS02	I
ISA11	X

IPRIOR: Overrides the interchange processing priority defined in the envelope profile member. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB15	E
STX12	T

ICOMMAGREE: Overrides the interchange communication agreement provided in the envelope profile member. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB17	E

GSIDQ: Overrides the group sender ID qualifier provided in the envelope profile member. WebSphere Data Interchange provides the equivalent data on receive translation. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
UNG03	E

GRIDQ: Overrides the group receiver ID qualifier provided in the envelope profile member. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
UNG05	E

GRESPAG: Overrides the group responsible agency code and controlling agency provided in the envelope profile member. WebSphere Data Interchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNG09	E
GS07	I
GS07	U
GS07	X

DUPTRANS: For receive translation only. Specifies if a transaction is part of a duplicate envelope. Valid values are:

Y	Part of a duplicate envelope
N	Not part of a duplicate envelope

FORCEC: For send translation only. Forces a C record to be written. Valid values are:

Y	Always writes a C record
N or (blank)	Only writes a C record when an error occurs (default)

APPLTPID: The trading partner nickname of the application trading partner as defined in the trading partner profile.

EDITPID: The trading partner nickname of the EDI trading partner as defined in the trading partner profile.

RSVD1: Reserved for WebSphere Data Interchange.

CUSERDATA: The value in this field is copied to the TRCB where it can be modified by user exits. Before any C record is output by WebSphere Data Interchange, the value in this field is copied from the equivalent field in the TCRB. On receive translation, the value in the TRCB can be set using the WebSphere Data Interchange reserved variable DICUSERDATA. WebSphere Data Interchange does not use this field.

RSVD2: Reserved for WebSphere Data Interchange.

Data Transformation Node Names/AMM Structure for C record

The WDI Data Format Parser will create the FFCREC from the C record information. A customer written parser may also create an FFCREC Abstract Message Model node with any combination of the name/value nodes to enable the Data Format Deenveloper/Enveloper to process/create a customer defined C record layout.

```
...body (N)
......FFCREC (N)
.........INTPID (NV) = (char) 'MMTEPO1'
```

Utility records format

```
.........FORMATID (NV) = (char) 'MMTEPO1CDXX'
.........TESTIND (NV) = (char) 'N'
.........MUWIND (NV) = (char) 'Y'
.........XPANDED (NV) = (char) '1'
.........ITYPE (NV) = (char) 'Y'
.........ETYPE (NV) = (char) ' '
.........GTYPE (NV) = (char) ' '
.........TTYPE (NV) = (char) ' '
.........QTYPE (NV) = (char) ' '
.........ISID (NV) = (char) 'MMTEPO1'
.........IRID (NV) = (char) 'MMTEPO1'
.........IVERREL (NV) = (char) ' '
.........ISPW (NV) = (char) ' '
.........IAPREF (NV) = (char) ' '
.........GSID (NV) = (char) ' '
.........GRID (NV) = (char) ' '
.........GVER (NV) = (char) ' '
.........GREL (NV) = (char) ' '
.........GAPW (NV) = (char) ' '
.........TVER (NV) = (char) ' '
.........TREL (NV) = (char) ' '
.........HOLDFLAG (NV) = (char) ' '
.........BNDLFLAG (NV) = (char) ' '
.........ROUTCODE (NV) = (char) ' '
.........ISYNTAXID (NV) = (char) 'UNA'
.........ISYNTAXVER (NV) = (char) ' '
.........ISIDQUAL (NV) = (char) 'ZZ'
.........ISENDNAME (NV) = (char) ' '
.........IREVROUT (NV) = (char) ' '
.........IRIDQUAL (NV) = (char) 'ZZ'
.........IRECVNAME (NV) = (char) ' '
.........IROUTEADDR (NV) = (char) ' '
.........ISTDID (NV) = (char) ' '
.........IPRIOR (NV) = (char) ' '
.........ICOMMAGREE (NV) = (char) ' '
.........GSIDQ (NV) = (char) ' '
.........GRIDQ (NV) = (char) ' '
.........GRESPAG (NV) = (char) ' '
.........DUPTRANS (NV) = (char) ' '
.........FORCEC (NV) = (char) ' '
.........APPLTPID (NV) = (char) 'ANY'
.........EDITPID (NV) = (char) 'ANY'
.........MAPID (NV) = (char) ' '
.........CUSERDATA (NV) = (char) '(Value too large for buffer - 256 bytes)'
```

Data (D) records

There are two formats for data (D) records. One format is used when all the data for a transaction is provided by a single structure. The other format is used when data for a transaction is provided by multiple structures. The format of D records is described in the following tables.

For Fixed-to-Fixed mapping, when there is no target data format, the STRUCTNAME is the segment ID value from the EDI standard transaction definition, and D records are always output in the format described for single structures on 289.

When translating D records, use the name in the `ID field`, and populate ISA and GS segments with the values passed in the C record.

Notes:

1. The largest record your application or the translator can handle is 32000 bytes, which is the largest LRECL allowed for QSAM files. During translate-to-application processing, WebSphere Data Interchange divides any records that are larger than 32000 bytes into one or more X records, and uses a D record as the last record of the structure. For example, a structure that is 80000 bytes in length requires two X records (64000 bytes) followed by one D record (16000 bytes). During translate-to-EDI-standard processing, the program that creates the application file must create X and D records for structures exceeding 32000 bytes in length. Define the application file as variable-blocked (VB) if the application file will contain X records.

2. X records are not required for WDI Data Transformation. A new keyword, XRECS, has been added to the PERFORM command to allow the user to indicate no X records will be generated for inbound processing with C and D data format and application data is the target message.

Data record format - single structure

Table 79. Data record format: single structure

Label	Position	Length	Type	Description
RECID	1-1	1	Char	Record identifier
DATARCD	2-32750	32749	Char	Application transaction data

Data record label descriptions (single structure)

The following are descriptions of the labels for the data record when data structures are passed together.

RECID: Specifies if a transaction exceeds 32000 bytes. Valid values are:

D

> An entire transaction, or the last record of a transaction that exceeds 32000 bytes

X

> The first and middle records of a transaction that exceed 32000 bytes

DATARCD: Specifies application transaction data.

Data record format (multiple structures)

Label	Position	Length	Type	Description
RECID	1-1	1	Char	Record identifier
STRUCNAM	2-17	16	Char	Structure name
DATARCD	18-32750	32733	Char	Application transaction data

Data record label descriptions (multiple structures)

The following are descriptions of the labels for the data record when data structures are passed separately.

RECID: Specifies if a transaction exceeds 32000 bytes. Valid values are:

D

> An entire transaction, or the last record of a transaction that exceeds 32000 bytes

X

> The first and middle records of a transaction that exceed 32000 bytes

STRUCNAME: The structure name defined in the data format for this transaction.

DATARCD: Specifies application transaction data.

End transaction and interchange (Z) records

The Z record is optional and is used in the z/OS environment when the size of an interchange is being controlled by the application, or in the CICS environment when recoverable resources are being used. The Z record marks the end of:

- The application data for a transaction. If a Z record is not present, WebSphere Data Interchange can only detect the end of a transaction from reading the next transaction's C record.

  If your input data is in recoverable intrapartition TD queues, using the Z record keeps the reading of the C record for the next transaction from being part of the syncpoint interval for the current transaction. The Z record provides a formal end for a transaction rather than the implied transaction end achieved when the next C record is read.

- The application data for a transaction and indicates that the current transaction is the last transaction for an interchange when a Z record with the ENDINTERCH field set to 1 (Z1) is present. If a Z1 record is not present, WebSphere Data Interchange can only detect the end of an interchange from reading the next transaction's C record.

  If your input data is in recoverable intrapartition TD queues, using a Z1 record keeps the reading of the C record for the next transaction from being part of the syncpoint interval for the current interchange. The Z1 record provides a formal end for an interchange rather than the implied end achieved when the next C record is

read. The Z1 record can be used to artificially limit the size of an interchange.
A Z1 record causes the current interchange to be completed. A new interchange
starts with the next transaction.

Z and Z1 records are only necessary when using the C and D record format, and
when the conditions mentioned below are true. Otherwise, using Z and Z1 records is
optional. The conditions for use are:

- The WebSphere Data Interchange Utility is being executed in the CICS environment.
- Application data is given to the WebSphere Data Interchange Utility in recoverable
  intrapartition TD queues.
- WebSphere Data Interchange is allowed to issue CICS SYNCPOINT commands.

Z record format

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = **Z**
ENDINTERCH	2	1	Char	End of interchange indicator

Z record label descriptions

RECID: A value of **Z** identifies this record as a transaction or interchange terminator.

ENDINTERCH: A value of **1** indicates the preceding transaction is the last in the
current interchange. The next C record begins a new transaction and new interchange.

Raw data records

Table 80 describes the format of raw data records. The data format must indicate the
position, length, and type of record ID field within the data. The data format can also
provide the field in the application data that contains the internal trading partner ID
value. For TRANSLATE TO STANDARD processing, the data format ID is provided by
the RAWFMTID keyword on the PERFORM command. During TRANSLATE TO
APPLICATION processing, the translator automatically supplies the record ID values.
The internal trading partner ID value is taken from the trading partner receive
usage/rule when raw data is requested.

WebSphere Data Interchange can create raw data output during TRANSLATE TO
STANDARD processing with Fixed-to-Fixed mapping. Record ID values and the internal
trading partner ID default values are used but might be overridden during mapping.

Raw data record format

Table 80. Raw data record format

Label	Position	Length	Type	Description
DATARCD	1-32750	32750	Char	Application transaction data

Optional records

The following optional record types are available:
- "Information (I) records" on page 293
- "Interchange header (E) records" on page 295
- "Group header (G) records" on page 295
- "Transaction set header (T) records" on page 295
- "Queuing totals (Q) records" on page 295
- "File (F) records" on page 296

You can request optional records using the TYPE fields in the control record ("Control (C) records" on page 277), on the Additional Records panel, or by using the OPTRECS keyword ("OPTRECS" on page 165) in one of the following commands:

```
DEENVELOPE
DEENVELOPE AND TRANSLATE
ENVELOPE
ENVELOPE AND SEND
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE
REENVELOPE
REENVELOPE AND SEND
RETRANSLATE TO APPLICATION
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO APPLICATION
TRANSLATE TO STANDARD
```

Different optional records are available for different commands. The commands and the record types they support are defined in Table 81.

Table 81. Optional records supported by command

Command	Record type					
	E	F	G	I	Q	T
DEENVELOPE DEENVELOPE AND SEND	X	X	X	X	X	X
ENVELOPE ENVELOPE AND SEND	X		X	X	X	X
RECEIVE AND DEENVELOPE RECEIVE AND TRANSLATE	X		X	X	X	X
REENVELOPE REENVELOPE AND SEND	X		X	X	X	X
RETRANSLATE TO APPLICATION						
TRANSLATE AND ENVELOPE	X		X	X	X	X
TRANSLATE AND SEND	X		X	X	X	X
TRANSLATE TO APPLICATION				X		X
TRANSLATE TO STANDARD				X		

For TRANSLATE TO STANDARD operations, the WebSphere Data Interchange Utility writes the optional records to a file as described below:

- Writes the records to FFSTRAK if it exists.
- Writes the records to FFSEXCP if FFSTRAK does not exist.
- Writes no records if FFSTRAK does not exist and raw data (RAWFMTID keyword) is requested.

For TRANSLATE TO APPLICATION operations, the WebSphere Data Interchange Utility writes the optional records to a file as described below:

- Writes the records to the application output file if C and D records are requested.
- Writes the records to FFSEXCP if the raw data records are requested (RAWDATA keyword).

For DEENVELOPE operations, the WebSphere Data Interchange Utility writes the optional records to the FFSEXCP file.

Information (I) records

Table 82 describes the format and contents of the information (I) record. You can request an information record on the Optional Record Options panel, in the `ITYPE` field in the control record, or by using the `OPTRECS` keyword in a command. WebSphere Data Interchange returns one record for each transaction with the values that were present at the time the transaction was translated.

Table 82. Information record format

Label	Position	Length	Type	Description
RECID	1-1	1	Char	Record ID = **I**
IHXCTL	2-15	14	Char	Interchange header control number
ISID	16-50	35	Char	Interchange sender ID
IRID	51-85	35	Char	Interchange receiver ID
IDATE	86-91	6	Char	Interchange date
ITIME	92-97	6	Char	Interchange time
IVERREL	98-102	5	Char	Interchange version/release
IGT	103-108	6	Char	For sending, the total number of groups in the interchange at the current time For receiving, the total number of groups processed so far in the interchange
ITT	109-114	6	Char	For sending, the total number of transactions in the interchange at the current time For receiving, the total number of transactions processed so far in the interchange

Utility records format

Table 82. Information record format (continued)

Label	Position	Length	Type	Description
IST	115-124	10	Char	For sending, the total number of segments in the interchange at the current time For receiving, the total number of segments processed so far in the interchange
IBT	125-132	8	Char	For sending, the total number of bytes in the interchange at the current time For receiving, the total number of bytes processed so far in the interchange
ISPW	133-146	14	Char	Interchange password
IAPREF	147-160	14	Char	Interchange application reference
GHXCTL	161-174	14	Char	Control number of group header
GFGID	175-180	6	Char	Group functional group ID
GSID	181-215	35	Char	Group application sender ID
GRID	216-250	35	Char	Group application receiver ID
GDATE	251-256	6	Char	Group date
GTIME	257-262	6	Char	Group time
GVER	263-274	12	Char	Group version
GREL	275-286	12	Char	Group release
GTT	287-292	6	Char	For sending, current group transaction total For receiving, total transactions in group
GAPW	293-306	14	Char	Group password
THXCTL	307-320	14	Char	Control number of transaction set header
TTC	321-326	6	Char	Transaction code
TVER	327-332	6	Char	Transaction version
TREL	333-338	6	Char	Transaction release
TST	339-348	10	Char	Transaction segment total
AC	349-383	35	Char	Application control field value
THANDLE	384-403	20	Char	Expanded value of the handle assigned to this transaction in the Transaction Store
TPNICKN	404-419	16	Char	Trading partner nickname
GDATE2	420-427	8	Char	Group date with century
RESERVED	428-483	56	Char	Reserved for future use

Interchange header (E) records

Table 83 shows the format and contents of the interchange (E) header record. You can request an interchange header record using the ETYPE field in the control record or the OPTRECS keyword in a command. WebSphere Data Interchange returns one record for each interchange that is created or received.

Table 83. Interchange header (E) record format

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = **E**
EDATA	2-256	255	Char	Standard interchange header image

Group header (G) records

Table 84 shows the format and contents of the group header (G) record. You can request a group header record using the GTYPE field in the control record or the OPTRECS keyword in a command. WebSphere Data Interchange returns one record for each group created or received.

Table 84. Group header (G) record format

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = **G**
GDATA	2-256	255	Char	Functional group header image

Transaction set header (T) records

Table 85 shows the format and contents of the transaction set header (T) record. You can request a transaction header record using the TTYPE field in the control record or the OPTRECS keyword in a command. WebSphere Data Interchange returns one record for each transaction created or received.

Table 85. Transaction set header (T) record format

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = **T**
TDATA	2-256	255	Char	Transaction set header image

Queuing totals (Q) records

Table 86 shows the format and contents of the queuing totals record. You can request a queuing totals record using the QTYPE field in the control record or the OPTRECS keyword in a command. WebSphere Data Interchange returns one record to the exception file each time an interchange is queued or deenveloped.

Table 86. Queueing totals (Q) record format

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = **Q**.

Table 86. Queueing totals (Q) record format (continued)

Label	Position	Length	Type	Description
QBT	2-9	8	Char	Number of bytes queued for the envelope.
QST	10-19	10	Char	Number of segments in the envelope.
QTT	20-25	6	Char	Number of transactions in the envelope.
QGT	26-31	6	Char	Number of groups in the envelope.
QDSNAME	32-87	56	Char	Physical data set name from which transactions were read or to which transactions were written. The name is terminated with a NULL character (X'00').

File (F) records

Table 87 shows the format and contents of the file (F) record. This allows the real file of the applications file to be found.

Table 87. File (F) record layout

Label	Length	Type	Description
RECID	1	Char	Record ID = **F**.
FTYPE	1	Char	Type of data written. E=EDI D=ADF X=XML C=CSV
FLNAME	8	Char	Logical filename from **PERFORM** command
FPNAME	AIX/Windows - 272 z/OS - 56	Char	Physical file name
FRCVID	35	Char	Receiver ID from EDI header
FRCVQUAL	4	Char	Receiver qualifier from EDI header
FSNDID	35	Char	Sender ID from EDI header
FSNDQUAL	4	Char	Sender qualifier from EDI header
FNICKNAME	16	Char	Trading partner nickname from trading partner profile
FRECCNT	15	Char	Record count

Data Transformation C and D Support

The WDI Utility will detect C and D data format processing when application data is the source when no dictionay/document keyword is specified on the PERFORM TRANSFORM command. Note the data format definition used during WDI Client mapping must also be defined as C and D application record type on the Record ID Info tab. The C record FORMATID field identifies the document to be used and will be extracted and used during the translation process. The WDI Client mapping special variable DICUserData can be used to map information from the source document to the C record field CUSERDATA when application data is the target document. This field can contain any data up to 256 bytes long.

Data Transformation processing does not currently use the internal trading partner ID; therefore the application and EDI trading partner fields will be used and the customer will need to use the fully expanded C record format which is 1024 bytes. The Abstract Message Model (AMM) node names will follow the names described in "WebSphere Data Interchange Utility records format" on page 276. The name/value nodes will go under a name node FFCREC. The RECID field will NOT be present in the AMM since it is already known that this is a 'C' record. Any C record value that identifies an override value will set the appropriate attribute. For example, the C record override ISID will set the attribute IchgSndrId as though a SetProperty command were issued within the mapping.

The following C record fields are not currently supported with data transformation processing:

```
INTPID 2-36 35 (Char Internal trading partner ID)
HOLDFLAG 296-296 1 (Char Held status indicator)
BNDLFLAG 297-297  1 (Char Bundle indicator)
ROUTCODE 298-300 3 (Char Generic routing code)
```

Management reporting

The management reporting data extracts are formatted as sequential files containing fixed-length (1024 bytes) records. All data extracts are written to the EDIQUERY file. Since other commands use the EDIQUERY file, you should define the EDIQUERY file as variable block 32756 rather than fixed block 1024 to accommodate commands that output larger records, such as the image records created by the ENVELOPE DATA EXTRACT and TRANSACTION DATA EXTRACT commands.

The output is tabular with columns representing categories of information and rows containing the actual data entries. The formats of the data extracts are described in the following tables.

Trading partner profile data extract

This record defines the trading partner profile settings being used to extract data. The fields are described in Table 88 on page 298.

Management reporting

Table 88. Trading partner profile data extract fields

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = **TPI**
Trading Partner Nickname	4-19	16	Char	Trading partner ID in WebSphere Data Interchange
Company Name	20-59	40	Char	Company name of the trading partner
Address line 1	60-99	40	Char	First line of the company's address
Address line 2	100-139	40	Char	Second line of the company's address
Comment line 1	140-179	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Comment line 2	180-219	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Contact Name	220-249	30	Char	Name of a contact at this trading partner
Contact Phone	250-274	25	Char	Phone number of the contact
Network ID	275-282	8	Char	ID of the network used to communicate with this trading partner
Interchange ID	283-317	35	Char	Receiver ID used in interchanges to this trading partner
Account ID	318-349	32	Char	Network account ID
User ID	350-381	32	Char	Network user ID
Direction	382-382	1	Char	Direction of the (inbound or outbound)
Date of Last Transmission	383-390	8	Char	Interchange date of last transmission to this trading partner
Interchange control number	391-404	14	Char	Interchange control number of last transmission to this trading partner
Group control number	405-418	14	Char	Group control number of the last transmission to this trading partner
Transaction control number	419-432	14	Char	Transaction control number of the last transaction to this trading partner
Filler	433-1024	592	Char	Filler for expansion

Trading partner capability data extract

This record defines the trading partner capability settings being used to extract data. The fields are described in Table 89 on page 299.

Table 89. Trading partner capability data extract fields

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = **TPC**
Trading Partner Nickname	4-19	16	Char	Trading partner ID in WebSphere Data Interchange
Internal Trading Partner ID	20-54	35	Char	ID used for this trading partner internally (customer number, supplier number)
Company Name	55-94	40	Char	Company name of the trading partner
Address line 1	95-134	40	Char	First line of the company's address
Address line 2	135-174	40	Char	Second line of the company's address
Comment line 1	175-214	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Comment line 2	215-254	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Direction	255-255	1	Char	Direction of the (inbound or outbound)
Standard ID	256-263	8	Char	ID of the EDI standard (X12, EDIFACT)
Version ID	264-265	2	Char	Version of the EDI standard
Release ID	266-267	2	Char	Release of the EDI standard
Description	268-317	50	Char	Description of the EDI standard/version/release
Transaction ID	318-325	8	Char	Transaction ID (such as ORDERS, DISPATCH, 850, 860)
Map ID	326-341	16	Char	Trading partner map name
Measurement date	342-349	8	Char	Date testing or production started with this map/trading partner combination
Measurement ID	350-353	4	Char	Type of statistic
Total number of transactions	354-368	15	Char	Total number of test or production transactions exchanged with this map/trading partner combination
Total errors	369-383	15	Char	Total number of test or production transactions exchanged with this map/trading partner combination that had errors
Appl Trading Partner	384-399	16	Char	Application trading partner ID
Filler	400-1024	625	Char	Filler for expansion

Network activity data extract

This record defines the network activity settings being used to extract data. The fields are described in Table 90.

Table 90. Network activity data extract fields

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = **NTA**
Requestor ID	4-19	16	Char	Requestor ID
Network ID	20-27	8	Char	Network ID
Network name	28-57	30	Char	Network descriptive name
Account number	58-89	32	Char	Network account number
User ID	90-121	32	Char	Network user ID
Direction	122-122	1	Char	Direction of the transmission
Charge code	123-123	1	Char	Network charge code
Measurement ID	124-127	4	Char	Type of statistic
Day	128-135	8	Char	Measurement date
Interchange envelopes	136-150	15	Char	Total number of interchange envelopes
Total characters	151-165	15	Char	Total number of characters sent
Filler	166-1024	859	Char	Filler for expansion

Transaction activity data extract

This record defines the transaction activity settings being used to extract data. The fields are described in Table 91.

Table 91. Transaction activity data extract fields

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = **TPA**
Trading Partner Nickname	4-19	16	Char	Trading partner ID in WebSphere Data Interchange
Internal Trading Partner ID	20-54	35	Char	ID used for this trading partner internally (such as customer number, supplier number)
Company Name	55-94	40	Char	Company name of the trading partner
Address line 1	95-134	40	Char	First line of the company's address
Address line 2	135-174	40	Char	Second line of the company's address
Comment line 1	175-214	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)

Table 91. Transaction activity data extract fields (continued)

Label	Position	Length	Type	Description
Comment line 2	215-254	40	Char	Can be used to further classify the trading partner
Direction	255-255	1	Char	Direction of the transmission
Standard ID	256-263	8	Char	ID of the EDI standard
Version ID	264-265	2	Char	Version of the EDI standard
Release ID	266-267	2	Char	Release of the EDI standard
Description	268-317	50	Char	Description of the EDI standard/version/release
Transaction ID	318-325	8	Char	Transaction ID
TPT ID	326-341	16	Char	Trading partner transaction ID
Data format ID	342-357	16	Char	Name of the data format
Measurement ID	358-361	4	Char	Type of statistic
Measurement Date	362-369	8	Char	Date of this statistic
Total transactions	370-384	15	Char	Total transactions for the indicated date
Total errors	385-399	15	Char	Total transactions in error for the indicated date
Appl Trading Partner	400-415	16	Char	Application trading partner ID
Filler	416-1024	609	Char	Filler for expansion

Transaction Store data extract information categories

You can extract information from the WebSphere Data Interchange Transaction Store databases using the ENVELOPE DATA EXTRACT and the TRANSACTION DATA EXTRACT commands. Refer to "Producing management reports from the Transaction Store" on page 10 for a description of these PERFORM commands. The categories of information that can be requested are described in Table 92.

Table 92. Transaction Store information categories

This value:	Requests:	Using this keyword:
E	Interchange data	INTERCHANGE(Y)
G	Group data	GROUP(Y)
T	Transaction data	TRANSACTION(Y)
A	Application data	APPLICATION(Y)
R	Transaction image	IMAGE(Y)
E, G, T	Send acknowledgment data	SENDACKDATA(Y)
E, G, T	Receive acknowledgment data	RECEIVEACKDATA(Y)
F, K	Send acknowledgment image	SENDACKIMAGE(Y)
F, K	Receive acknowledgment image	RECEIVEACKIMAGE(Y)

All data extracted from the Transaction Store is written to the EDIQUERY file. The information categories are either written as separate records by setting CONCATENATE to **N**, or combined and written as a single record by setting CONCATENATE to **Y**.

The following rules apply when requesting information categories:

1. Images (**R**, **F**, and **K**) are not produced unless you also request the transaction.
2. Images (**R**, **F**, and **K**) are always written as separate records even when concatenation has been requested.
3. Send acknowledgment data (**E**, **G**, and **T**) is not produced unless you also request the group record. If you want transaction acknowledgment data, you must also request the transaction record.
4. Send acknowledgment data (**E**, **G**, and **T**) is always concatenated to the corresponding group or transaction record even when concatenation has not been requested.
5. Receive acknowledgment data (**E**, **G**, and **T**) is not produced unless you also request the group or transaction record.
6. Receive acknowledgment data (**E**, **G**, and **T**) is always concatenated to the corresponding group or transaction record even when concatenation has not been requested.

Transaction Store data extract common key

All records created by Transaction Store Data Extract command have a common 141-byte key field that begins with a 3-character record ID. Portions of the key that do

not apply to a particular record are initialized with blanks. Table 93 defines the common key format used to extract data, and shows which fields are used for which record types.

Table 93. Transaction Store common key format

Label	Position	Length	Type	Description
Record ID	1	3	Char	Record ID = **TX** Used for record types: A, E, G, K, R, T
Nickname	4	16	Char	Trading partner nickname Used for record types: A, E, G, K, R, T
Direction	20	1	Char	Direction of the transaction Used for record types: A, E, G, K, R, T
Control Number	21	14	Char	Interchange control number Used for record types: A, E, G, K, R, T
Receiver ID	35	35	Char	Interchange receiver ID Used for record types: A, E, G, K, R, T
Control Number	70	14	Char	Group control number Used for record types: A, G, K, R, T
Control Number	84	14	Char	Transaction control number Used for record types: A, K, R, T
Controlling handle	98	20	Char	Handle value of controlling transaction (YYYYMMDDHHMMSSnnnnnn) Used for record types: A, K, R, T
Transaction handle	118	20	Char	Handle value of transaction (YYYYMMDDHHMMSSnnnnnn) Used for record types: A, K, R, T
Sequence Number	138	4	Char	Sequence number Used only for record type A

Transaction Store data extract record formats

The following tables show the formats for the records created by the TRANSACTION STORE DATA EXTRACT command. All tables, except images, are padded to a length of 1024 bytes to leave room for expansion. If concatenation is requested in the WebSphere Data Interchange Utility control statements, the full records as described are concatenated into a single record. Images are written separately and use the full logical record length of the EDIQUERY data set.

Interchange data extract record layout

This record defines the interchange settings being used to extract data. The fields are described in Table 94.

Table 94. Interchange data extract record layout

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = **E**. **E1** = Functional acknowledgment **E2** = Transaction acknowledgment
Nickname	4-19	16	Char	Trading partner nickname.
Direction	20-20	1	Char	Direction of the interchange.
Control Number	21-34	14	Char	Interchange control number.
Receiver ID	35-69	35	Char	Interchange receiver ID.
Filler	70-141	72	Char	Blanks.
Sender ID	142-176	35	Char	Interchange sender ID.
Fake flag	177-177	1	Char	No interchange header.
Sequence error flag	178-178	1	Char	Interchange out of sequence (ENVELOPE DATA EXTRACT only).
Usage indicator	179-179	1	Char	Type of usage.
Duplicate interchange flag	180-180	1	Char	Duplicate interchange received.
Envelope date	181-194	14	Char	Date/time envelope created (YYYYMMDDHHMMSS).
Send date	195-208	14	Char	Date/time envelope sent (YYYYMMDDHHMMSS).
TA1 acknowledgment	209-209	1	Char	TA1 acknowledgment received.
TA1 date	210-223	14	Char	Date/time TA1 received (YYYYMMDDHHMMSS).
Network status code	224-225	2	Char	Network status.
Network status text	226-245	20	Char	Network status code in text format.
Acknowledgment expected	246-246	1	Char	Network acknowledgment expected.
Acknowledgment received	247-247	1	Char	Network acknowledgment received.
Acknowledged date	248-261	14	Char	Date/time envelope of network acknowledgment (YYYYMMDDHHMMSS).
Message user class	262-269	8	Char	Message user class assigned when sent.

Table 94. Interchange data extract record layout (continued)

Label	Position	Length	Type	Description
Message name	270-277	8	Char	Message name assigned when sent.
Sequence number	278-282	5	Char	Sequence number assigned when sent.
Message ID	283-290	8	Char	Message ID assigned when sent.
Physical data set name	291-346	56	Char	Physical data set name to which data was queued.
Group count	347-357	11	Char	Number of groups in interchange.
Transaction count	358-368	11	Char	Number of transactions in interchange.
Segment count	369-379	11	Char	Number of segments in interchange.
Interchange size	380-390	11	Char	Number of bytes in interchange.
Interchange header	391-640	250	Char	Interchange header image.
Interchange trailer	641-670	30	Char	Interchange trailer image.
Filler	671-1024	354	Char	Reserved for expansion.

Group data extract record layout

This record defines the group settings being used to extract data. The fields are described in Table 95.

Table 95. Group data extract record layout

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = **G**. **G1** = Functional acknowledgment **G2** = Transaction acknowledgment
Nickname	4-19	16	Char	Trading partner nickname.
Direction	20-20	1	Char	Direction of the group.
Control Number	21-34	14	Char	Interchange control number.
Receiver ID	35-69	35	Char	Interchange receiver ID.
Control Number	70-83	14	Char	Group control number.
Filler	84-141	58	Char	Blanks.
Fake flag	142-142	1	Char	Interchange did not have groups.
Sender ID	143-177	35	Char	Application sender ID.
Receiver ID	178-212	35	Char	Application receiver ID.
Acknowledgment expected	213-213	1	Char	Functional acknowledgment expected.

Transaction Store data extraction information categories

Table 95. Group data extract record layout (continued)

Label	Position	Length	Type	Description
Acknowledgment received	214-214	1	Char	Functional acknowledgment received. **Note:** If an acknowledgment is not expected, this field is blank.
Acknowledgment date	215-228	14	Char	Date/time group acknowledgment received (YYYYMMDDHHMMSS).
Acknowledgment handle	229-248	20	Char	Handle value for the functional acknowledgment transaction (YYYYMMDDHHMMSSnnnnnn).
Transaction count	249-259	11	Char	Number of transactions in group.
Segment count	260-270	11	Char	Number of segments in group.
Group size	271-281	11	Char	Number of bytes in group.
Group header	282-434	153	Char	Image of the group header.
Group trailer	435-460	26	Char	Image of the group trailer.
Filler	461-1024	564	Char	Filler for expansion.

Transaction data extract record layout

This record defines the transaction settings being used to extract data. The fields are described in Table 96.

Table 96. Transaction data extract record layout

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = **T**. **T1** = Functional acknowledgment **T2** = Transaction acknowledgment
Nickname	4-19	16	Char	Trading partner nickname.
Direction	20-20	1	Char	Direction.
Control Number	21-34	14	Char	Interchange control number.
Receiver ID	35-69	35	Char	Interchange receiver ID.
Document ID	70-77	8	Char	Standard transaction ID
Control Number	78-91	14	Char	Group control number.
Control Number	92-105	14	Char	Transaction control number.
Controlling handle	106-125	20	Char	Handle value of the controlling transaction (YYYYMMDDHHMMSSnnnnnn).
Transaction handle	126-145	20	Char	Handle value of the transaction (YYYYMMDDHHMMSSnnnnnn).
Filler	146-149	4	Char	Blanks.

Table 96. Transaction data extract record layout (continued)

Label	Position	Length	Type	Description
Enveloped date	150-163	14	Char	Date/time transaction was enveloped (YYYYMMDDHHMMSS).
Creation date	164-177	14	Char	Date/time transaction put into the store.
Transaction status code	178-179	2	Char	Current transaction status.
Transaction status text	180-199	20	Char	Transaction status in a text format.
Acknowledgment received	200-200	1	Char	Group acknowledgment received.
Acknowledgment received text	201-220	20	Char	Group acknowledgment received in a text format.
Acknowledgment date	221-234	14	Char	Date/time group acknowledgment was received (YYYYMMDDHHMMSS).
Trx Acknowledgment expected	235-235	1	Char	Transaction acknowledgment expected.
Trx Acknowledgment received	236-236	1	Char	Transaction acknowledgment received.
Trx Acknowledgment received text	237-256	20	Char	Transaction acknowledgment received in a text format.
Trx Acknowledgment date	257-270	14	Char	Date/time transaction acknowledgment was received (YYYYMMDDHHMMSS).
Acknowledgment handle	271-290	20	Char	Handle value of the transaction acknowledgment transaction (YYYYMMDDHHMMSSnnnnnn).
Segment count	291-301	11	Char	Number of segments in transaction.
Transaction size	302-312	11	Char	Number of bytes in transaction.
Enveloped segment count	313-323	11	Char	Number of segments in transaction when enveloped.
Enveloped transaction size	324-334	11	Char	Number of bytes in transaction when enveloped.
Format ID	335-350	16	Char	Last data format ID.
Transaction ID	351-358	8	Char	Standard transaction.
Group ID	359-364	6	Char	Function Group ID associated with the transaction.
Envelope member	365-372	8	Char	Member name used for enveloping.
Envelope type	373-373	1	Char	Type of envelope associated with transaction.

Transaction Store data extraction information categories

Table 96. Transaction data extract record layout (continued)

Label	Position	Length	Type	Description
Network ID	374-381	8	Char	Network associated with trading partner.
Standard ID	382-389	8	Char	EDI standard ID of the transaction.
Standard Version	390-391	2	Char	EDI standard version.
Standard Level	392-393	2	Char	EDI standard release level.
Internal Trading Partner ID	394-428	35	Char	Last internal trading partner ID.
Application control field	429-463	35	Char	Last application control field.
Delivery date	464-477	14	Char	Last date given/gotten from application (YYYYMMDDHHMMSS).
Earliest envelope date	478-485	8	Char	Earliest date that transaction will be enveloped (YYYYMMDD).
Earliest purge date	476-493	8	Char	Earliest date that transaction will be automatically purged (YYYYMMDD).
Error level	494-494	1	Char	Translation error level.
Store status	495-496	2	Char	Store status of the transaction.
Store status text	497-516	20	Char	Store status in a text format.
Override flag	517-517	1	Char	Envelope overrides provided for transaction.
Held flag	518-518	1	Char	Transaction held.
Test flag	519-519	1	Char	Test transaction.
Duplicate flag	520-520	1	Char	Transaction part of duplicate envelope.
Purge flag	521-521	1	Char	Purge transaction.
Translate flag	522-522	1	Char	Translated.
Detached flag	523-523	1	Char	Transaction detached.
Override handle	524-543	20	Char	Handle for enveloping overrides (YYYYMMDDHHMMSSnnnnnn).
Network security profile name	544-551	8	Char	Network security profile name for group level encryption.
Encryption key	552-567	16	Char	Encryption key name for group level encryption.
Authentication key	568-583	16	Char	Authentication key name for group level encryption.
Application assigned control number	584-597	14	Char	Transaction control number if assigned by the application.
Data element delimiter	598-598	1	Char	Data element delimiter used when transaction was translated.

Table 96. Transaction data extract record layout (continued)

Label	Position	Length	Type	Description
Sub element-delimiter	599-599	1	Char	Subelement delimiter used when transaction was translated.
Segment terminator	600-600	1	Char	Segment terminator used when transaction was translated.
Decimal notation	601-601	1	Char	Decimal notation used when transaction was translated.
Release character	602-602	1	Char	Release character used when transaction was translated.
Segment ID separator	603-603	1	Char	Segment ID separator used when transaction was translated.
Transaction header	604-688	85	Char	Image of the transaction header.
Transaction trailer	689-714	26	Char	Image of the transaction trailer.
Last ID	715-730	16	Char	Last used.
Filler	731-1032	302	Char	Reserved for expansion.

Application data extract record layout

This record defines the application settings being used to extract data. The fields are described in Table 97.

Table 97. Application data extract record layout

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = **A**
Nickname	4-19	16	Char	Trading partner nickname
Direction	20-20	1	Char	Direction of format
Control Number	21-34	14	Char	Interchange control number
Receiver ID	35-69	35	Char	Interchange receiver ID
Control Number	70-83	14	Char	Group control number
Control Number	84-97	14	Char	Transaction control number
Controlling handle	98-117	20	Char	Handle value of controlling transaction (YYYYMMDDHHMMSSnnnnnn)
Transaction handle	118-137	20	Char	Handle value of transaction (YYYYMMDDHHMMSSnnnnnn)
Sequence Number	138-141	4	Char	Sequence number
Format ID	142-157	16	Char	Data format ID
Application ID	158-165	8	Char	Application ID
BATCH ID	166-173	8	Char	Batch ID

Transaction Store data extraction information categories

Table 97. Application data extract record layout (continued)

Label	Position	Length	Type	Description
Delivery Date	174-187	14	Char	Date/time delivered to application (YYYYMMDDHHMMSS)
Error level	188-188	1	Char	Translation error level
Acceptable error level	189-189	1	Char	Acceptable error level
Error count	190-200	11	Char	Number of errors found
Filler	201-1024	824	Char	Reserved for expansion

Transaction/Acknowledgment image data extract record layout

This record defines the transaction and acknowledgment image settings being used to extract data. The fields are described in Table 98.

Table 98. Transaction/Acknowledgement image data extract record layout

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID. Valid values are: **RX** = Transaction image continued **RZ** = Final transaction image record **FX** = Functional acknowledgment image continued **FZ** = Final functional acknowledgment image record **KX** = Transaction acknowledgment image continued **KZ** = Final transaction acknowledgment image record
Nickname	4-19	16	Char	Trading partner nickname.
Direction	20-20	1	Char	Direction of image data.
Control Number	21-34	14	Char	Interchange control number.
Receiver ID	35-69	35	Char	Interchange receiver ID.
Control Number	70-83	14	Char	Group control number.
Control Number	84-97	14	Char	Transaction control number.
Controlling handle	98-117	20	Char	Handle value of controlling transaction (YYYYMMDDHHMMSSnnnnnn).
Transaction handle	118-137	20	Char	Handle value of transaction (YYYYMMDDHHMMSSnnnnnn).
Filler	138-141	4	Char	Blanks.
Total size	142-152	11	Char	Total size of the image.
Record size	153-163	11	Char	Size of image data in this record.
Image	164-end	Variable	Char	Transaction or acknowledgment image.

Chapter 4. Exit routines

An exit routine is a program that you provide to perform some service for your application or data. WebSphere Data Interchange calls the exit routine at an appropriate time, and passes it the information needed to accomplish the task. When the task is completed, the exit routine returns the results to WebSphere Data Interchange. For example, WebSphere Data Interchange might pass encrypted data to the exit routine and receive back the data in decrypted form. The information passed to each type of exit routine (its parameters) is described in detail.

There are two types of exit routines. One type of exit routine is a user extension to WebSphere Data Interchange and can interact directly in the current WebSphere Data Interchange session. The other type is an independent program and has no knowledge of the current session. The user extension exits are described first, followed by descriptions of the independent program exits.

There are six user exits that your application programs can use to extend or enhance the capabilities of WebSphere Data Interchange:

- Any-to-any data transformation exit routines will allow users to specify in their data transformation maps that a user-written program should be called
- Field exit routines provide additional processing for application data (translate-to-standard) and EDI standard data (translate-to-application) during translation.
- Transaction exit routines (pre-translation and post-translation) provide additional processing for an entire transaction after translate-to-standard or before translate-to-application.
- Security exit routines protect transaction data through encryption and authentication. Filtering and compression exits are also defined as part of this process.
- Point-to-point network program exit routines get invoked on communication requests for a point-to-point network.
- Message handling exit routines are invoked to process responses from the networks that are not directly supported by WebSphere Data Interchange.

The first four instances are described in the following sections. Point-to-point network programs and message handlers are discussed in Chapter 8, "Interfacing to other networks and applications," on page 599.

The architecture for user extensions to WebSphere Data Interchange is very similar to the architecture used when calling services using the WebSphere Data Interchange API. Services are given logical names (for example, TRANPROC for translation services) and the association between a logical name and a physical load module is accomplished at execution time. User extensions use this same architecture in that exits are identified with a logical name. A logical name for an exit is specified at various points in the customization process, as follows:

1. The logical name for a field exit routine is specified in the `User exit routine name` field on the Special Handling panels during the mapping process.

2. The logical name for a pre-translation routine is specified in the `Pre-translation exit routine` field on the Add Trading Partner Usage for Receiving panel. The logical name for a post-translation routine is specified in the `Post-translation exit routine` field on the Add Trading Partner Usage for Sending panel.

3. The logical names for security exit routines are specified in the network security profile (SECUPROF), as follows:
   - For a compression routine in the `Comp. program` field
   - For a filtering routine in the `Filtering program` field
   - For an encryption/decryption routine in the `Encr. program` field
   - For an authentication routine in the `Auth. program` field

Before using the logical name of a service in one of the fields mentioned above, you must define the exit routine to WebSphere Data Interchange in the user program information profile User Exits. To define an exit routine, you must associate a logical exit name with a physical load module name and with the programming language used to write the exit routine. For a complete description of this profile, refer to the WebSphere Data Interchange User's Guide.

Exit languages

User exits for Windows and AIX can be written in C or C++ (C++ exits must use extern 'C' linkage and are treated as C programs). User exits for z/OS and CICS can be written in Assembler, C, or COBOL. You must specify the implementation language for a user exit in the user program information profile in the `Program language` field of the profile entry. Valid values are:

A   Assembler programs.

C   C language programs. WebSphere Data Interchange supports Microsoft Visual C++ 6.0 complier for Windows, IBM VisualAge® C++ 5.0 compiler for AIX and System Application Architecture (SAA®) C/370 compiler for z/OS and CICS.

J   COBOL programs written using a COBOL compiler other than IBM COBOL II.

K   COBOL II programs. WebSphere Data Interchange fully supports the IBM COBOL II compiler.

Note: References to COBOL in this book refer to both IBM COBOL II programs and non-IBM COBOL II programs. However, non-IBM COBOL II support is limited to field exit routines, pre-translation exit routines, and post-translation exit routines, unless specified otherwise.

Exit linkage editor instructions

When WebSphere Data Interchange determines that a user exit should be called, the logical name of the user exit is used to read the User Exits profile entry to determine the physical load module name (`module name`) and the implementation language (`Program language`). WebSphere Data Interchange issues an operating system load for the load module and then passes control to the exit routine. Information about the exit is retained by WebSphere Data Interchange so if the exit routine is needed more than once, the subsequent requests are answered by again passing control to the exit routine. Because programs are only loaded once, all exits can be defined as

REUSEABLE, but we recommend that they be defined as **REENTRANT**. COBOL programs should use the `RENT compile time` option.

For Assembler, C, and COBOL II exit programs, parameters that are provided to an exit routine can either be above or below the 16 MB line. Therefore, these exit programs must be linked with 31-bit addressing (AMODE 31). These exit routines can reside either above or below the 16 MB line.

The `Load module name` field in the User Exits profile must match the name given to the load module in the linkage editor control statements. This is the name that WebSphere Data Interchange attempts to load and if a load module by this name is not found in any load library (STEPLIB/JOBLIB/LINKLIB), the result is a system 806 ABEND.

Note: In CICS, a PPT entry must exist for the program or a CICS load failure results.

The entry point for a program written in Assembler or COBOL should be the same as the physical load module name. Programs written in C have the following requirements:

* The function name for the C routine must be **MAIN**.
* The linkage editor control statements should have an INCLUDE for the FXXZCITF load module which is in the WebSphere Data Interchange load module data set (EDI.V2R1M0.SEDILMD1).
* FXXZCITF should be made the entry point for the load module. The FXXZCITF program establishes the C environment and then transfers control to the main function in the program.

Any-to-any data transformation

Any-to-any data transformation is enhanced to include the capability to call user written programs. These programs can be written in C or C++. C and C++ functions are called by WebSphere Data Interchange using the DLL calling convention. WebSphere Data Interchange does not support the function name mangling which occurs in C++. These functions have to be extern C blocked. User-written functions can accept up to four optional input string arguments, and will return a string as output. These strings can be character strings or UCS2 unicode strings.

For information regarding the **User Exits** Profile see

Exit Function

The any-to-any mapping function is is called **Exit**. **Exit** can be used within transformation maps to invoke a field exit and will return a string value. The DLL to be loaded and the function within the DLL to be executed are specified in the **User Exits** profile. The function has up to five parameters (the exit routine profile name and four optional strings). Results that are returned can be used to update a variable or target a simple element in an assignment statement. The **Exit** function uses the following format:

```
result Exit(exitname[, parameter[, parameter[, ...]]])
```

Where:

Exit languages

result   A string returned by the user exit

exitname   A string expression that results in the **User Exits** profile member name that contains the function information to be executed.

parameter   Evaluates to a string that wil be passed to the field exit. Parameter can not evaluate to a simple element in the target data. Up to 4 parameters can be passed to the **Exit** function.

You must define field exits in the **User Exits** profile (See WebSphere Data Interchange User's Guide.) If the exit is not properly defined, an error is issued. Zero to four parameters can be passed to the exit and it is the job of the field exit to determine how many parameters to expect.

User written function prototype

```
int  func(
   char*   pOutPut,      /* Pointer to output buffer           */
   long    lOutLeng,     /* Maximum number of output characters */
   void**  ppUEContext,  /* Pointer to user exit context pointer */
   char*   pOption1,     /* Pointer to first optional string    */
   char*   pOption2,     /* Pointer to second optional string   */
   char*   pOption3,     /* Pointer to third optional string    */
   char*   pOption4);    /* Pointer to forth optional string    */
```

Note: If the **User Exits** profile indicates that Unicode strings are involved, then the `char*` fields above would be `wchar_t*` fields.

The user-written function should return a zero to indicate a successful execution and a non-zero to indicate otherwise. A non-zero return code from a user-written function will cause data transformation to stop and an error to be logged. The logged message will include the DLL name, the function name, and the return code value. `LOutLeng` contains the maximum number of output characters specified in the **User Exits** profile. The actual output buffer size is the maximum number of output characters plus a NULL terminator. The output buffer will be initialized to all binary zeroes prior to user-written function invocation. Upon return to WebSphere Data Interchange, WebSphere Data Interchange knows the actual number of output characters because the output is a NULL terminated character string. The `ppUEContext` field is a four-byte place holder. The user-written function may allocate storage for its own use. A pointer to this storage can then be stored in `ppUEContext` and would be available to the user written function across multiple invocations. It is up to the user-written function to eventually free any storage it acquired.

Field exit routines

Field exit routines provide additional processing for application data (translate-to-standard) and EDI standard data (translate-to-application) during translation.

The logical name for a field exit routine is specified in the `User exit routine name` field on the Special Handling panels during the mapping process. The physical characteristics of the exit are defined in the User Exits (ADAMCTL) profile.

During translate-to-standard operations, the translator calls a field exit routine before taking any action with the `Application data` field.

During translate-to-application operations, the translator calls a field exit routine before taking any action with an EDI standard data element.

In either case, the exit routine can inspect the data and do any of the following:

- Verify the data against a predefined set of rules. The exit can tell WebSphere Data Interchange to ignore the data or to ignore the rules through return code settings. For more information, see "Send parameters" on page 316, and "Receive parameters" on page 317.
- Change the value of the data and tell WebSphere Data Interchange to use the new value.
- Save the value of the data for use by other field exit routines.

Field exit routines shipped with WebSphere Data Interchange

Field exit routines allow you to decide whether to use the value of an application field based on the data in another application field. Field exit routines must always be paired. EDICHKI or EDICHKU must always be followed by EDIQQF. These field exit routines are designed to work with send translation only.

The following field exit routines are shipped with WebSphere Data Interchange:

- EDICHKI checks the value of a field and ignores the application data. This exit sets a flag indicating whether the field contained all blanks.
- EDICHKU checks the value of a field and still uses the application data in the field. This exit sets a flag indicating whether the field contained all blanks.
- EDIQQF uses the flag set by either EDICHKI or EDICHKU to determine whether the field contained non-blank data. If the field contained blanks, this exit returns to the translator and tells it to ignore the data. If a literal is associated with this field, the literal is also ignored if the flag indicates an all-blank field.

For example, if you have application field *A* mapped to EDI standard field *X*, and application field *B* mapped to EDI standard field *Y*, but you only want EDI standard field *X* to appear if application field *B* contains data, you could:

1. Map application field *B* to EDI standard field *Y* and to field exit EDICHKI.
2. Map application field *A* to EDI standard field *X* and to field exit EDIQQF.

If you have the above case, but you want both EDI standard fields *X* and *Y* created only if application field *B* contains data, you could:

1. Map application field *B* to EDI standard field *Y* and to field exit EDICHKU.
2. Map application field *A* to EDI standard field *X* and to field exit EDIQQF.

Send parameters

The list below describes the parameters that are passed to a field exit routine during translate-to-standard operations. At this point, the exit routine can inspect the application data and determine how to process the data in creating EDI standard data element values.

See "Field exit parameter language definitions" on page 319 for examples on how to declare the parameters in Assembler, COBOL, and C programs.

The parameter list for field exit routines during translate-to-standard operations consists of the following pointers:

Service name block (SNB)

See "Service Name Block (SNB)" on page 632 for a detailed description of this block. The ZSNBNAME field contains the logical name for the exit specified in the User exit routine name field on the Send Special Handling panel. You can combine many field exit routines into a single physical load module and use the value in ZSNBNAME to determine the reason the exit is being called.

Common control block (CCB)

See "Common Control Block (CCB)" on page 636 for a detailed description of this block. The ZCCBRC field is used to tell WebSphere Data Interchange what further actions should be taken against the current application data. Valid values are:

0   Continues normal processing for the application field. If the return field length is not zero, check to see if the temporary work area contains a new value for the field.

1   Ignores the application field but any default literal processing still applies.

2   Ignores both the application field and the default literal.

3 – 20 Reserved.

21 and higher
An error was detected by the user exit. WebSphere Data Interchange creates a log record (message TR0006) indicating a user exit error occurred, which is treated as a level 1 (data element) error, and does not process the field any further (same as a return code of **1**).

Field value

The actual application field from the data format structure. You can change the field's value directly in this buffer if you do not increase the length. If the value is changed directly in the buffer, however, and the field is mapped more than once, subsequent mappings and exits might see the changed value rather than the original value. To avoid this, use the temporary work area and the return field length to provide the changed value to WebSphere Data Interchange.

Field offset (4 byte binary value)

You can use this value to determine which field you are processing. However, WebSphere Data Interchange does not pass the entire structure to the field exit routine. It passes only the field specified by the translation usage/rule. You cannot use this field to access the entire structure.

Field length (4 byte binary value)

Your exit routine can change the value in this field to a smaller value if the field should be shortened. If you need to increase the size of this field, you must use the temporary work area and the new length field.

Permanent work area (4096 byte buffer)

Your exit routine can use this work area for its processing. WebSphere Data Interchange initializes the work area to binary zeros at the start of translation. After initialization, your exit routine determines the content and format of the work area. The same work area is passed to all exit routines during the translation session.

Temporary work area (1024 byte buffer)

Your field exit routine can use this work area to store a modified version of the input data. WebSphere Data Interchange initializes this work area with blanks before calling the exit routine.

Return field length (4 byte binary value)

This field has a value of zero on entry to the exit routine. A nonzero value in this field, when the exit routine returns to WebSphere Data Interchange, indicates that the data in the temporary work area should be used.

Receive parameters

The list below describes the parameters that are passed to a field exit routine during translate-to-application operations. At this point, the exit routine can inspect EDI standard data elements and determine how to process the data in creating application field values.

See "Field exit parameter language definitions" on page 319 for examples on how to declare the parameters in Assembler, COBOL, and C programs.

The parameter list for field exit routines during translate-to-application operations consists of the following pointers.

Service name block (SNB)

See "Service Name Block (SNB)" on page 632 for a detailed description of this block. The `ZSNBNAME` field contains the logical name for the exit specified in the `User exit routine name` field on the Receive Special Handling panel. You can combine many field exit routines into a single physical load module and use the value in `ZSNBNAME` to determine the reason the exit is being called.

Common control block (CCB)

See "Common Control Block (CCB)" on page 636 for a detailed description of this block. The `ZCCBRC` field is used to tell WebSphere Data Interchange what further actions should be taken against the current application data. Valid values are:

0   Continues normal processing for the EDI standard data element. If the return field's length is not zero, check to see if the temporary work contains a new value for the data element.

1   Ignores the data element, but any default literal processing still applies.

2   Ignores the data element and the default literal.

3 – 20   Reserved.

21 and higher
> An error was detected by the user exit. WebSphere Data Interchange creates a log record (message TR0006) indicating a user exit error occurred, which is treated as a level 1 (data element) error, and does not process the data element any further (same as a return code of **1**). To set the functional acknowledgment code, see the temporary work area.

Data element value

The actual data element value from the segment. You can change the value directly in this buffer if you do not increase the length. If the value is changed directly in the buffer, however, and the data element is mapped more than once, subsequent mappings and exits might use the changed value rather than the original value. To avoid this, use the temporary work area and the return field length to provide the changed value to WebSphere Data Interchange.

Field offset (4 byte binary value)

You can use this value to determine which data element you are processing. However, WebSphere Data Interchange does not pass the entire segment to the field exit routine. It passes only the data element specified by the translation usage/rule. You cannot use this field to access the entire segment.

Field length (4 byte binary value)

Your exit routine can change the value in this field to a smaller value if the data element should be shortened. If you need to increase the size of this field, you must use the temporary work area and the new length field.

Permanent work area (4096 byte buffer)

Your exit routine can use this work area for its processing. WebSphere Data Interchange initializes the work area to binary zeros at the start of translation. After initialization, your exit routine determines the content and format of the work area. The same work area is passed to all exit routines during the translation session.

Temporary work area (1024 byte buffer)

Your field exit routine can use this work area to store a modified version of the input data. WebSphere Data Interchange initializes this work area with blanks before calling the exit routine. If the user exit found an error in the data and rejects the data by setting

the CCB return code to a value greater than **20**, you can use the first byte of this work area to set the functional acknowledgment error code for the AK4 segment. The return value must be numeric, or it is ignored. Other than that, the return value is neither edited or validated. If 999 or CONTRL functional acknowledgments are being created, WebSphere Data Interchange converts the 997 AK4 value to the appropriate 999 or CONTRL value.

Return field length (4 byte binary value)

This field has a value of zero on entry to the exit routine. A non-zero value in this field, when the exit routine returns to WebSphere Data Interchange, identifies that the data in the temporary work area should be used.

Field exit parameter language definitions

The following sections show how to define the parameters provided to field exit routines in Assembler, C, and COBOL.

Assembler definition

The Assembler definitions for parameters are shown below.

```
*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS    DSECT
SNBDATA  DS   A    Address of the SNB
CCBDATA  DS   A    Address of the CCB
FLDDATA  DS   A    Address of data
FLDOFF   DS   A    Address of 4 bytes containing offset
FLDLEN   DS   A    Address of 4 bytes containing length
PERMAREA DS   A    Address of 4096 work area
TEMPAREA DS   A    Address of 1024 temporary area
TEMPLEN  DS   A    Address of 4 bytes to return length value
*
USREXIT  CSECT
    USING PARMS,R1
```

C definition

The C definitions for parameters are shown below.

```
typedef struct WORKAREA
worka;
struct WORKAREA {
    char   working[4096];
};


main(snbdata,
        ccbdata,
        flddata,
        fldoffset,
        fldlength,
        permarea,
```

```
              temparea,
              templength)
       snb   *snbdata;        /* Service name block pointer set up by DI */
       ccb   *ccbdata;        /* Common block pointer used by DI */
       char  *flddata;        /* Pointer to the data */
       long  *fldoffset;      /* Address of offset of field */
       long  *fldlength;      /* Length of the data */
       worka  *permarea;      /* Address of a work area */
       char  *temparea;       /* Area when I can move result data */
       long  *templength;     /* Length of data moved to result area */
```

COBOL definition

The COBOL definitions for parameters are shown below.

```
LINKAGE SECTION.

***********************************************************
*      DF - DATA FORMAT  FIELD DATA                       *
***********************************************************
01  FLD-DATA          PIC X(100).

***********************************************************
*      OFFSET OF DF FIELD DATA WITHIN STRUCTURE           *
***********************************************************
01  FLD-OFFSET        PIC 9(09) COMP.

***********************************************************
*  LENGTH OF THE FIELD BEING PASSED TO THE EXIT ROUTINE   *
***********************************************************
 01  FLD-LENGTH        PIC 9(09) COMP.

***********************************************************
*         THE 4096 BYTE PERMANENT WORK AREA               *
***********************************************************
01  PERM-AREA         PIC X(4096).

***********************************************************
*         THE 1024 BYTE TEMPORARY WORK AREA               *
***********************************************************
01  TEMP-AREA         PIC X(1024).

***********************************************************
*      LENGTH OF DATA IN THE TEMPORARY WORK AREA          *
***********************************************************

01  TEMP-LENGTH       PIC 9(09) COMP.
PROCEDURE DIVISION USING    SNB-DATA
            CCB-DATA
            FLD-DATA
            FLD-OFFSET
            FLD-LENGTH
            PERM-AREA
            TEMP-AREA
            TEMP-LENGTH.
```

Transaction exit routines

Pre-translation and post-translation exit routines provide additional processing for an entire transaction after translate-to-standard or before translate-to-application.

The logical name for a pre-translation routine is specified in the `Pre-translation exit routine` field on the Add Trading Partner Usage for Receiving panel. The logical name for a post-translation routine is specified in the `Post-translation exit routine` field on the Add Trading Partner Usage for Sending panel. The physical characteristics of the exit are defined in the User Exits (ADAMCTL) profile.

Pre-translation exit

During translate-to-application operations, the translator calls a pre-translation routine before any translation takes place. The exit has access to the entire transaction (all data between but not including the transaction set header and transaction set trailer). The pre-translation exit routine can perform any operation on this data. Any modifications made by the exit routine become part of the transaction image. When modifying the data, observe the following restrictions:
* Do not change the length of the data.
* Do not change any character to make it look like a segment delimiter.

Post-translation exit

During translate-to-standard operations, the translator calls a post-translation routine after the translation is complete and before the transaction image is written to the Transaction Store. The exit program has access to the entire transaction (all the data between but not including the transaction set header and transaction set trailer). The post-translation exit routine can perform various modifications on this transaction. In modifying the data, observe the following restrictions:

* Do not change the length of the data.

* Do not change any segment delimiters.

* Do not change any non-segment delimiter character to a segment delimiter character. This restriction ensures that the receiving translator can verify that it is receiving the correct number of segments, specified by the transaction set trailer.

* Do not change the data to appear as though it is a transaction set trailer, group trailer, or interchange trailer.

Pre- and Post-translation exit parameters

The parameter list for pre-translation and post-translation exit routines consists of the following pointers.

Service name block (SNB)

See "Service Name Block (SNB)" on page 632 for a detailed description of this block. The `ZSNBNAME` field contains the logical name for the exit specified in the `Post-translation exit routine` field on the Add Trading Partner Usage for Sending panel or in the `Pre-translation exit routine` field on the Add Trading Partner Usage for Receiving panel. You can combine many field exit routines into a single physical load module and use the value in `ZSNBNAME` to determine the reason the exit is being called.

Common control block (CCB)

See "Common Control Block (CCB)" on page 636 for a detailed description of this block. The ZCCBRC field is used to tell WebSphere Data Interchange what further actions should be taken against the current application data. Valid values are:

0   The modified data returned by the exit routine becomes the transaction image.

1 – 20   Reserved.

21 and higher

      The original data is used as the transaction image. WebSphere Data Interchange creates a log record (message TR0006) indicating a user exit error occurred, which is treated as a level 1 (data element) error.

Transaction image

The complete transaction image between the transaction set header and transaction set trailer (not including the header or trailer).

Compatibility parameter (4 byte binary value containing 0)

Used only to maintain compatibility between the user exit parameter list and the pre-translation and post-translation parameter list.

Image length (4 byte binary value)

The length of the transaction image. The exit routine must not change this value. Unpredictable results occur if this value changes.

Permanent work area (4096 byte buffer)

Your exit routine can use this work area for its processing. WebSphere Data Interchange initializes this work area to binary zeros at the start of translation. After initialization, your exit routine determines the content and format of the work area. The same work area is passed to all exit routines during the translation session.

Temporary work area (1024 byte buffer)

Your exit routine can use this work area as necessary. WebSphere Data Interchange initializes this work area with blanks before calling the exit routine.

Compatibility field (4 byte binary value containing 0)

Used only to maintain compatibility between the user exit parameter list and the pre-translation and post-translation parameter list.

Translation exit language definitions

The following sections show how to define the parameters provided to the pre-translation and post-translation exit routines in Assembler, C and COBOL.

COBOL definition

The COBOL definitions for parameters are shown below.

```
LINKAGE SECTION.


***********************************************************
```

```
*       TRANSACTION IMAGE BETWEEN HEADER AND TRAILER      *
************************************************************

01  TRX-DATA              PIC X(32768).


************************************************************
*       COMPATIBILITY FIELD                               *
************************************************************

01  FLD-COMPAT            PIC 9(09) COMP.


************************************************************
*   LENGTH OF THE TRANSACTION IMAGE                       *
************************************************************

01  TRX-LENGTH            PIC 9(09) COMP.


************************************************************
*        THE 4096 BYTE PERMANENT WORK AREA                *
************************************************************

01  PERM-AREA             PIC X(4096).


************************************************************
*        THE 1024 BYTE TEMPORARY WORK AREA                *
************************************************************

01  TEMP-AREA             PIC X(1024).


************************************************************
*       COMPATIBILITY FIELD                               *
************************************************************

01  FLD-COMPAT1           PIC 9(09) COMP.

PROCEDURE DIVISION USING     SNB-DATA
            CCB-DATA
            TRX-DATA
            FLD-COMPAT
            FLD-LENGTH
            PERM-AREA
            TEMP-AREA
            FLD-COMPAT1.
```

C definition
The C definitions for parameters are shown below.

```
typedef struct WORKAREA worka;
struct WORKAREA {
  char   working-4096-;
};

main(snbdata,
     ccbdata,
     trxdata,
```

```
        fldcomp,
        trxength,
        permarea,
        temparea,
        fldcompat1)
snb    *snbdata;      /* Service name block pointer set up by DI    */
ccb    *ccbdata;      /* Common block pointer used by DI    */
char   *trxdata;      /* Pointer to the transaction image     */
long   *fldcompat;    /* Compatibility field    */
long   *trxlength;    /* Length of the transaction image    */
worka  *permarea;     /* Address of a work area     */
char   *temparea;     /* Area when I can move result data    */
long   *fldcopat1;    /* Compatibility field    */
```

Assembler definition

The Assembler definitions for parameters are shown below.

```
*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS    DSECT
SNBDATA    DS    A    Address of the SNB
CCBDATA    DS    A    Address of the CCB
TRXDATA    DS    A    Address of transaction image
COMPAT     DS    A    Compatibility field
TRXLEN     DS    A    Address of 4 bytes containing length of trx.
PERMAREA   DS    A    Address of 4096 work area
TEMPAREA   DS    A    Address of 1024 temporary area
COMPAT1    DS    A    Compatibility field
*
USREXIT  CSECT
    USING PARMS,R1
```

Get/Put envelope exit and service

WebSphere Data Interchange allows you to retrieve an envelope from storage after an enveloping operation (bypassing the write of the output file), and to provide an envelope to storage before the deenveloping operation (bypassing the read of an input file). For more information, see "Get envelope service" on page 574 and "Put envelope service" on page 575.

The following section describes the Get/Put Envelope exit processing and service when the exit program is defined as a user exit rather than a program. An exit program is specified in the utility control statements (IEXIT, ITYPE, IAREA and IACCESS) or in the TRCB fields (IUSEREXIT, IUSERTYPE, IUSERAREA, IUSERACCESS). A user exit type is defined with an ITYPE value of **UE**. See "Get/Put envelope program" on page 346 for a description of the parameters when the exit is an independent program rather than a user exit.

Get envelope call

The retrieval of envelope data (GET) is initiated by an API envelope call to WebSphere Data Interchange from a user-written API program. WebSphere Data Interchange envelopes the data into internal storage and then checks for a user exit in the IUSEREXIT field of the TRCB. If an exit is found, instead of writing the envelope to a file, WebSphere Data Interchange calls the user exit. A pointer field (IUSERAREA) is provided in the TRCB to allow the user's API program to pass a user-defined area to the user exit. This user-defined area can be used by the API program to provide parameters for the exit program.

The user exit then calls the Get service to retrieve the envelope from the WebSphere Data Interchange internal storage into a user-specified buffer and transfers it to its ultimate destination. Repeated calls must be made to retrieve subsequent pieces of the envelope until a return code of 8 with extended return code of 3 is detected (indicating no more data). No data is returned with this return code.

On successful return from the user exit (CCB return code is zero), WebSphere Data Interchange continues processing. On any unsuccessful return (CCB return code is not zero), a message (TR1255 or TR1256) is logged with the return code, processing terminates, and the return code is returned to the API program.

Put envelope call

The providing of envelope data (PUT) is initiated by an API deenvelope call to WebSphere Data Interchange from a user-written API program. WebSphere Data Interchange checks for a user exit in the IUSEREXIT field of the TRCB. If an exit is found, instead of reading the envelope from a file, WebSphere Data Interchange calls the user exit. A pointer field (IUSERAREA) is provided in the TRCB to allow the user's API program to pass a user-defined area to the user exit. This user-defined area can be used by the API program to provide parameters for the exit program.

The user exit then retrieves the envelope from its origin and calls the Put service to store the envelope data into the WebSphere Data Interchange internal storage from a user-specified buffer. If the API program does not want to pass all of the envelope data into a single buffer, multiple calls can be made to the Put service to store subsequent pieces of the envelope.

On successful return from the user exit (CCB return code is zero), WebSphere Data Interchange deenvelopes the data that was stored by the user exit. On any unsuccessful return (CCB return code not zero), a message (TR1255 or TR1256) is logged with the return code, processing terminates, and the return code is returned to the API program.

FXXZccc stub program

When WebSphere Data Interchange calls a Get Envelope or Put Envelope exit, it passes two parameters in addition to the normal SNB, CCB, and FCB parameters. One is a Get/Put control block that is used as the first parameter to call the Get/Put service. The other is the user-defined area pointed to by TRCB field, IUSERAREA.

Get/Put envelope exit and services

Get or Put envelope exit: The interface to the Get or Put Envelope exit is the WebSphere Data Interchange language-dependent stub (FXXZ*ccc*). The format of the stub is:

```
FXXZccc(SNB,CCB,FCB,GPCB,USERAREA)
```

The parameters for the FXXZccc stub are defined in Table 99.

Table 99. Parameters for the interface to the get or put envelope exit

Parameter	Description
SNB	The service name block that WebSphere Data Interchange passes as input to the exit routine.
CCB	The common control block that WebSphere Data Interchange passes as input to the exit routine.
FCB	The function control block that WebSphere Data Interchange passes as input to the exit routine. This block is initialized by WebSphere Data Interchange with a function code of **1** on an enveloping operation (to allow its use in calling the Get service), and a function code of **2** on a deenveloping operation (to allow its use in calling the Put service).
GPCB	The Get/Put control block that WebSphere Data Interchange passes as input to the exit. This block was initialized by WebSphere Data Interchange with values necessary to call the Get or Put service. It must be passed as the first parameter to the service.
USERAREA	The address of the user-defined area. WebSphere Data Interchange uses the value from the `IUSERAREA` field in the TRCB for this address.

Get/Put envelope service: The interface to the Get or Put service (called by the exit) is the WebSphere Data Interchange language-dependent stub (FXXZccc) and is dependent on the language in which the exit was written. The format of the stub is:

```
FXXZccc(GPCB,CCB,FCB,buffer,length)
```

The parameters for the FXXZ*ccc* stub are defined in Table 100.

Table 100. Parameters for the interface to the get or put service

Parameter	Description
GPCB	The Get/Put control block that WebSphere Data Interchange passed to the exit. This block was initialized by WebSphere Data Interchange with values necessary to call the Get or Put service.
CCB	The common control block that WebSphere Data Interchange passed to the exit routine.
FCB	The function control block that WebSphere Data Interchange passed to the exit routine. This block was initialized by WebSphere Data Interchange with a function code of **1** on an enveloping operation (to allow its use in calling the Get service), and a function code of **2** on a deenveloping operation (to allow its use in calling the Put service).
BUFFER	The address where WebSphere Data Interchange should place the envelope on a Get function or the address from which WebSphere Data Interchange should move the envelope on a Put function.

Table 100. Parameters for the interface to the get or put service (continued)

LENGTH	The address of a 4-byte field. On a Get function call, the field contains the size of the buffer and the actual number of bytes retrieved is returned here. On a Put function call, the field contains the actual number of bytes in the buffer.

Get envelope service return codes: The Get Envelope Service indicates in the CCB whether the Get process was successful. The CCB contains the following return codes (RC) and extended return codes (ERC).

RC = 8, ERC = 1
> The function code in the FCB is invalid.

RC = 8, ERC = 3
> The end of the data (no data is returned with this code).

RC = 8, ERC = 4
> The buffer is too small for minimum data. No data is returned but the minimum size required for the call is returned in the length field.

RC = 0, ERC = 0
> The envelope data was returned.

Put envelope service return codes: The Put Envelope Service indicates in the CCB whether the Put process was successful. The CCB contains the following return codes (RC) and extended return codes (ERC).

RC = 8, ERC = 1
> The function code in the FCB is invalid.

RC = 12, ERC = 2
> A virtual storage failure occurred during the Put function.

RC = 0, ERC = 0
> The envelope data was stored.

Security routines

Security exit routines protect transaction data through encryption and authentication. Encryption protects data against unauthorized viewing. Authentication protects data against unauthorized changes.

Filtering and compression are also defined as part of the encryption and authentication architecture. Filtering ensures that data does not contain characters that could conflict with the control characters used in transmitting an interchange. Filtering is only necessary if the network used to send the data has restrictions on the characters that can be transmitted. Compression can be used to decrease the amount of data that is being transmitted, which results in more efficient transmission and storage of the interchange. If a network is sensitive to the data that is transmitted, a filtering routine must be used to ensure there are no conflicting characters.

You need a way to communicate with your trading partner that data you are sending requires security processing, and a way for your trading partner to tell you that data

being sent to you requires security processing. ANSI has defined how this communication takes place by defining security segments that are placed within the interchange to signal that security processing is required and the exact nature of that processing. The security segments consist of a security header to flag the start of security processing and a security trailer segment to flag the end of security processing. ANSI has also defined that security processing can only occur at specific points within an interchange.

An entire functional group can require security processing and is identified by the S1S security header segment and the S1E security trailer segment.

A transaction can require security processing and is identified by the S2S security header segment and the S2E security trailer segment.

Note: It is possible for a transaction to be secured using the S2S and S2E segments and to exist within a functional group that has been secured with the S1S and S1E segments. Group and transaction security are independent.

The order of security processing has also been standardized. When data is being prepared for sending (ENVELOPE function), the order is:
1. Authentication
2. Compression
3. Encryption
4. Filtration

When data is being received that requires security processing (DEENVELOPE function), the order is:
1. Filtration
2. Decryption
3. Decompression
4. Authentication

A transaction image is always stored in the Transaction Store in clear text. Security processing takes place during the enveloping and deenveloping processes. During the envelope process:
- A transaction image is retrieved from the Transaction Store.
- Security processing takes place against the image.
- The secured image within an interchange is written to the file associated to the network and the interchange is delivered to the trading partner by the network.

During the deenvelope process:
- The secured interchange is read from the file.
- Security processing takes place against the image.
- The transaction image (now in clear text) is written to the Transaction Store and future translations for the transaction do not require security processing.

The following section describes how security processing is enabled for both the send and receive side, and is followed by detailed descriptions of the interface to the user exit routines that provide the encryption, authentication, compression, and filtration functions.

Note: WebSphere Data Interchange provides encryption, authentication, and filtration routines that can be used if they suit your needs. Refer to the network security profile (SECUPROF) description in the for definitions of the routines.

Enabling security during send

Enabling security for transactions that you create is a two-step process. The first step signals that you want security processing and provides the key names to use. The second step defines exactly what processing should take place and provides the data necessary for automatically building the security header and trailer segments (S1S and S1E, S2S and S2E). Proceed as follows:

1. Security is enabled using the following fields on the Add Trading Partner Usage for Sending panel.

   - Group encryption key name
   - Group authentication key name

   Note: If either of the above two fields is provided, during the enveloping process, the group in which this transaction is placed has security processing. S1S and S1E segments are built by WebSphere Data Interchange and the group is encrypted and/or authenticated.

   - Transaction encryption key name
   - Transaction authentication key name

   Note: If either of the above two fields is provided, during the enveloping process, this transaction will have security processing. S2S and S2E segments are built by WebSphere Data Interchange and the transaction is encrypted and/or authenticated.

2. Security is defined by providing a member in the network security profile (SECUPROF). There are two places where the network security profile member ID can be specified. If you have unique security requirements for a particular transaction, use the `Group network security profile name` or the `Trans network security profile name` field on the Trading Partner Usage Overrides for Sending panel. The `Security ID` field in the trading partner profile (TPPROF) contains the default network security profile member ID that is used. The network security profile member provides the following information:

   - Fields to indicate if authentication and/or encryption should take place. If authentication or encryption is asked for in the profile member but no key name is provided on the Add Trading Partner Usage for Sending panel, no authentication or encryption is done. If an authentication and/or encryption key name is provided on Add Trading Partner Usage for Sending panel, but is not requested for in the profile member, no authentication or encryption is done.
   - Fields to indicate what type of filtering, if any, should be done.
   - Data that is used to build the S1S and/or S2S security segment.

Security routines

- Names of the programs that provide the security support being requested.

If security has been enabled and defined, WebSphere Data Interchange invokes the user exits defined in the profile during the enveloping process. The user exits receive the data that must be encrypted, authenticated, compressed, or filtered, and are expected to return the processed data. WebSphere Data Interchange automatically builds the required S1S and S1E, or S2S and S2E, segments.

Enabling security during receive

On the receive side, security is self-enabled. The S1S and S2S segments in the data being received define the security processing that must take place. During the deenveloping process, WebSphere Data Interchange detects the security segments and invokes the necessary security user exits based on the data received. The user exits that get invoked are defined in the network security profile (SECUPROF). The network security profile member used is provided in the `Security ID` field of the trading partner profile member.

On the send side, you can have a different set of programs for each transaction associated with a trading partner. However, on the receive side, only one set of programs can be associated with a trading partner since the network security profile member is identified in the trading partner profile member. For more information on how an exit routine can disperse processing to other exit routines, see "Call exit routine" on page 344.

Security parameters

The following sections define the parameters for the security routines of encryption, authentication, compression, and filtration. There is a lot of similarity in the parameters passed to these routines because, at a very high level, the functions performed by each routine are very similar. A tremendous amount of data can be passed to each routine. Each routine processes the data and returns the data to WebSphere Data Interchange. The amount of data received by these routines is not necessarily the same as the amount of data produced. A compression routine should produce less than it receives, while a filtration routine generally produces more than it receives. Although an encryption routine usually does not change the length of the data, WebSphere Data Interchange does allow the length to change.

The amount of data processed by these routines can be quite large; sometimes there is too much data to fit in the space available to process it. For this reason, the interface to these routines provides a mechanism for processing data in pieces. The size of the buffer used to pass data to these routines is controlled by the `Buffer size` field specified in the network security profile member. If you do not specify a buffer size, WebSphere Data Interchange obtains a buffer large enough to hold all the data up to 32000 bytes. When one of the security routines is called, the parameters indicate the size of the buffer being used, the amount of data in that buffer, and the number of residual bytes remaining that would not fit in the buffer. If the amount of data to be processed exceeds the value in the `Buffer size` field, a Get data routine is provided to retrieve the residual data. Once the data is processed, a Put data routine is provided so that the results can be returned to WebSphere Data Interchange. For more information,

see "Put data routine" on page 343. The put data routine can be called multiple times if the amount of data produced exceeds the value of the `Buffer size` field.

One of the parameters to all of the security routines is the network security profile member data block, which is a copy of the data from the network security profile member (SECUPROF) that caused the exit routine to be invoked.

Encryption routine

The `Encr.program` field in the network security profile (SECUPROF) specifies the logical name of an encryption routine. This logical name must match an entry in the User Exits (ADAMCTL) profile, which contains the physical name of the routine and the implementation language.

The encryption routine receives the parameters described below, encrypts or decrypts the data, and then return the results to WebSphere Data Interchange using a Put data routine. For more information, see "Put data routine" on page 343.

As much data as possible is passed to the encryption routine in an input buffer. If more data must be processed than can fit in the buffer, a Get data routine is provided to obtain the residual data. For more information, see "Get data routine" on page 342.

The encryption routine also has an output buffer for holding the output data. The output buffer is the same size as the input buffer. A Put data routine is provided for putting the results into the buffer. The exit routine must call the Put data routine whenever the output buffer is full, and again at the end of the process to put any data that remains in the output buffer.

A sample encryption routine is provided in "Encryption examples" on page 737.

Encryption parameters

The parameters described below are passed to an encryption or decryption exit routine. For examples of how to declare the parameters in Assembler, C, or COBOL programs, see "Assembler definition" on page 319, "C definition" on page 319, and "COBOL definition" on page 320.

Service name block (SNB): See "Service Name Block (SNB)" on page 632 for a detailed description of this block. The `ZSNBNAME` field contains the logical name for the exit (value in the `Encr. program` field from the network security profile member). You can combine many exit routines into a single physical load module and use the value of `ZSNBNAME` to determine why the exit is being called.

Common control block (CCB): See "Common Control Block (CCB)" on page 636 for a detailed description of this block. The `ZCCBRC` and `ZCCBERC` fields are used to report any errors found by the exit routine. If the return code (`ZCCBRC`) and extended return code (`ZCCBERC`) not zero, WebSphere Data Interchange assumes that encryption or decryption failed and logs a message (TR0849) with the return code and extended return code as part of the message. Valid values for `ZCCBERC` are:

0   The exit terminated without errors.
1   The ZFCBFUNC function code is not valid.

2 – 3 Reserved.

4 The service requested has not been defined in the User Exits (ADAMCTL) profile.

5 – 10 Reserved.

11 The encryption key name is not known.

12 – 20

Reserved.

21 and higher

Errors were defined by the exit routine.

Function control block (FCB): See "Function control block (FCB)" on page 640 for a detailed description of this block. Valid values for the `ZFCBFUNC` field are:

1 Encrypting.

2 Decrypting.

3 Assigning an initialization vector.

Encryption handle: Used to get residual data (see "Get data routine" on page 342) and to put results ("Put data routine" on page 343). A 4 byte binary value.

Key name: The key name to be used during encryption or decryption. A 16 byte value.

Security data block (SECUDB): The network security profile member (SECUPROF) that defined the exit being called.

Buffer size: The size of the input and output buffers. A 4 byte binary value.

Input buffer: Holds the data to be processed. The size of this file is determined by the value set in the `Buffer size` field.

Output buffer: Holds the data that has been processed.The size of this file is determined by the value set in the `Buffer size` field.

Input data length: The amount of data in the input buffer. A 4 byte binary value.

Residual length: The residual number of characters that must be processed but could not be put into the input buffer because of size restrictions. If this value is not zero, the exit routine, after processing all the data in the input buffer, can request the residual data by calling a get data routine ("Get data routine" on page 342). A 4 byte binary value.

Initialization vector: If this is a request to obtain an initialization vector (`ZFCBFUNC` value of **3**), this is where the exit routine returns the value. If this is an encryption request, the value returned in this field should be the encrypted initialization vector. An 8 character value.

Encryption routine language definitions

COBOL definition: The COBOL definitions for the encryption parameters are shown below.

```
       LINKAGE SECTION.

       ************************************************************
       *       HANDLE - USED IN PUTDATA AND GETDATA ROUTINES    *
       ************************************************************

       01  HANDLE          PIC 9(09) COMP.

       ************************************************************
       *      KEYNAME - KEY NAME USED FOR ENCRYPTION/DECRYPTION *
       ************************************************************

       01  KEY-NAME        PIC X(16).

       ************************************************************
       *   THE SIZE OF THE INPUT AND OUTPUT DATA BUFFERS        *
       ************************************************************

       01  BUFFER-SIZE     PIC 9(09) COMP.

       ************************************************************
       *   BUFFER CONTAINING DATA TO ENCRYPT OR DECRYPT         *
       ************************************************************

       01  INPUT-DATA      PIC X(4096).

       ************************************************************
       *   BUFFER CONTAINING THE ENCRYPTED/DECRYPTED DATA       *
       ************************************************************

       01  OUTPUT-DATA     PIC X(4096).

       ************************************************************
       *       LENGTH OF DATA IN THE INPUT DATA BUFFER          *
       ************************************************************

       01  DATA-LENGTH     PIC 9(09) COMP.

       ************************************************************
       * AMOUNT OF DATA THAT REMAINS TO BE PROCESSED THAT WOULD *
       * NOT FIT IN THE INPUT DATA BUFFER.  CALLS TO THE GETDATA*
       * ROUTINE SHOULD BE MADE UNTIL RESIDUAL-LENGTH IS ZERO   *
       ************************************************************
       01  RESIDUAL-LENGTH    PIC 9(09) COMP.

       ************************************************************
       * INITIALIZATION VECTOR AREA                             *
       ************************************************************

       01  INITIALIZATION-VECTOR PIC X(08).

       PROCEDURE DIVISION USING    SNB-DATA
          CCB-DATA
          FCB-DATA
          HANDLE
```

```
            KEY-NAME
            SECDB-DATA
            BUFFER-SIZE
            INPUT-DATA
            OUTPUT-DATA
            DATA-LENGTH
            RESIDUAL-LENGTH
            INITIALIZATION-VECTOR.
```

C definition: The C definitions for the encryption parameters are shown below.

```
main(snbdata,
     ccbdata,
     fcbdata,
     handle,
     keyname,
     secdata,
     bufsize,
     inbuf,
     outbuf,
    datalen,
    residual,
    vector)

    snb    *snbdata;        /* Service name block pointer set up by DI    */
    ccb    *ccbdata;        /* Common block pointer used by DI     */
    fcb    *fcbdata;        /* Function control block    */
    void   *handle;         /* Handle used in getdata and putdata    */
    char   *keyname;        /* Name of key for encryption/decryption    */
    secdb   *secdata;       /* Security profile data block    */
    long   *bufsize;        /* The size of inbuf and outbuf    */
    char   *inbuf;          /* Data to be encrypted or decryption    */
    char   *outbuf;         /* Work buffer to hold result data    */
    long   *datalen;        /* Amount of data in inbuf    */
    long   *residual;       /* Amount of data remaining    */
    char   *vector;         /* Area for the initialization vector    */
```

Assembler definition: The Assembler definitions for the encryption parameters are shown below.

```
*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS     DSECT
SNBDATA    DS    A    Address of the SNB
CCBDATA    DS    A    Address of the CCB
FCBDATA    DS    A    Address of the FCB
HANDLE    DS    A    Address of the handle for getdata/putdata
KEYNAME    DS    A    Address of keyname for encryption/decryption
SECDATA    DS    A    Address of the security data block
BUFSIZE    DS    A    Address of size for INBUF and OUTBUF
INBUF    DS    A    Address of data to be encrypted/decrypted
OUTBUF    DS    A    Address for resultant data
DATALEN    DS    A    Address of amount of data in INBUF
```

```
RESIDUAL   DS   A    Address of amount of data not in INBUF
VECTOR     DS   A    Address of initialization vector
*USREXIT  CSECT
   USING PARMS,R1
```

Authentication routine

The `Auth program` field in the network security profile (SECUPROF) specifies the logical name of an authentication routine. This logical name must match an entry in the User Exits (ADAMCTL) profile, which contains the physical name of the routine and the implementation language.

The authentication routine is expected to return a `Message Authentication Code` (MAC) value produced by the authentication process. A MAC is a cryptographically computed value that is the result of passing text or numeric data through the authentication algorithm using a specific key.

As much data as possible is passed to the authentication routine in an input buffer. If more data must be processed than can fit in the buffer, a get data routine is provided to obtain the residual data. For more information, see "Get data routine" on page 342. A sample authentication routine is provided in "Authentication examples" on page 732.

Authentication routine parameters

The list below describes the parameters that are passed to an authentication exit routine. For examples of how to declare the parameters in Assembler, COBOL, and C programs, see "Assembler definition" on page 319, "C definition" on page 319, and "COBOL definition" on page 320. The parameter list consists of the following pointers:

Service name block (SNB):   See "Service Name Block (SNB)" on page 632 for a detailed description of this block. The `ZSNBNAME` field contains the logical name for the exit (value in the `Auth. program` field from the network security profile member). You can combine many exit routines into a single physical load module and use the value in `ZSNBNAME` to determine why the exit is being called.

Common control block (CCB):   See "Common Control Block (CCB)" on page 636 for a detailed description of this block. The `ZCCBRC` and `ZCCBERC` fields are used to report any errors found by the exit routine. If the return code (`ZCCBRC`) and extended return code (`ZCCBERC`) are not zero, WebSphere Data Interchange assumes that encryption or authentication failed and logs a message (TR0849) with the return code and extended return code as part of the message. Valid values for the `ZCCBERC` field are:

0   The exit terminated without errors.
1   The ZFCBFUNC function code is not valid.
2 – 3   Reserved.
4   The service requested has not been defined in the User Exits (ADAMCTL) profile.
5 – 10   Reserved.
11   The authentication key name is not known.
12 – 20
              Reserved.
21 and higher
              Errors were defined by the exit routine.

Function control block (FCB): See "Function control block (FCB)" on page 640 for a detailed description of this block. The `ZFCBFUNC` field will have one of the following values:

1   Sending
2   Receiving

Authentication handle: Used to get residual data (see "Get data routine" on page 342). A 4-byte binary value.

Key name: The key name to be used during authentication. A 16 byte value.

Security data block (SPDB): The network security profile member (SECUPROF) that defined the exit being called.

Buffer size: The size of the input buffer. A 4 byte binary value.

Input buffer: The data to process. The size of this file is determined by the value set in the `Buffer size` field.

Input data length: The amount of data in the input buffer. A 4 byte binary value.

Residual length: The residual number of characters that must be processed but could not be put into the input buffer because of size restrictions. If this value is not zero, the exit routine, after processing all the data in the input buffer, can request the residual data by calling a Get data routine (see "Get data routine" on page 342). A 4-byte binary value.

MAC value: This is where the MAC value produced by the routine should be returned. A 4 byte binary value.

Authentication exit language definitions

COBOL definition: The COBOL definitions for the authentication parameters are shown below.

```
LINKAGE SECTION.

************************************************************
*      HANDLE - USED IN PUTDATA AND GETDATA ROUTINES      *
************************************************************

01  HANDLE           PIC 9(09) COMP.

************************************************************
*      KEYNAME - KEY NAME USED FOR AUTHENTICATION         *
************************************************************

01  KEY-NAME         PIC X(16).

************************************************************
*    THE SIZE OF THE INPUT AND OUTPUT DATA BUFFERS        *
************************************************************
```

```
01  BUFFER-SIZE          PIC 9(09) COMP.


************************************************************
*    BUFFER CONTAINING DATA TO AUTHENTICATE               *
************************************************************

01  INPUT-DATA           PIC X(4096).


************************************************************
*        LENGTH OF DATA IN THE INPUT DATA BUFFER          *
************************************************************

01  DATA-LENGTH          PIC 9(09) COMP.


************************************************************
*  AMOUNT OF DATA THAT REMAINS TO BE PROCESSED THAT WOULD *
*  NOT FIT IN THE INPUT DATA BUFFER.  CALLS TO THE GETDATA*
*  ROUTINE SHOULD BE MADE UNTIL RESIDUAL-LENGTH IS ZERO   *
************************************************************
01  RESIDUAL-LENGTH      PIC 9(09) COMP.


************************************************************
*  MAC VALUE RETURNED                                     *
************************************************************
01  MAC-VALUE            PIC 9(09) COMP.


PROCEDURE DIVISION USING    SNB-DATA
    CCB-DATA
    FCB-DATA
    HANDLE
    KEY-NAME
    SECDB-DATA
    BUFFER-SIZE
    INPUT-DATA
    DATA-LENGTH
    RESIDUAL-LENGTH
    MAC-VALUE.
```

C definition: The C definitions for the authentication parameters are shown below.

```
main(   snbdata,
    ccbdata,
    fcbdata,
    handle,
    keyname,
    secdata,
    bufsize,
    inbuf,
    datalen,
    residual,
    macvalue)

    snb   *snbdata;        /* Service name block pointer set up by DI   */
    ccb   *ccbdata;        /* Common block pointer used by DI    */
```

Authentication routine

```
fcb     *fcbdata;       /* Function control block     */
void    *handle;        /* Handle used in getdata and putdata    */
char    *keyname;       /* Name of key for authentication    */
secdb   *secdata;       /* Security profile data block     */
long    *bufsize;       /* The size of inbuf and outbuf    */
char    *inbuf;         /* Data to be authenticated     */
long    *datalen;       /* Amount of data in inbuf     */
long    *residual;      /* Amount of data remaining      */
long    *macvalue;      /* Area for MAC value produced     */
```

Assembler definition: The Assembler definitions for the authentication parameters
are shown below.

```
*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS     DSECT
SNBDATA     DS    A    Address of the SNB
CCBDATA     DS    A    Address of the CCB
FCBDATA     DS    A    Address of the FCB
HANDLE    DS    A    Address of the handle for getdata/putdata
KEYNAME     DS    A    Address of keyname for authentication
SECDATA     DS    A    Address of the security data block
BUFSIZE     DS    A    Address of size for INBUF and OUTBUF
INBUF   DS    A    Address of data to be authenticated
DATALEN     DS    A    Address of amount of data in INBUF
RESIDUAL    DS    A    Address of amount of data not in INBUF
MACVAL    DS    A    Address of 4 byte MAC value
*
USREXIT  CSECT
    USING PARMS,R1
```

Compression routine

The `Comp. program` field in the network security profile (SECUPROF) specifies the
logical name of a compression routine. This logical name must match an entry in the
User Exits (ADAMCTL) profile, which contains the physical name of the routine and the
implementation language.

The compression routine is passed the parameters listed below, compresses or
decompresses the data, and then returns the results to WebSphere Data Interchange
using a Put data routine. For more information, see "Put data routine" on page 343.

As much data as possible is passed to the compression routine in an input buffer. If
more data must be processed than can fit in the buffer, a Get data routine is provided
to obtain this residual data. For more information, see "Get data routine" on page 342.

The compression routine also has an output buffer for holding the output data. The
output buffer is the same size as the input buffer. A Put data routine is provided for
putting the results into the buffer. The exit routine must call the Put data routine
whenever the output buffer is full, and at the end of the process to put any remaining
data into the output buffer.

Compression parameters

The parameters for compression exits are exactly the same as those for filtering exits. See the description in the next section for the parameter description.

Filtering routine

The `Filtering program` field in the network security profile (SECUPROF) specifies the logical name of a filtering routine. This logical name must match an entry in the User Exits (ADAMCTL) profile, which contains the physical name of the routine and the implementation language.

The filtering routine is passed the parameters described below, filters or defilters the data and returns the results to WebSphere Data Interchange using a Put data routine. For more information, see "Put data routine" on page 343.

As much data as possible is passed to the filtering routine in an input buffer. If more data must be processed than can fit in the buffer, a Get data routine is provided to obtain the residual data. For more information, see "Get data routine" on page 342.

The filtering routine also has an output buffer for holding the output data. The output buffer is the same size as the input buffer. A Put data routine is provided for putting the results into the buffer. The filter routine must call the Put data routine whenever the output buffer is full, and at the end of the process to put any remaining data into the output buffer.

Filtering parameters

The list below describes the parameters that are passed to a filtering exit routine. For examples of how to declare the parameters in Assembler, COBOL, and C programs, see "Assembler definition" on page 334, "C definition" on page 334, and "COBOL definition" on page 332

Service name block (SNB): See "Service Name Block (SNB)" on page 632 for a detailed description of this block. The `ZSNBNAME` field contains the logical name for the exit (value of the `Filtering program` field from the network security profile member). You can combine many exit routines into a single physical load module and use the value in `ZSNBNAME` to determine why the exit is being called.

Common control block (CCB): See "Common Control Block (CCB)" on page 636 for a detailed description of this block. The `ZCCBRC` and `ZCCBERC` fields are used to report any errors found by the filtering routine. If the return code (`ZCCBRC`) and extended return code (`ZCCBERC`) are not zero, WebSphere Data Interchange assumes the filtering or defiltering routine failed, and logs a message (TR0849) with the return code and extended return code as part of the message. Valid values for the `ZCCBERC` field are:

0   The exit terminated without errors.
1   The ZFCBFUNC function code is not valid.
2 – 3   Reserved.
4   The service requested has not been defined in the User Exits (ADAMCTL) profile.
5 – 20   Reserved.

21 and higher

Errors were defined by the exit routine.

Function control block (FCB): See "Function control block (FCB)" on page 640 for a detailed description of this block. The ZFCBFUNC field will have one of the following values:

1   Filtering or compression

2   Defiltering or decompression

Compression handle:: Used to get residual data ("Get data routine" on page 342) and to put results ("Put data routine" on page 343). A 4 byte binary value.

Security data block (SECUDB): The network security profile member (SECUPROF) that defined the exit being called.

Buffer size: The size of the input and output buffers. A 4 byte binary value.

Input buffer (buffer size): The data to process.

Output buffer (buffer size): A buffer that can be used to hold the processed data. The Put data routine must be used to communicate the results back to WebSphere Data Interchange.

Input data length: The amount of data in the input buffer. A 4 byte binary value.

Residual length: The residual number of characters that must be processed but could not be put into the input buffer because of size restrictions. If this value is not zero, the exit routine, after processing all the data in the input buffer, can request the residual data by calling a Get data routine ("Get data routine" on page 342). A 4 byte binary value.

Filtering exit language definitions

COBOL definition: The COBOL definitions for the filtering parameters are shown below.

```
 LINKAGE SECTION.

***********************************************************
*      HANDLE - USED IN PUTDATA AND GETDATA ROUTINES    *
***********************************************************

 01  HANDLE           PIC 9(09) COMP.

***********************************************************
*   THE SIZE OF THE INPUT AND OUTPUT DATA BUFFERS       *
***********************************************************

 01  BUFFER-SIZE      PIC 9(09) COMP.

***********************************************************
*   BUFFER CONTAINING DATA TO FILTER/COMPRESS           *
```

```
        ************************************************************

        01  INPUT-DATA          PIC X(4096).

        ************************************************************
        *   BUFFER CONTAINING THE FILTERED/COMPRESSED DATA      *
        ************************************************************

        01  OUTPUT-DATA         PIC X(4096).

        ************************************************************
        *         LENGTH OF DATA IN THE INPUT DATA BUFFER       *
        ************************************************************

        01  DATA-LENGTH         PIC 9(09) COMP.

        ************************************************************
        *  AMOUNT OF DATA THAT REMAINS TO BE PROCESSED THAT WOULD *
        *  NOT FIT IN THE INPUT DATA BUFFER.  CALLS TO THE GETDATA*
        *  ROUTINE SHOULD BE MADE UNTIL RESIDUAL-LENGTH IS ZERO   *
        ************************************************************
        01  RESIDUAL-LENGTH     PIC 9(09) COMP.

         PROCEDURE DIVISION USING SNB-DATA
                                  CCB-DATA
                                  FCB-DATA
                                  HANDLE
                                  SECDB-DATA
                                  BUFFER-SIZE
                                  INPUT-DATA
                                  OUTPUT-DATA
                                  DATA-LENGTH
                                  RESIDUAL-LENGTH.
```

C definition: The C definitions for filtering parameters are shown below.

```
main(snbdata,
     ccbdata,
     fcbdata,
     handle,
     secdata,
     bufsize,
     inbuf,
     outbuf,
     datalen,
     residual)
     snb    *snbdata;      /* Service name block pointer set up by DI    */
     ccb    *ccbdata;      /* Common block pointer used by DI    */
     fcb    *fcbdata;      /* Function control block    */
     void   *handle;       /* Handle used in getdata and putdata    */
     secdb  *secdata;      /* Security profile data block    */
     long   *bufsize;      /* The size of inbuf and outbuf    */
     char   *inbuf;        /* Data to be filtered or compressed    */
     char   *outbuf;       /* Work buffer to hold result data    */
     long   *datalen;      /* Amount of data in inbufq    */
     long   *residual;     /* Amount of data remainingq    */
```

Filtering routine

Assembler definition: The Assembler definitions for filtering parameters are shown below.

```
*
* DSECT describing parameters to this routine
* The address of this parameter list is contained in Register 1
* at entry to the program.
*
PARMS    DSECT
SNBDATA    DS    A    Address of the SNB
CCBDATA    DS    A    Address of the CCB
FCBDATA    DS    A    Address of the FCB
HANDLE    DS    A    Address of the handle for getdata/putdata
SECDATA    DS    A    Address of the security data block
BUFSIZE    DS    A    Address of size for INBUF and OUTBUF
INBUF    DS    A    Address of data to be filtered/compressed
OUTBUF    DS    A    Address for resultant data
DATALEN    DS    A    Address of amount of data in INBUF
RESIDUAL    DS    A    Address of amount of data not in INBUF
*
USREXIT    CSECT
USING PARMS,R1
```

Security support routines

The following section describes security support routines.

Get data routine

Authentication, encryption, filtering, and compression can involve large amounts of data. The size of the buffer that the interface uses to pass data to these routines is controlled by the buffer size specified in the network security profile member. If the WebSphere Data Interchange administrator does not specify a buffer size, WebSphere Data Interchange obtains a buffer that is large enough to hold the data; however, the buffer cannot exceed 32 K.

When WebSphere Data Interchange calls a security exit routine, the values that WebSphere Data Interchange passes indicate the size of the buffer being used, the amount of data in the buffer, and the number of residual bytes remaining that would not fit in the buffer. Use the Get data routine to retrieve the residual data.

The interface to the Get data routine is the WebSphere Data Interchange language-dependent stub (FXXZ*ccc*). The format of the stub is:

```
FXXZccc(handle,ccb,fcb,buffer,length)
```

The parameters of the FXXZ*ccc* stub are defined in Table 101.

Table 101. Parameters for the get data routine

Parameter	Description
handle	The authentication, encryption, compression, or filtering handle that WebSphere Data Interchange passes as input to the calling routine.

Table 101. Parameters for the get data routine (continued)

ccb	The common control block that WebSphere Data Interchange passes as input to the calling routine.
fcb	A function control block with a function value of **1 (**get data request).
buffer	The address where WebSphere Data Interchange should place the residual data. This address can be the same address that WebSphere Data Interchange passes to the calling routine.
length	The address of a 4-byte field that contains the maximum number of bytes of residual data that WebSphere Data Interchange should return.

Get data routine return codes: The Get data routine indicates in the CCB whether the get process was successful. The CCB contains the following return codes (RC) and extended return codes (ERC):

RC=8, ERC=1

>   The function code in the FCB is invalid.

RC=8, ERC=3

>   There is no data remaining to get.

RC=0, ERC=0

>   The Get data routine returned the data in the specified buffer.

Notes:

1. If the return codes are zero, the buffer contains the data, and the length indicates the number of characters that the Get data routine returned in the buffer.

2. The Get data routine decreases the number of residual bytes by the number of bytes returned in this request.

Put data routine

The encryption, compression, and filtering routines process input data and create output data that can have the same length as the input data. WebSphere Data Interchange provides these routines with an output buffer that can be used as a work area in creating the output data. Code your encryption, compression, and filtering routines to return the output data to WebSphere Data Interchange by calling the Put data routine. The size of the output buffer is the same size as the input buffer.

The interface to the Put data routine is the WebSphere Data Interchange language-dependent stub (FXXZ*ccc*). The format of the stub is:

```
FXXZccc(handle,ccb,fcb,buffer,length)
```

The parameters of the FXXZ*ccc* stub are defined in Table 102.

Table 102. Parameters for the Put data routine

Parameter	Description
handle	The encryption, compression, or filtering handle that WebSphere Data Interchange passes as input to the calling routine.
ccb	The common control block that WebSphere Data Interchange passes as input to the calling routine.

Table 102. Parameters for the Put data routine (continued)

fcb	A function control block with a function value of **2** (put data request).
buffer	The address where WebSphere Data Interchange should put the data. This address can be the same address as the output buffer that WebSphere Data Interchange passes as input to the calling routine.
length	The address of a 4-byte field that specifies the maximum number of bytes that WebSphere Data Interchange should put.

Put Data routine return codes: The Put data routine indicates in the CCB whether the put process was successful. The CCB contains the following return codes (RC) and extended return codes (ERC):

RC=8, ERC=1
> The function code in the FCB is invalid.

RC=8, ERC=2
> A virtual storage failure occurred while the system put the data.

RC=0, ERC=0
> The data is accepted.

Note: When the return codes are zero, the buffer continues to build additional output data if necessary.

Call exit routine

During receive processing, you might want to have a routine that inspects control blocks or data, and then calls a secondary exit routine based on the results of that inspection. For example, an exit routine can inspect a service code and call another routine that handles a certain type of filtering. The intermediary routine used to call the secondary routine is a Call exit routine. The secondary routine that is called with the call exit service receives the same parameters as the original routine.

The interface to the Call exit routine is the WebSphere Data Interchange language-dependent stub (FXXZ*ccc*). The format of the stub is:

```
FXXZccc(handle,ccb,fcb,routname,0)
```

The parameters of the FXXZ*ccc* stub are defined in Table 103.

Table 103. Parameters for the Call exit routine

Parameter	Description
handle	The encryption, compression, or filtering handle that WebSphere Data Interchange passes as input to the calling routine.
ccb	The common control block that WebSphere Data Interchange passes as input to the calling routine.
fcb	A function control block with a function value of **3** (secondary routine call).
routname	The name of the routine that is being called. This name must be 8 characters long, left-justified, and padded with blanks. This name must also be the same as the name that the WebSphere Data Interchange administrator specifies in the User Exits (ADAMCTL) profile.

Call exit routine return codes: The Call exit routine indicates in the CCB whether the call process was successful. The CCB contains the following return codes (RC) and extended return codes (ERC):

RC=12, ERC=4

> The Call routine could not load or locate the secondary routine that Routname specifies.

RC=*nn*, ERC=nn

> The return code from the secondary routine that was called.

Independent programs

There are two basic types of independent programs that WebSphere Data Interchange can invoke. There are response programs that only apply in the CICS environment (see Chapter 5, "Using WebSphere Data Interchange in the CICS environment," on page 347). There are programs that process data extraction records during the DATA EXTRACTION and MANAGEMENT REPORT commands, and the interface to these programs is described in the following sections.

An independent program does not directly participate in the current WebSphere Data Interchange session. The program is not accessed using the service and exit architecture of WebSphere Data Interchange but is linked using an z/OS LINK macro or the EXEC CICS LINK facility. This being the case, there are no language restrictions imposed by WebSphere Data Interchange for the implementation language of the program. However, in an z/OS environment, the parameters passed to the routine can reside above the 16 MBb line and, therefore, the language must be capable of supporting 31-bit addressing mode (AMODE 31).

Data extract exit

When performing any of the data extract commands described in "Producing management reports from the Transaction Store" on page 10, you can use a program that gets control before the extraction record is written. This program can modify the data or indicate that the record should not be written at all. The name of the program is provided in the USERPGM keyword on the PERFORM command. The name provided is the load module name to which WebSphere Data Interchange will link before writing an output record.

In z/OS, the parameters given to the program point to:

* A 4 byte binary value that is initialized with a zero. Setting the value of this field to any nonzero value indicates that the extract record should not be written.
* The CCB. See "Common Control Block (CCB)" on page 636for a detailed description of the CCB.
* A 4 byte binary value that contains the length of the extract record.
* The extract record.

In CICS, the program receives a COMMAREA containing the data extract exit control block with the format described in Table 104 on page 346.

Independent programs

Table 104. Data extract exit control block

Name	Format	Size	Offset	Description
Return Code	Bin	4	0	Initialized with a zero. Setting the value of this field to any nonzero value indicates that the extract record should not be written.
CCB Address	Bin	4	4	The CCB used to initialize WebSphere Data Interchange.
Record length	Bin	4	8	The number of bytes in the extraction data record (next field).
Data	Char	1-32738	12	The extract record.

Get/Put envelope program

WebSphere Data Interchange allows you to invoke a program that will get control of each interchange created. This program gets control after the interchange has been written to the output file. An exit program is specified in the utility control statements (IEXIT, ITYPE, IAREA and IACCESS) or in the TRCB fields (`IUSEREXIT`, `IUSERTYPE`, `IUSERAREA`, `IUSERACCESS`). An independent program is defined with an `ITYPE` **PG**. For more information on the parameters used when the exit is a user exit program rather than an independent program, see "Get/Put envelope exit and service" on page 324.

This program is invoked through an z/OS LINK macro in the z/OS environment or the EXEC CICS LINK command in the CICS environment. The control block passed to the program contains the address of:

1. The user area. In z/OS, this is the address of the 16 bytes specified by the IAREA keyword. In CICS, this is the address of the UTILCB.
2. The current TRCB.
3. The current TPPDB.
4. The WebSphere Data Interchange CCB.

In z/OS, Register 1 (shown below) contains the address of a pointer to the area in storage that contains the four addresses above. In CICS, these four pointers define the COMMAREA passed to the program.

```
R1 ---> control blk address ---> +0   user area address
              +4  TRCB address
              +8  TPPDB address
              +C  CCB address
```

Chapter 5. Using WebSphere Data Interchange in the CICS environment

Most functions of WebSphere Data Interchange operate similarly regardless of the environment in which they are executed. However, before designing applications to use WebSphere Data Interchange in the CICS environment, programmers should be aware of the differences that do occur. This chapter describes how to interface application programs with WebSphere Data Interchange in the CICS environment, and discusses the issues to be considered. The reader is expected to have a working knowledge of programming in a CICS environment. For information about interfacing WebSphere Data Interchange with other networks and applications, see Chapter 8, "Interfacing to other networks and applications," on page 599.

Running the WebSphere Data Interchange Utility in the CICS environment

The WebSphere Data Interchange Utility is the central point of access to WebSphere Data Interchange functions and the most commonly used interface. In CICS, when you develop an application program to run the WebSphere Data Interchange Utility, you can write the application in any programming language supported by CICS.

The WebSphere Data Interchange Utility provides standard CICS interfaces and can be instructed to use different CICS storage mechanisms. This section describes the issues involved in application programming interaction with the WebSphere Data Interchange Utility. For a description of the specific WebSphere Data Interchange functions that can be accessed by the WebSphere Data Interchange Utility, see Chapter 1, "Using The Utility," on page 1

Invocation options

The following methods can be used to invoke the WebSphere Data Interchange Utility:

- CICS LINK command

  Your application uses this command to link to the EDIFFUT program. This is the simplest interface you can use. With this command, the WebSphere Data Interchange Utility executes synchronously with your application. When the WebSphere Data Interchange Utility finishes running, it issues a CICS RETURN command to give control back to your application. If you use DB2, see "DB2 setup considerations" on page 359.

- CICS START command

  Your application uses this command to start the CICS transaction EDIB. The WebSphere Data Interchange Utility executes in a separate CICS transaction, asynchronously with your application. If you use the CICS START command, consider using response applications, because they are an essential element of the entire process. For more information, see "Response applications" on page 381.

- CICS Automatic Task Initiation (ATI)

  To use this command, you must first add an intrapartition TD queue to the Destination Control Table (DCT) and associate a trigger level and transaction EDIB with the TD queue. As with the CICS START interface, the WebSphere Data

Running WebSphere Data Interchange in the CICS environment

Interchange Utility executes in a separate CICS transaction, asynchronously with your application. If you use ATI, consider using response applications. For more information, see "Response applications" on page 381 for more details.

- EDIW CICS transaction

  You can use this CICS transaction to enter ad hoc PERFORM commands which, in turn, invoke the WebSphere Data Interchange Utility. This is an easy way to test your utility commands before writing an application program.

Passing control information

You must supply control information to the WebSphere Data Interchange Utility to process your request. See "WebSphere Data Interchange Utility control information" on page 368 for the format of the control information.

The method used for passing the control information depends on the invocation method you use. The methods are:

1. When you use the CICS LINK command, you can pass control information in one of two ways:
   - Through a communications area (COMMAREA), using the CICS LINK command with the COMMAREA and LENGTH keywords.
   - Through the Transaction Work Area (TWA). Your application should first move the control information into the TWA, and then issue the CICS LINK command with no COMMAREA or LENGTH specified. With this method, the WebSphere Data Interchange Utility assumes that the control information begins at offset zero of the TWA.

2. When you use the CICS START command, you pass the control information to the WebSphere Data Interchange Utility with the FROM and LENGTH keywords.

3. When you use the CICS ATI command, you write the control information to the associated TD queue by issuing a CICS WRITEQ TD command.

Determining results

The WebSphere Data Interchange Utility passes the results back to the invoking application or forwards them to a response application. The format of the results is the same as the incoming control information, except that additional results fields are filled in. See "WebSphere Data Interchange Utility control information" on page 368.

The results are passed as follows:

1. If you used the CICS LINK command and the control information was supplied to the WebSphere Data Interchange Utility through the COMMAREA, the results are returned to the same COMMAREA. If you used the TWA to pass control information, the results are returned to the TWA on return to your application.

2. If you used the CICS START command or CICS ATI to initiate the WebSphere Data Interchange Utility, you must provide a response application to process the results. The response application can be one of the following:
   - A program to which the WebSphere Data Interchange Utility will issue a CICS LINK to give control to the program. When the CICS LINK command is issued, the COMMAREA and LENGTH keywords are used to pass the results. Your application then obtains the COMMAREA address and processes the results.

- A CICS transaction that the WebSphere Data Interchange Utility will CICS START. When the CICS START command is issued, the `FROM` and `LENGTH` keywords are used to pass the results. Your application then obtains the results by issuing a CICS RETRIEVE.

See "Response applications" on page 381 for more details about response applications.

WebSphere Data Interchange abend return codes

When a CICS abend is detected, WebSphere Data Interchange returns **-8** in the `ZCCBRC` field and the EBCDIC representation of the CICS abend code in the `ZCCBERC` field. This return code combination is global and is not mentioned in upcoming tables.

CICS storage mechanisms

The WebSphere Data Interchange Utility supports the use of MQSeries queues and two different types of CICS storage mechanisms for most of the sequential files it uses:

- Temporary Storage (TS) queues

  TS queues are useful for files that are processed multiple times, or when recovery is not an issue. When TS queues are used as an output destination, such as a print file, exception file, and query file, they are cleared before they are used. The following three WebSphere Data Interchange Utility files are the exception. Records are appended, instead of cleared, when these TS queues are used as output destinations:

  – Envelope TS queues

    If you want the TS queue cleared, use the `CLEARFILE` keyword. Using this keyword clears the TS queue before a receive is processed and after a send is processed. If you do not use the `CLEARFILE` keyword, the WebSphere Data Interchange Utility appends the transmitted data to the TS queue.

    CICS restricts the size of a TS queue to 32-K records. Generally, WebSphere Data Interchange processes only one envelope TS queue (either inbound or outbound) at a time. However, the WebSphere Data Interchange Utility can process more than one inbound TS queue using the DEENVELOPE or the DEENVELOPE AND TRANSLATE commands. When deenvelope only or translate is specified in a continuous receive profile member, the Utility automatically processes the multiple incoming TS queues associated with it. For more information, see "Processing multiple incoming TS queues" on page 351.

  – Functional Acknowledgment TS queues

    If you want the TS queue cleared, use the `CLEARFILE` keyword to clear the TS queue before a receive is processed and after a send is processed. If you do not use the `CLEARFILE` keyword, the WebSphere Data Interchange Utility appends the transmitted data to the queue.

  – Output application TS queues created because of a translate-to-application type command.

- Transient Data (TD) queues

  The two types of TD queues that you can use with the WebSphere Data Interchange Utility are:

  – Extrapartition TD queues

Extrapartition TD queues are z/OS sequential data sets. You can use them for output when further processing is required in z/OS or use them to input data generated in z/OS. Extrapartition TD queues are always appended by the WebSphere Data Interchange Utility. These files are never cleared.

– Intrapartition TD queues

If recovery is a high priority, recoverable intrapartition TD queues are the best choice. If recovery is not a high priority, non-recoverable intrapartition TD queues are a good choice.

The destructive read property of intrapartition TD queues is ideal in eliminating the potential for reprocessed data. If the queues are recoverable, the input queues being read are kept in synchronization with all modified resources. This helps clearly define what resources are part of the unit of work. Restart processing is much easier with recoverable TD queues because the queues are always at a complete unit of work boundary. These files are never cleared.

• MQSeries queues

MQSeries queues are specified using WebSphere Data Interchange MQSeries Queue profile member names and a `File type` of **MQ**. MQSeries queues are very useful if your application receives data from, or passes data to, other applications which are not in your CICS region. These other applications can be z/OS batch applications or applications running on other operating systems such as AIX. MQSeries allows cross-platform communications between applications. In order to use WebSphere Data Interchange MQSeries support, you must have MQSeries installed and running on your CICS region.

Attention: Be careful when selecting the storage mechanism for your application's interface with WebSphere Data Interchange. Data can be inadvertently reprocessed or lost if the wrong storage mechanism is used.

CICS envelope queue alternatives

Normally, envelope files created by WebSphere Data Interchange are TS queues. Whether WebSphere Data Interchange envelopes data to be sent or is called to deenvelope incoming data, the file containing the data is usually a TS queue.

You can override this TS queue convention with a user exit program that uses the Get envelope service (outbound) and the Put envelope service (inbound). You must define the `User exit program name`. When using the WebSphere Data Interchange Utility, you invoke the exit by using the keywords and values listed below on enveloping or deenveloping PERFORM commands.

IEXIT The user exit program name
IACCESS
      M
ITYPE UE

When using the API, these keywords correspond to TRCB fields, and are filled in as above. For simple examples on how to do this, and for more information, see "Get Envelope service example" on page 748 and "Put Envelope service example" on page 748.

Pre- and Post-envelope programs

You can invoke a user-written program directly after an envelope has been created or deenveloped. To invoke this type of program, use an EXEC CICS LINK command from WebSphere Data Interchange. The DFHCOMMAREA contains pointers to the following control blocks: UCB, TRCB, TPPDB, and CCB. When using the WebSphere Data Interchange Utility, you can invoke this type of program by using the keywords and values listed below on enveloping or deenveloping PERFORM commands.

IEXIT The program name

ITYPE PG

When using the API, these keywords correspond to TRCB fields, and are filled in as above. For simple examples on how to do this, see "Inbound envelope program example" on page 748 and "Outbound envelope program example" on page 749.

Processing multiple incoming TS queues

The WebSphere Data Interchange Utility can process up to six inbound TS queues using the DEENVELOPE or the DEENVELOPE AND TRANSLATE commands. This type of processing occurs automatically during continuous receive processing when either DEENVELOPE or DEENVELOPE AND TRANSLATE is specified in the continuous receive profile member. The format of a multiple TSQ control block is described below.

Table 105. Format of the multiple TSQ control block

Name	Offset	Length	Type	Description
RESERVED	0	16	Char	Reserved for WebSphere Data Interchange
MTSQTSQ1	16	8	Char	1st TS queue name
MTSQTSQ2	24	8	Char	2nd TS queue name
MTSQTSQ3	32	8	Char	3rd TS queue name
MTSQTSQ4	40	8	Char	4th TS queue name
MTSQTSQ5	48	8	Char	5th TS queue name
MTSQTSQ6	56	8	Char	6th TS queue name
RESERVED	64	16	Char	Reserved for WebSphere Data Interchange

When WebSphere Data Interchange detects an active continuous receive that involves more than one incoming TS queue, storage is acquired to hold the multiple-TSQ control block and then loaded with the appropriate TSQ names. The address of the multiple TSQ control block is then placed in the UCB and the `Multiple-TSQ` field in the UCB is set to **Y**. Similarly, your application can invoke the Utility with a DEENVELOPE or a DEENVELOPE AND TRANSLATE command and, by following the same steps described above for continuous receive, take advantage of multiple-TSQ processing. For more information, see `FFMTFLG` and `FFMTCBP` in "WebSphere Data Interchange Utility control information" on page 368.

Running WebSphere Data Interchange in the CICS environment

After invoking all necessary response programs, the WebSphere Data Interchange Utility releases the storage it acquired to hold the multiple-TSQ control blocks. If a continuous receive is not set up to deenvelope or translate (in other words, invokes only a continuous receive response program), the response program must release any storage it acquires to hold multiple-TSQ control blocks. Incoming S@#xxxxx TS queues generated by Expedite/CICS are not deleted by WebSphere Data Interchange or by Expedite/CICS. Your response program must delete these queues. If there are multiple TS queues, there will be multiple S@#xxxxx queues.

When the WebSphere Data Interchange Utility processes multiple incoming envelope TS queues, your application must acquire sufficient storage for a multiple-TSQ control block, load it appropriately, and set the `FFMTFLG` and `FFMTCBP` fields in the UCB.

If your application is to invoke the Utility in this manner, only one DEENVELOPE or DEENVELOPE AND TRANSLATE command should be passed into the Utility per Utility invocation. Multiple incoming envelope TS queues can be processed using HOT-DI, the UTILSRV API, or regular Utility invocation.

Ensuring serial processing of WebSphere Data Interchange Utility files

The WebSphere Data Interchange Utility must ensure serial access to all sequential files it uses, such as the print file, envelope file, and tracking file. If two WebSphere Data Interchange Utility transactions are executed concurrently, and both are generating interchanges to the same TS queue, the interchange records in the output file might be intermixed. To avoid this potential problem, the WebSphere Data Interchange Utility uses the CICS ENQ and CICS DEQ commands to serialize access to all sequential files whenever a file is logically opened. Application programs you develop should also support this serialization.

The standard resource naming convention for WebSphere Data Interchange is: `$EDI.resfile.restype`. The resource name must be 16 characters long. The file name (`resfile`) must be 8 characters long, left-justified, and padded with blanks. The file type (`restype`) must be 2 characters long. Valid values for `restype` are:

TM A temporary storage queue in main storage

TS A temporary storage queue in auxiliary storage

TD A transient data queue

For example, the resource name for an application file named AP01, which is defined as a TD queue, would be constructed as follows:

```
$EDI.AP01    .TD
```

Using the CICS ENQ and CICS DEQ commands with the correct resource names ensures that your application references complete information in the sequential files.

Using the previous example, suppose AP01 is an application output file generated as a result of a translation-to-application type operation. If your application issues the proper CICS ENQ command before reading the queue, WebSphere Data Interchange guarantees that the TD queue contains only complete application data transactions.

After reading the TD queue, your application issues the appropriate CICS DEQ command to allow other applications, such as another WebSphere Data Interchange Utility transaction, to have access to the queue.

If the CICS ENQ and CICS DEQ commands are not used in this way, your application must handle the possibility of incomplete work.

Units of work and recovery considerations

When developing a CICS application, it is important to understand what is considered a unit of work by the application code you will be executing (in this case, the WebSphere Data Interchange Utility). In general, a unit of work is defined as the portion of work between two synchronization points. For CICS, the following are synchronization points:

- CICS Transaction Initiation

  Marks the beginning of the first unit of work in a single CICS transaction. This unit of work might be the only unit of work in the transaction if the CICS SYNCPOINT command is not issued.

- CICS SYNCPOINT Command

  Marks the completion of the current unit of work and the start of the next unit of work.

- CICS Transaction Termination

  Marks the end of the last unit of work in a CICS transaction. At this point, an implicit CICS SYNCPOINT occurs. If no CICS SYNCPOINT commands were issued during transaction execution, transaction termination marks the completion of the only unit of work in the transaction.

CICS also supplies the CICS SYNCPOINT ROLLBACK command backs out the current unit of work as though it never existed. CICS also backs out in-process transactions that were interrupted because of an extraneous event such as a power outage or cancellation of the CICS job.

For CICS, a unit of work consists of updates to recoverable resources. If the unit of work is synchronized, all modifications to recoverable resources are applied. If the unit of work is not synchronized, all modifications are backed out. When a backout occurs, none of the updates made to recoverable resources are applied. Whether the backout occurs after one or multiple updates, all changes that occurred after the last synchronization point are removed.

All updates made to recoverable resources within a unit of work are considered a single group. Either the entire group of changes is applied or none are applied, depending on whether the unit of work is synchronized or backed out.

Transaction Store DB2 tables are inherently recoverable. The following user resources can be defined as recoverable and can be accessed by the WebSphere Data Interchange Utility during execution:

- Print file
- Exception file
- Tracking file
- Report file

Running WebSphere Data Interchange in the CICS environment

- Envelope TS queues
- Query file for data extract functions
- Application input files
- Application output files

The three recoverable resource options for these files are intrapartition TD queues, recoverable TS queues, and MQSeries Queues. If you define these files as recoverable, all changes made to them are synchronized when the unit of work is completed successfully or backed out when the unit of work is not completed.

WebSphere Data Interchange bases its recovery scheme on CICS and DB2 recoverable resources and the synchronization or backout of these resources. If a failure occurs and you are using recoverable resources, your application can reinitiate the WebSphere Data Interchange Utility, passing the same control and even the application information as it was given when the failure occurred.

Note: WebSphere Data Interchange recovery does not use any type of journaling or checkpointing internally. WebSphere Data Interchange does not automatically pick up where it left off but must be re-driven by your application.

WebSphere Data Interchange Utility unit of work

The unit of work varies based on the function of the WebSphere Data Interchange Utility being executed. If you want the WebSphere Data Interchange Utility to control the unit of work, set the `Syncpoint interval` field to **0** or **1**. For more information, see the `SYNCVAL` field in "WebSphere Data Interchange Utility control information" on page 368.

Table 106 lists all WebSphere Data Interchange Utility commands and the point at which a CICS SYNCPOINT command can be issued.

Table 106. Unit of work for each datainterchange utility command

For command:	A syncpoint is issued:
TRANSLATE TO STANDARD TRANSLATE TO APPLICATION RETRANSLATE TO APPLICATION	After processing is completed for each transaction.
ENVELOPE REENVELOPE TRANSLATE AND ENVELOPE DEENVELOPE DEENVELOPE AND TRANSLATE	After processing is completed for each interchange. You can override this setting by using `RECOVERY`(**T**), forcing a syncpoint after processing is completed for each transaction.

Table 106. Unit of work for each datainterchange utility command (continued)

For command:	A syncpoint is issued:
SEND	Initially, before the communications program is invoked to place transactions into SEND STARTED status. This syncpoint is for all interchanges being updated. The status of every record in the WebSphere Data Interchange Utility that is part of this send is updated, and a syncpoint is issued. Again, after the communications program is invoked to place transactions into SEND REQUESTED or SEND REQUEST ERROR status. This set of syncpoints is repeated for each file of interchanges sent.
TRANSLATE AND SEND ENVELOPE AND SEND REENVELOPE AND SEND	Initially, after processing is completed for each interchange. You can override this setting by using `RECOVERY(`**T**`)`, forcing a syncpoint after processing is completed for each transaction. Again, after all translation and enveloping are completed, but before the communications program is invoked to place transactions into SEND STARTED status. This syncpoint is issued for all interchanges being updated. A third time, after the communications program is invoked to place transactions into SEND REQUESTED or SEND REQUEST ERROR status. This set of syncpoints is repeated for each file of interchanges sent.
RECEIVE AND DEENVELOPE RECEIVE AND TRANSLATE	Initially, after the receive is completed and management reporting statistics are recorded to the database. This syncpoint occurs even if management reporting is not active. Again, after processing is completed for each interchange. You can override this setting by using `RECOVERY(`**T**`)`, forcing a syncpoint after processing is completed for every transaction.
RECEIVE	After the receive is completed and management reporting statistics are recorded to the database. This syncpoint occurs even if management reporting is not active.
PURGE UNPURGE HOLD RELEASE	For every transaction whose status is updated. Bundled transactions are an exception. The syncpoint is issued after the entire bundle is updated.

Table 106. Unit of work for each datainterchange utility command (continued)

For command:	A syncpoint is issued:
REMOVE TRANSACTIONS	After every 100 deletions or when the number of deletions specified in the `NUMDELS` field is reached. If `STANDALONE`(**Y**) is also specified, only one syncpoint is issued at the end of the entire run.
IMPORT	For each record that is added to the DB2 database. Applies only to the DB2 environment.
UPDATE STATUS PROCESS NETWORK ACKS	For every interchange whose status is updated.
UPDATE STATISTICS	After every 50 updates or when the number of updates specified in the `NUMUPDTS` field is reached.
REMOVE STATISTICS	After every 100 deletions or when the number of deletions specified in the `NUMDELS` field is reached.
PRINT COMMANDS DATA EXTRACT COMMANDS EXPORT CLOSE MAILBOX QUERY	Not applicable.

Using the RECOVERY keyword

For WebSphere Data Interchange Utility commands that include some type of interchange processing, the unit of work is controlled by the recovery level. The following commands include interchange processing:

- DEENVELOPE
- DEENVELOPE AND TRANSLATE
- ENVELOPE
- ENVELOPE AND SEND
- RECEIVE AND DEENVELOPE
- RECEIVE AND TRANSLATE
- REENVELOPE
- REENVELOPE AND SEND
- TRANSLATE AND ENVELOPE
- TRANSLATE AND SEND

You can specify the recovery level by setting the `RECOVERY` keyword as described below:

- Interchange level recovery (**E**)

  All recoverable resources associated with an interchange are included in the unit of work scope. This includes input data read from recoverable intrapartition TD queues, Transaction Store DB2 files, records written to the tracking file, and so on.

  As discussed previously, when a CICS SYNCPOINT command is issued, the updates are applied to all recoverable resources within the unit of work. If a backout occurs, all updates are removed from the system and all resources return to the state they were in before the unit of work began (default in CICS).

- Transaction level recovery (**T**)

All recoverable resources associated with a transaction are included in the unit of work scope. This poses a problem for clean recovery because interchange and group records are added to the Transaction Store with the first transaction of every interchange. A CICS SYNCPOINT is issued after each transaction.

If a backout occurs in the middle of an interchange, any transactions previously translated would already be added to the Transaction Store. To avoid this condition in the CICS environment, it is best to use interchange-level recovery.

Identifying the WebSphere Data Interchange unit of work

When you perform a TRANSLATE TO STANDARD operation, there are some situations where the WebSphere Data Interchange Utility cannot determine the end of a unit of work until it reads application data for the next transaction. In these cases, the unit of work is not limited to one transaction but includes the current transaction and part of the next transaction. Your application can assist WebSphere Data Interchange in determining the end of the unit of work.

- When using C and D record formatted data, the WebSphere Data Interchange Utility cannot detect the end of a transaction without reading the next transaction's C record. When you supply a Z record after the last D record in every transaction, the WebSphere Data Interchange Utility recognizes the Z record as the signal to complete the transaction and end the unit of work, starting a new interchange. WebSphere Data Interchange does not need to detect the transaction's end by reading the next transaction's control record. This improves both transaction level recovery and general processing speed.

- When using raw data, supplying the ending structure name in the data format definition prevents the WebSphere Data Interchange Utility from reading the first structure of the next transaction to determine the end of the current transaction. Unfortunately, if the end structure repeats within the data, this is not possible. To ensure interchange level recovery, do not include interchange breaks in the same input file. By passing separate logical files for each interchange and supplying the ending structure name, the WebSphere Data Interchange Utility does not perform an extra read to determine the end of interchange condition. This preserves interchange level recovery with the application file.

Including WebSphere Data Interchange changes in your application's unit of work

This is an important issue if you want the changes made by the WebSphere Data Interchange Utility to recoverable resources to be synchronized with the changes made by your application to the same recoverable resources. The value you set in the SYNCVAL field controls this synchronization. A value of **-1** tells the WebSphere Data Interchange Utility to suppress CICS SYNCPOINT commands. This puts the application in charge of the unit of work, except when an error occurs while accessing a recoverable resource such as a DB2 table. In this case, the WebSphere Data Interchange Utility issues the CICS SYNCPOINT ROLLBACK command regardless of the desired syncpoint interval. This ensures that all changes in a unit of work can be backed out in case of error. For more information, see the SYNCVAL field description in "WebSphere Data Interchange Utility control information" on page 368.

The following guidelines are suggested if the WebSphere Data Interchange Utility is to be part of your application's unit of work:

Running WebSphere Data Interchange in the CICS environment

- Do not use the `RECOVERY` keyword. Use the default value of **E**). Even though your application controls the unit of work, the WebSphere Data Interchange Utility is sensitive to the recovery level. Using a recovery level of **E** causes WebSphere Data Interchange to hold Transaction Store updates until all CPU-intensive work has completed. This greatly increases the level of concurrency which can be achieved with other CICS transactions performing WebSphere Data Interchange Utility functions.

- Make all application changes to recoverable resources after the WebSphere Data Interchange Utility has returned control to your application. The translation process is CPU-intensive and WebSphere Data Interchange Utility processing throughput is greater than an average CICS transaction. By holding updates to recoverable resources until the end of processing, your application experiences greater concurrency with other applications accessing the same recoverable resources.

- To avoid potential deadlock, always access your recoverable resources in the same order and handle the WebSphere Data Interchange Utility as a logical recoverable resource.

- Do not set the syncpoint interval value to **-1,** and then issue a CICS SYNCPOINT command from a transaction level response program. For more information, see "Response applications" on page 381.

- Use the CICS SYNCPOINT command in continuous receive response programs and utility termination response programs. For more information, see "Response applications" on page 381.

Terminal-attached applications

The throughput time of the WebSphere Data Interchange Utility varies based on the work it is requested to process. In general, throughput time is not sub-second. If your application is associated with terminal users and a sub-second response time is mandatory, you should execute the WebSphere Data Interchange Utility in a separate CICS transaction. This can be accomplished by using the CICS START command or ATI.

You can use response applications to determine whether the execution of the WebSphere Data Interchange Utility was successful. Response transactions can be initiated through the CICS START command with the `TERMID` option. This allows your response application to send a message to the associated terminal, and informs the terminal user of the status of the request made to the WebSphere Data Interchange Utility. You can either:

- Use the RTERMID keyword on the CICS START command when starting transaction EDIB.

- Specify the `Response Terminal ID` field in the WebSphere Data Interchange Utility control information.

Running the WebSphere Data Interchange Utility in a separate CICS region

You can execute the WebSphere Data Interchange Utility in a separate region from the driving application. All three methods of invocation can be used to initiate the WebSphere Data Interchange Utility. To use the LINK interface from a remote CICS region, you must use the distributed LINK function provided in CICS/ESA 3.3 or later. If

the WebSphere Data Interchange Utility is to be executed through the CICS START or ATI commands, then any version of CICS can be used.

When splitting the WebSphere Data Interchange Utility into a separate region, you must use remote resources, such as TD or TS queues, so your CICS systems programmer must be involved if you choose this option. It is much simpler to implement the WebSphere Data Interchange Utility in the same region as the application, but the WebSphere Data Interchange Utility provides the flexibility to allow the split, if necessary.

DB2 setup considerations

WebSphere Data Interchange is shipped with a sample Resource Control Table (RCT). DB2 uses the RCT to determine, among other things, DB2 plan authorization. All transactions supplied by WebSphere Data Interchange have access to the WebSphere Data Interchange DB2 plan. If you are developing an application that issues the CICS LINK command to program EDIFFUT to gain access to WebSphere Data Interchange Utility services, your CICS systems programmer must add your CICS transaction IDs to the RCT. The following example RCT demonstrates the structure of a typical RCT.

The following assumptions are made in this example:

1. No more than five online users, EDIA entry, increase if more are required.
2. No more than tentranslations or other Expedite transactions allowed concurrently, EDIB entry, increase if more are required and system resources are available.
3. A maximum of 17 DB2 threads allowed in the pool, the sum of THRDM values for all ENTRY statements in the pool.
4. A DB2 TCB limit of 20, three + the sum of THRDM values for all POOL statements in the RCT.

```
    DSNCRCT TYPE=INIT,SUBID=DB93, X
        DPMODI=EQ,ERRDEST=(CSMT,*,*),ROLBI=YES,SNAP=X,      X
        STRTWT=YES,SHDDEST=CSSL,SUFFIX=00,THRDMAX=20,          X
        TRACEID=192,TWAITI=YES
    DSNCRCT TYPE=POOL,AUTH=(TXID,*,*), X

        DPMODE=EQ,PLAN=DEFAULT,ROLBE=YES,THRDM=17, X
        THRDA=17,THRDS=0,TWAIT=YES,TXID=POOL
    *----------------------------------------------------------------*
    *  A minimum of 2 threads is required for EDIE, EDIX             *
    *----------------------------------------------------------------*
    DSNCRCT TYPE=ENTRY,AUTH=(SIGNID,*,*),THRDM=2,THRDA=2,          X
        TXID=(EDIE,EDIX),ROLBE=YES,TWAIT=POOL,PLAN=DIENU22C
    DSNCRCT TYPE=ENTRY,AUTH=(SIGNID,*,*),THRDM=5,THRDA=5,          X
        TXID=(EDIA),ROLBE=YES,TWAIT=POOL,PLAN=DIENU22C
    *----------------------------------------------------------------*
    *                   WebSphere Data Interchange
    *  Customer written transactions which EXEC CICS LINK to program
    *  EDIFFUT should also be added to the EDIB macro entry. Note:
    *  LGO1, IMR1, and IST1 are Expedite/CICS transactions and can be
    *  removed if Expedite/CICS is not used.
    *----------------------------------------------------------------*
    DSNCRCT TYPE=ENTRY,AUTH=(SIGNID,*,*),THRDM=10,THRDA=10, X
```

```
         TXID=(EDIB,EDID,EDIG,EDIM,EDIQ,EDIR,          X
         EDIS,EDIT,EDIV,EDIW,EDIZ,LGO1,IMR1,IST1),          X
         ROLBE=YES,TWAIT=POOL,          X
         PLAN=DIENU22C
      DSNCRCT TYPE=FINAL
      END
```

DB2 thread pool considerations

When setting up your DB2 thread pools, make sure to establish one DB2 thread pool for EDIB (and other transactions), a second DB2 thread pool in the CICS RCT for EDIX and EDIE, and a third DB2 thread pool for EDIA transactions.

The number of threads in the pool for EDIB (and other transactions) is determined by the processing resources available to the system on which WebSphere Data Interchange is running. The number of DB2 threads represents the number of EDIB pool transactions that will be executed concurrently. The maximum number of threads needed is the sum of all TRANCLASS values for the transactions in the EDIB pool. The minimum is three, but five or more is typical. The pool transactions include EDIB, EDID, EDIG, EDIQ, EDIR, EDIS, EDIV, EDIW, EDIZ, LGO1, IMR1, and IST1. You can also include user transactions that initiate the WebSphere Data Interchange Utility (EDIFFUT) or CICS START EDIB.

The EDIX/EDIE pool must have two or more threads available. EDIX is a long running task and requires a thread for processing. As part of the syncpoint process, EDIX gives up its thread between uses. Since EDIX communicates with the translator (EDIB) using WAIT/POST logic, EDIB will WAIT until EDIX satisfies its request. EDIX requires the WebSphere Data Interchange Transaction Store lock (EDITSLT) to complete its processing. EDIE is a started task, triggered by CICS when an entry is written to the EDI3 intrapartition TD queue, and requires a thread for processing. More than one EDIE transaction can be started by CICS, but the transactions will serialize so that two threads are sufficient. No more than 3 threads are recommended for this pool.

The EDIA pool is for the long running task, EDIA. EDIA is the administrative tool for WebSphere Data Interchange and is a user-operated, conversational transaction that updates DB2 tables. Because an EDIA transaction must be running for each user, a DB2 thread is required for each instance of EDIA. If the TRANCLASS for EDIA is set at 10, 10 users can operate within the CICS EDIA facility concurrently. Set the DB2 thread value at 10 to allow each concurrent access. Keeping EDIA in a separate pool prevents the TRANCLASS instances of EDIA from allocating all the DB2 threads available to transactions.

Make sure to establish TRANCLASS values for EDIB, EDIA, and EDIW and for any transactions that CICS LINK to EDIFFUT. The values you set depend on the available system resources. The sum of the TRANCLASS values for all such transactions must be at least two less than CICS Max Task value. This will allow EDIX and EDIE to start when all Facility/Utility/Translator transactions are running.

CICS processing can deadlock if EDIB is waiting for an EDIX or EDIE complete. A CICS Max Task condition which does not allow EDIE or EDIX to start will cause a

deadlock in CICS. This will occur if all CICS Max Task entries are being used by EDIB, EDIA, or other transactions and are waiting on service from EDIX or EDIE.

CICS startup considerations

WebSphere Data Interchange cannot run with `Transaction Isolation` set to **On**. The CICS SIT parameter must be set to **No**.

The TS queue EDICSDA contains a pointer to the Service Director Global Area (CSD). The CSD is shared between WebSphere Data Interchange transactions EDIA and EDIT, or EDIB and EDIT.

Running WebSphere Data Interchange in a HOT-DI environment

HOT-DI was developed to meet the ever-increasing need to improve performance and is used to keep multiple DI tasks running, thus eliminating the performance overhead required to start and end the translator for each individual transaction. During normal translation processing, WebSphere Data Interchange needs information to determine how to process data. The translator must get storage, read translation and validation tables, read profiles, read the mapping instructions generated in the control string, and so on. This is all part of WebSphere Data Interchange's initialization routine. Each execution of the WebSphere Data Interchange Utility, whether executed by PERFORM commands or through the API, causes the translator to repeat all of these initialization steps.

With HOT-DI, repetitive initialization is eliminated by allowing a user-written application to initialize WebSphere Data Interchange once, invoke WebSphere Data Interchange's translation services multiple times, and terminate WebSphere Data Interchange once. Initializing multiple HOT-DI sessions increases translation concurrency and overall throughput. Any commands that can be issued with PERFORM statements can be executed by HOT-DI. However, the intent of HOT-DI is to perform translation at a very high rate. HOT-DI does this particularly well when processing large volumes of small transactions being received or sent to many different trading partners.

The HOT-DI concept requires that WebSphere Data Interchange objects, such as usages/rules and profiles, be read only at initialization time. This means that object updates performed while HOT-DI is executing are not picked up until the HOT-DI tasks are terminated and re-initialized. This implementation involves using the WebSphere Data Interchange API and the Utility, which requires user-written applications or tasks. This can be achieved several ways. One scenario is outlined below.

HOT-DI implementation requires WebSphere Data Interchange and can only be used with Expedite/CICS when using the PERFORM commands to send data. It is not currently supported by environments using WebSphere Data Interchange's continuous receive mailbox function.

Initializing HOT-DI

Initialization syntax

```
DIINIT
FXXZccc(SNB,CCB1,FCB,applid,sysid)
FXXZccc(SNB,CCB2,FCB,applid,sysid)
FXXZccc(SNB,CCB3,FCB,applid,sysid)
```

DITCOM CICS task WebSphere Data Interchange termination	FXXZccc

- DI INIT FCB=4
   CCB=1
- DI INIT FCB=4
   CCB=2
- DI INIT FCB=4
   CCB=3
.
.
- Write to TSQ
	API for
	WebSphere Data Interchange
	Environment
	Services

FXXZccc

API for
WebSphere Data Interchange
Environment
Services

CCB ID TSQ

DI CCB=1 AVAIL-Y/N
DI CCB=2 AVAIL-Y/N
DI CCB=3 AVAIL-Y/N
.
.
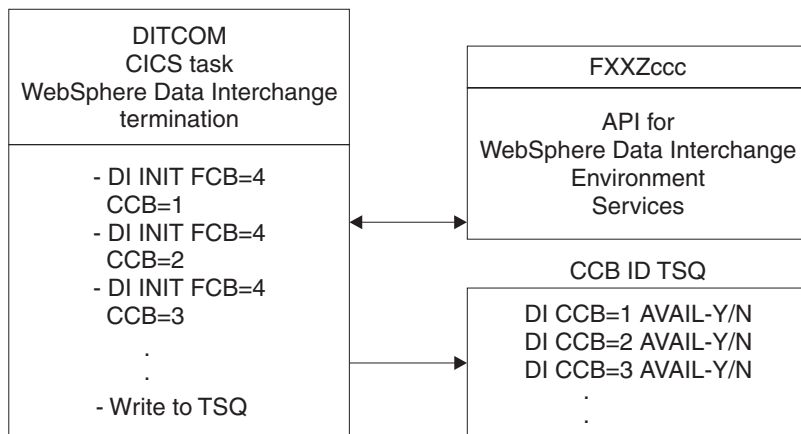
Figure 18. HOT-DI initialization diagram

Initializing WebSphere Data Interchange

A user-written CICS task, such as DIINIT, is required to initialize WebSphere Data Interchange but after initialization, this task should end. Field information and pointers to storage areas required by WebSphere Data Interchange are kept in the CCB. Part of the HOT-DI initialization process is to save this CCB so that the translation requests submitted afterward can bypass the initialization phase.

For more information about the initialization request syntax and control block parameters, see "Initializing the environmental API" on page 438.

Initialization syntax

`FXXZccc(SNB,CCB,FCB,applid,sysid)`

- Set the `ZSNBNAME` field in the SNB to **ENVSERV**.
- Set the `ZFCBFUNC` field in the FCB to **4**. WebSphere Data Interchange will use the shared storage option when acquiring main storage.
- The CCB must be different for each initialized WebSphere Data Interchange task and must be used when requesting other WebSphere Data Interchange utility services, such as translation. The CCB area obtained before initialization and passed for all subsequent calls is obtained through a shared GETMAIN command; for example:

```
EXEC CICS
    GETMAIN SET (CCB-POINTER) LENGTH (608) SHARED
END-EXEC
```

Initializing multiple HOT-DI tasks

You can use a CICS task, such as DIINIT, to initialize HOT-DI as many times as your system's storage allows (usually five or six tasks). A main storage area must be acquired for each HOT-DI initialized. The size of the storage required is equal to the size of a CCB. You can save the address of each CCB in a TSQ or TDQ known to the CICS task. Placing the CCB address in a TSQ allows external access to an initialized HOT-DI CCB to be achieved. You should associate a user flag with each CCB to show the processing status (either **AVAILABLE**, indicating no current translation is being performed or **BUSY**, indicating a translation or other WebSphere Data Interchange service is in process).

HOT-DI processing considerations

A user-written CICS task (such as GETDI) controls when a WebSphere Data Interchange service (such as translation) is started in one of the HOT-DI sessions. To start the service, the CICS task must know the user service desired and the name of an available HOT-DI session. To maintain synchronization when running concurrent HOT-DI tasks, you must use separate files for print, exception, reporting, tracking, and query. For more information, see "WebSphere Data Interchange CICS considerations" on page 803.

The CICS task (GETDI):

- Loops until an event occurs that signals EDI translation or a desired service.
- Obtains an available HOT-DI session (querying the user flag associated with the HOT-DI CCB address). This might involve reading a list of TS queue names of started HOT-DIs.
- Sets the User flag to **BUSY** for the selected HOT-DI session.
- Starts a CICS task (such as DIPROC) for processing using the CCB identifier. The user-written CICS task uses the CCB identifier to call the Utility Service for processing.
- After processing is completed, sets the User flag for the HOT-DI session to **AVAILABLE**.

Running WebSphere Data Interchange in a HOT-DI environment
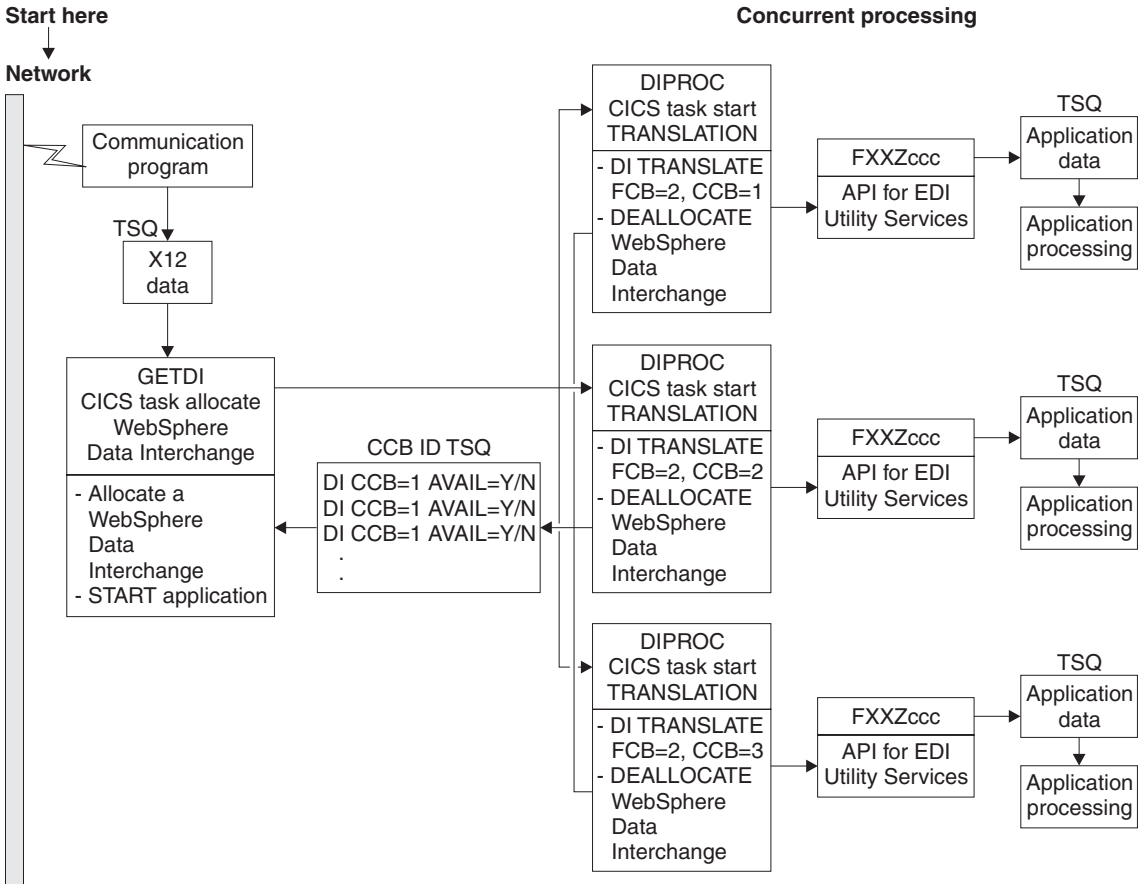
Figure 19. HOT-DI non-Expedite/CICS - inbound diagram

Call utility services

For information about the Call utility service request syntax and control block parameters, see "Utility service API" on page 440.

Processing function syntax

`FXXZccc(SNB,CCB,FCB,UTILCB)`

- Set the `ZSNBNAME` field in the SNB to **UTILSRV**.
- The CCB should be the CCB identifier passed from the allocate step.
- Set the `ZFCBFUNC` field in the FCB to **2**. WebSphere Data Interchange will process in HOT-DI mode.
- Set the `BATFLG` flag, indicating whether or not the utility is being invoked from the API. When the API is used to call the UTILSRV service (as in the case of HOT-DI), this field must be set to **B**.

- Initialize the UTILCB appropriately for the processing request. For information about the format of this control block, see "WebSphere Data Interchange Utility control information" on page 368.

WebSphere Data Interchange return code considerations

Your user-written program can interrogate WebSphere Data Interchange's API return codes (`ZCCBRC` field) and extended return codes (`ZCCBERC` field) in the CCB. For descriptions of these codes, see WebSphere Data Interchange Messages and Codes.

The mapping global accumulators are not reset between HOT-DI invocations. If a global variable is used to conditionally execute a map switch with the WebSphere Data Interchange variable DIMAPSWITCH, you must ensure that global variables are handled appropriately in both translation usages/rules.

Terminating WebSphere Data Interchange

A user-written CICS task (such as DITERM) is needed to terminate each active HOT-DI task. Termination involves making a WebSphere Data Interchange termination request using the CCB, and then releasing the main storage area that held the CCB. For more information about the Termination request syntax and control block parameters, see "Terminating the API" on page 441.

If any CCB is busy for a specified number of attempts on each WebSphere Data Interchange session, the termination step should loop until all tasks have been terminated, or until the specified number of attempts to terminate has been reached.

Termination function syntax

```
FXXZccc(SNB,CCB,FCB)
```

- Set the `ZSNBNAME` field in the SNB to **ENVSERV**.
- Set the `ZFCBFUNC` field in the FCB to **2**. WebSphere Data Interchange will use shared GETMAINs.

Terminating HOT-DI

Termination syntax

```
DITERM
FXXZccc(SNB,CCB1,FCB)
FXXZccc(SNB,CCB2,FCB)
FXXZccc(SNB,CCB3,FCB)
```

Running WebSphere Data Interchange in a HOT-DI environment

```
┌─────────────────────────────┐
│         DITCOM              │
│        CICS task           │              ┌──────────────────────────┐
│  WebSphere Data Interchange │              │       CCB ID TSQ         │
│        termination         │              ├──────────────────────────┤
├─────────────────────────────┤              │  DI CCB=1 AVAIL-Y/N      │
│     - Read TSQ             │              │  DI CCB=2 AVAIL-Y/N      │
│                            │◄─────────────│  DI CCB=3 AVAIL-Y/N      │
│     - DI TERMINATE        │              │            .             │
│       CCB=1               │              │            .             │
│     - DI TERMINATE        │              └──────────────────────────┘
│       CCB=2               │
│     - DI TERMINATE        │              ┌──────────────────────────┐
│       CCB=3               │◄────────────►│        FXXZccc           │
│       .                   │              ├──────────────────────────┤
│       .                   │              │        API for           │
└─────────────────────────────┘              │ WebSphere Data Interchange│
                                            │      Environment         │
                                            │       Services           │
                                            └──────────────────────────┘
```
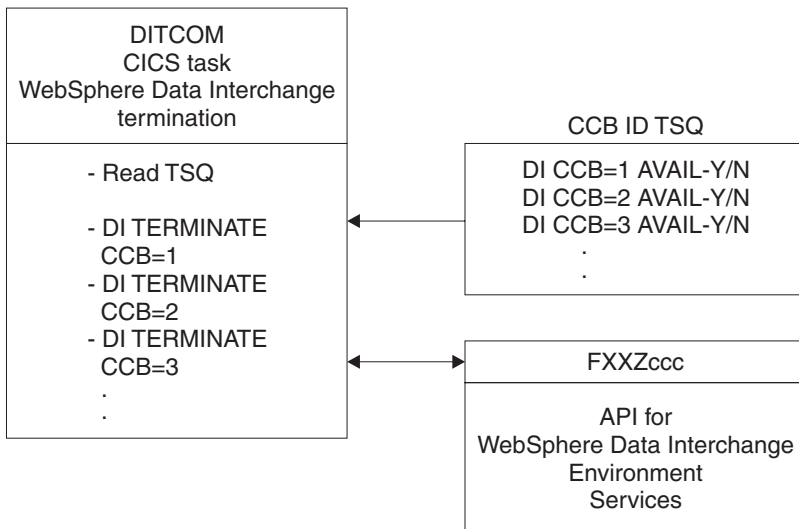
Figure 20. HOT-DI termination diagram

Outbound communications

Outbound communications can be requested using WebSphere Data Interchange's API with Utility services or Communication services.

For information about the Utility service or Communication service function request SYNTAX and control block parameters, see Chapter 1, "Using The Utility," on page 1 or "Communication services" on page 540.
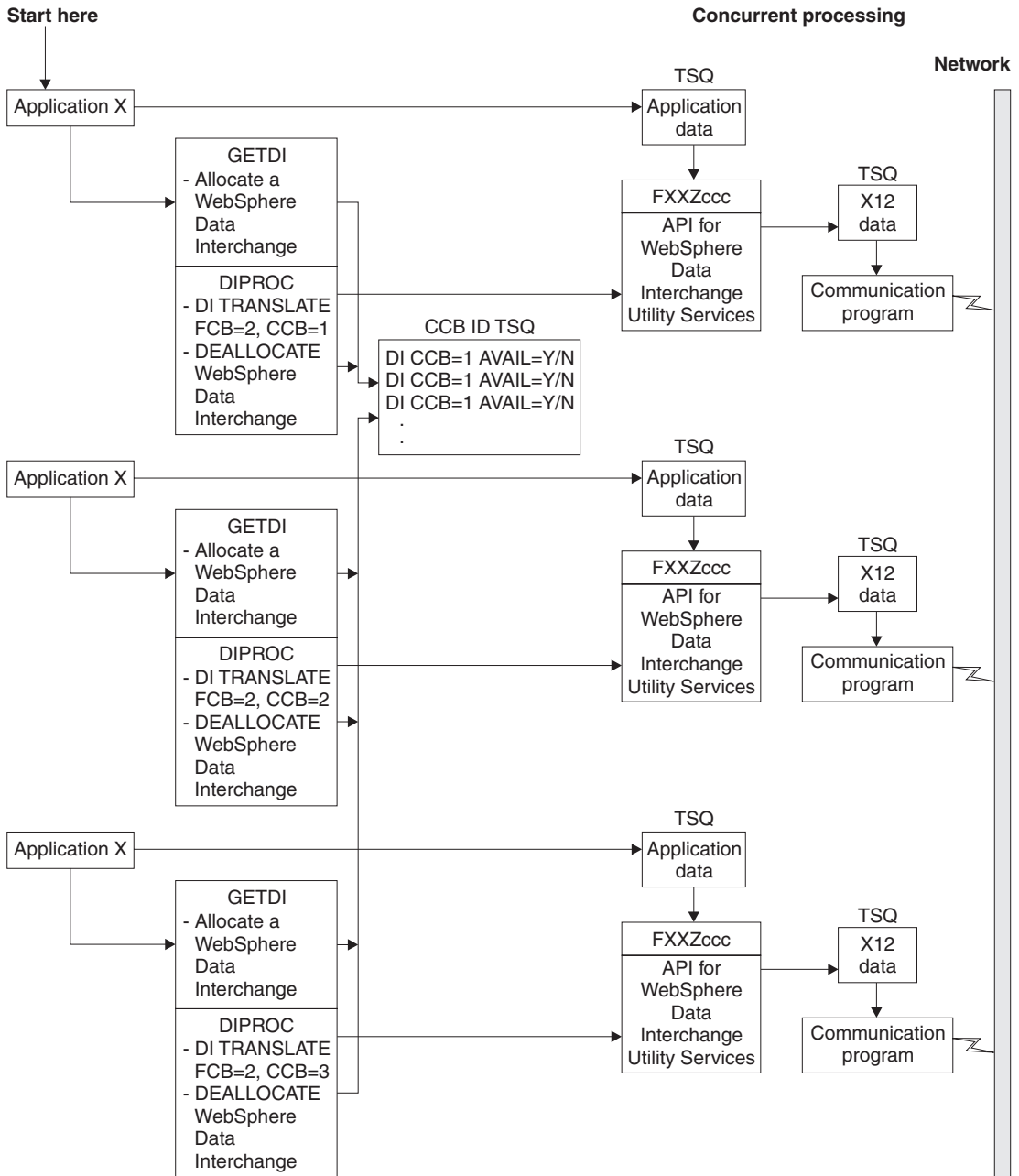
Figure 21. HOT DI non-Expedite/CICS - outbound diagram

WebSphere Data Interchange Utility control information

When your application invokes the WebSphere Data Interchange Utility, control information must be passed. The table below describes the format of the area given to the WebSphere Data Interchange Utility and includes the resulting control information given back to the invoking application or passed on to your response application is included. All fields are given to your application, not just the fields set by WebSphere Data Interchange.

The Result field indicates whether the field is used internally by WebSphere Data Interchange or is set by WebSphere Data Interchange before control is returned to the initiating application or passed on to a response application. The initiating program or response programs should inspect these fields to determine if the execution of the WebSphere Data Interchange Utility was successful. Some of these fields also contain names of TS queues that must be processed further. Valid values for the Result column are:

Y   The field is set by WebSphere Data Interchange before control is returned to the initiating application or passed on to a response application.

N   The field is used internally by WebSphere Data Interchange. Initialize it with blanks or binary zeros, and do not modify it after initialization.

blank   The field is used by WebSphere Data Interchange but does not require special consideration.

Format of WebSphere Data Interchange Utility control information

Table 107. Format of WebSphere Data Interchange utility control information

Name	Type	Offset	Length	Result	Description
SYNCVAL	Bin	0	4		Units of work before syncpoint
CMDP	Bin	4	4		Command string address
CMDLEN	Bin	8	4		Command string length
CMDNAME	Char	12	8		Command file name
CMDTYPE	Char	20	2		Command file type
DELIMITER	Char	22	1		Command value delimiter
PRTNAME	Char	23	8		Print file name
PRTTYPE	Char	31	2		Print file type
RPTNAME	Char	33	8		Report file name
RPTTYPE	Char	41	2		Report file type
EXCPNAME	Char	43	8		Exception file name
EXCPTYPE	Char	51	2		Exception file type
TRAKNAME	Char	53	8		Tracking file name
TRAKTYPE	Char	61	2		Tracking file type
QRYNAME	Char	63	8		Query file name
QRYTYPE	Char	71	2		Query file type

Table 107. Format of WebSphere Data Interchange utility control information (continued)

Name	Type	Offset	Length	Result	Description
APPLID	Char	73	8		Override application ID
LANGID	Char	81	6		Override language ID
RESPID	Char	87	8		Response program or transaction ID
RESPTYP	Char	95	2		Response type
RTERMID	Char	97	4		Response terminal ID
USRFLD	Char	101	16		User field to pass data
SYSID	Char	117	8	X	CICS system ID override
RESPFLAG	Char	125	1	Y	Response indicator
FILETYP	Char	126	2	Y	Network acknowledgment file type
ECBP	Bin	128	4		ECB address
CCBRC	Bin	132	4	Y	Severity code (UTILSEV)
CCBERC	Bin	136	4	Y	Condition code (UTILCCODE)
FILEID	Char	140	8	Y	Envelope TSQ/Net acknowledgment file
APPFILE	Char	148	8	Y	Translated data TS Queue
ABNDCODE	Char	156	4	Y	CICS abend code if one occurred
THANDLE	Char	160	20	Y	Transaction handle
FFIEHDR	Bin	180	4	Y	Information Exchange long header address
FARC	Bin	184	4	Y	Functional acknowledgment return code
FAERC	Bin	188	4	Y	Functional acknowledgment extended return code
FABUILT	Char	192	1	Y	Functional acknowledgment built flag
FFMTFLG	Char	193	1		EDI standard multi-TSQ flag
USERSYNC	Char	194	1		User syncpoint flag
APMTFLG	Char	195	1		Application multi-TSQ flag
USERCOND	Bin	196	4		User condition code pointer
APMTCBP	Bin	200	4		Application multi-TSQ pointer
RES1	Char	204	10		Reserved for future use
FFNOCNV	Char	214	1		No return code conversion flag
FANAME	Char	215	8		Functional acknowledgment file name
RESPACTV	Char	223	1	X	Response program active flag

WebSphere Data Interchange Utility control information

Name	Type	Offset	Length	Result	Description
WORKNAME	Char	224	8	X	Name of workfile
BATFLG	Char	232	1		Batch or UTILSRV API (HOTDI) flag
NOEXCP	Char	233	1	X	No exception file
INVPARM	Char	234	1	X	Network program invalid parameter
LOGACTV	Char	235	1	X	Logging CE0050 active
FFUSADDR	Bin	236	4	X	Pointer to this block
CCBP	Bin	240	4	X	Pointer to the CCB
FFMTCBP	Bin	244	4		EDI standard multi-TSQ control block pointer

WebSphere Data Interchange Utility control information field descriptions

SYNCVAL
The number of units of work completed before WebSphere Data Interchange issues a CICS SYNCPOINT command. For a complete description of the unit of work for each command, see "WebSphere Data Interchange Utility unit of work" on page 354. 4 byte binary value. The IMPORT command ignores this value. Valid values are:

0 or 1 WebSphere Data Interchange issues the syncpoints (default).

–1 Your application issues the syncpoints. WebSphere Data Interchange does not issue any syncpoints.

other WebSphere Data Interchange issues the specified number of syncpoints.

CMDP
The command string address of the command language input in main storage. Set this field only if the application remains active while WebSphere Data Interchange is running. The storage is treated as read-only, so lowercase characters are not changed to uppercase characters. For this reason, all keywords must be in uppercase. Passing the commands in main storage improves the performance of the WebSphere Data Interchange Utility.

CMDLEN
The length of the command language string in main storage. If the main storage option is not used, you must set this field to **0**, to indicate that values in the command file name (CMDNAME) and command type (CMDTYPE) fields should be used instead. 4-byte binary value.

CMDNAME

The file name of a TS queue or TD queue that contains the command language input to be processed. Lowercase characters are changed to uppercase characters, so the command language syntax is free-form. The default is **SYSIN**.

CMDTYPE

The file type of the command file. Valid values are **MQ**, **TD**, **TM**, and **TS**. The default is **TS**.

DELIMITER

This command value delimiter can be used in instead of the left and right parentheses to enclose values in the WebSphere Data Interchange Utility command language. You must supply this value if a command includes the application control field keyword (ACFIELD) with a value of a left or right parenthesis.

PRTNAME

The name of the print file. For a description of this file, see "Print file (PRTFILE)" on page 199. The default is **PRTFILE**.

PRTTYPE

The type of print file. Valid values are **MQ**, **TD**, **TM**, and **TS** (default).

RPTNAME

The name of the report file. For a description of this file, see "Report file (RPTFILE)" on page 200. The default is **RPTFILE**.

RPTTYPE

The type of report file. Valid values are **MQ**, **TD**, **TM**, and **TS** (default).

EXCPNAME

The name of the exception file. For a description of this file, see "Exception file (FFSEXCP)" on page 197. The default is **FFSEXCP**.

EXCPTYPE

The type of exception file. Valid values are **MQ**, **TD**, **TM**, and **TS** (default).

TRAKNAME

The name of the tracking file. For a description of this file, see "Tracking file (FFSTRAK)" on page 198. The default is **FFSTRAK**.

TRAKTYPE

The type of tracking file. Valid values are **MQ**, **TD**, **TM**, and **TS** (default).

QRYNAME

The name of the query file. For a description of this file, see "Query file (EDIQUERY)" on page 201. The default is **EDIQUERY**.

QRYTYPE
The type of query file. Valid values are **MQ**, **TD**, **TM**, and **TS** (default).

APPLID
The ID of the override application that initialized WebSphere Data Interchange. The activity log profile must contain a matching entry to define the log file used for recording errors and events pertaining to the application. The default is **EDIFFS**.

LANGID
The override language ID identifies which language version is being used. Valid value is:
ENU   English (default)

RESPID
The ID of a response program or transaction that WebSphere Data Interchange should invoke after it has finished processing. See "Response applications" on page 381 for more details.

RESPTYP
The type of response name. Valid values are:
PG   A program to be invoked
TX   A transaction to be started

RTERMID
The response terminal ID is used with the keyword `TERMID` on the CICS START command when the value in `RESPID` represents a transaction.

USRFLD
The area used to pass 16 bytes of user data for the specific use by the application. This field is passed to each response application as it gains control. See "Response applications" on page 381 for additional information.

SYSID
The CICS system ID override. For WebSphere Data Interchange use only.

RESPFLAG
The response indicator identifies which action caused the response program to be invoked. Valid values are:

C   The response program was invoked because a continuous receive completed. The `ENVFILE` field contains the name of the TS queue holding the interchange processed during this continuous receive. If the continuous receive was on behalf of a network acknowledgment, the `ENVFILE` field contains the name of the TS queue holding the network acknowledgment.

T   The response program was invoked because of a translation-to-application function. The `APPFILE` field contains the name of a TS queue holding data in the application format. The `HANDLE` field contains the transaction handle associated with the transaction.

U   The response program was invoked because the WebSphere Data Interchange Utility completed.

FILETYP

The type of network acknowledgment file specified in the FILEID field if the PROCESS NETWORK ACKS command was executed. Possible values are **MQ**, **TD**, **TM**, and **TS**.

ECBP

The address of an ECB that WebSphere Data Interchange posts when it finishes processing. This is useful if synchronous processing is desired, but the application must control the syncpoint interval. Your application uses CICS START to start transaction EDIB, and then issues a CICS WAIT EVENT on the ECB. If an ECB is not used, this field must be set to **0**.

CCBRC

The severity code. This field is equivalent to the UTILSEV field. Valid values are:

0   No errors were detected.

4   A low-severity warning was detected, but processing was not impaired.

8   An error was detected that prevented the request from completing successfully.

12   A severe error was detected that prevented the request from completing successfully.

–1   An abend occurred during WebSphere Data Interchange processing.

CCBERC

The WebSphere Data Interchange Utility condition code. This field is equivalent to the UTILCCODE field. For more information, refer to .

FILEID

The envelope file name of the TS queue holding the interchange that was received with the continuous receive facility. Applies only when the response application is invoked at the completion of a continuous receive. The response application must delete this TS queue or process it further. If the continuous receive is set up for network acknowledgments, this field contains the name of the TS queue that contains the network acknowledgment processed. If the TS queue contains a network acknowledgment, WebSphere Data Interchange deletes the TS queue upon return from your continuous receive response application. If the PROCESS NETWORK ACKS command is used, this field contains the name of the TS queue or TD queue containing the network acknowledgment. The FILETYP field contains the associated file type.

APPFILE

The application data TS queue holding the application data in C and D format or in raw data format. Applies only when the response application is invoked because a translation or retranslation request is being processed. This field tells the response application that the TS queue contains one transaction in application format. The response application must delete or process the TS queue further.

ABNDCODE

If an abend occurs during WebSphere Data Interchange processing, this field is set to the CICS abend code, and the `UTILSEV` field is set to **-1**, indicating there is a value in the `ABENDCODE` field. If an abend occurs, WebSphere Data Interchange also issues the CICS DUMP command with an associated dump code of **EDI1**.

THANDLE

The character representation of the Transaction Store handle of the translated transaction. Applies only when the response program is invoked because of a translation-to-application type function. A handle is always given, even if no data is generated from the translation. Data is not generated if the error level exceeds the acceptable value for this transaction. This value is not supplied if the response application is invoked because of continuous receive or WebSphere Data Interchange Utility completion.

FFIEHDR

The address of the receive message long header given by Information Exchange to Expedite/CICS during a continuous receive. This field is only passed when the utility is invoked due to a continuous receive. Refer to the for the format of the Information Exchange receive message header. This address is set only under the following conditions:

- If either the `Translate` or `Deenvelope` value is set to **Y** in the associated continuous receive profile member, the EDI1 TD queue must be defined. If EDI1 is not defined, the address is not set.

- If both the `Translate` and `Deenvelope` values are set to **N**, the `Response Type` value must be set to **PG** in the associated continuous receive profile member. In other words, if only a response application is to be invoked due to a continuous receive and no other WebSphere Data Interchange processing is requested, the application must be a program and not a CICS transaction to receive this field.

FARC

The highest return code encountered during functional acknowledgment processing.

FAERC

The highest extended return code encountered during functional acknowledgment processing.

FABUILT

Indicates whether functional acknowledgments were generated. This field works in tandem with the `FANAME` field.

FFMTFLG

During continuous receives, indicates whether incoming EDI standard data spans more than one TS queue. This field can also be set by a user application to identify multiple envelope queues used with a DEENVELOPE or a DEENVELOPE AND TRANSLATE command. This field works in tandem with the `FFMTCBP` field. See "Processing multiple incoming TS queues" on page 351.

USERSYNC
Indicate to WebSphere Data Interchange that a commit has occurred.This field must be set by response applications when the user application is controlling syncpointing (`SYNCVAL` is **-1**) and the response application issues a syncpoint, a commit, or a rollback.

APMTFLG
During continuous receives, indicates whether incoming application data spans more than one TS queue. This field can also be set by a user application to identify multiple envelope queues used with a DEENVELOPE or a DEENVELOPE AND TRANSLATE command. This field works in tandem with the `APMTCBP` field. See "Processing multiple incoming TS queues" on page 351.

USERCOND
The user condition code pointer. For WebSphere Data Interchange use only.

APMTCBP
This field is set during continuous receive with the address of a multiple TS queue control block, if the incoming application data spans more than one TS queue. A user application can also set this field with the address of a multiple TSQ control block used in association with a DEENVELOPE or a DEENVELOPE AND TRANSLATE command. This field works in tandem with `APMTFLG`. The value in this field must be a valid multiple TSQ control block address if `APMTFLG` is set to `Y`. For more information, see "Processing multiple incoming TS queues" on page 351.

RES1
Reserved for WebSphere Data Interchange.

FFNOCONV
The return code conversion flag. For WebSphere Data Interchange use only.

FANAME
The name of the file where functional acknowledgments are written. This field works in tandem with the `FABUILT` field.

RESPACTV
The response program active flag. For WebSphere Data Interchange use only.

WORKNAME
The name of the work file. For WebSphere Data Interchange use only.

BATFLG
Indicates whether the utility is being invoked from the API.

B Initializes the Syncpoint service and opens the print file for output (rather than extend the batch as in the case of HOT-DI).

blank Invokes EDIFFUT or CICS EDIB.

NOEXCP

Indicates there is no exception file. For WebSphere Data Interchange use only.

INVPARM

Indicates whether an invalid parameter was passed to the network program. For WebSphere Data Interchange use only.

LOGACTV

Indicates that logging CE0050 messages is active. For WebSphere Data Interchange use only.

FFUSADDR

The pointer to this block. For WebSphere Data Interchange use only.

CCBP

The pointer to the CCB. For WebSphere Data Interchange use only.

FFMTCBP

This field is set during continuous receive with the address of a multiple TS queue control block, if the incoming EDI standard data spans more than one TS queue. A user application can also set this field with the address of a multiple TSQ control block used in association with a DEENVELOPE or a DEENVELOPE AND TRANSLATE command. This field works in tandem with FFMTFLG. The value in this field must be a valid multiple TSQ control block address if FFMTFLG is set to Y. For more information, see "Processing multiple incoming TS queues" on page 351.

Continuous receive considerations

The continuous receive Facility is a WebSphere Data Interchange service that works in conjunction with Expedite/CICS and Information Exchange. It also can used with MQSeries trigger queues. By defining the members of a continuous receive profile, you can completely automate the receive process. For details on each field contained within the continuous receive profile, see "Continuous receive profile (CONTRECV-P2 for CICS only)" on page 275.

Using the continuous receive Facility, you can:
- Receive and deenvelope EDI standard data
- Translate the EDI standard data to application format
- Automatically initiate transaction-level response applications that process application data placed in TS queues
- Automatically receive and process network acknowledgments

Because Information Exchange passes the data to the host system immediately after the data enters the mailbox, WebSphere Data Interchange gains control shortly after data is sent from the trading partner's system.

With EDI data and transaction level response programs, your application receives control immediately after WebSphere Data Interchange generates the application data. When using continuous receive for network acknowledgments, the status of EDI transactions in your Transaction Store is kept up to date without initiating the UPDATE STATUS command at different points during the session.

Continuous receive using MQSeries

You can use MQSeries to trigger continuous receive processing. Like sending to an Information Exchange mailbox, you can have your trading partners deliver envelopes to MQSeries queues. To use MQSeries for continuous receives, do the following:

1. Your CICS and MQSeries administrators must enable the CICS region for MQSeries support. As part of this process, they should define an MQSeries trigger event queue, monitored by the MQSeries transaction CKTI in the region. In this example, the name of this queue is **CICS1.TRIGGER**.

2. Using the DEFINE PROCESS command, your MQSeries administrator must define one MQSeries process to provide MQSeries with information about the WebSphere Data Interchange transaction EDIQ. An example of the DEFINE PROCESS command follows:

        DEFINE PROCESS(EDIPROCESS) APPLICID(EDIQ) APPLTYPE(CICS)

3. Using the DEFINE QLOCAL command, your MQSeries administrator must define at least one MQSeries queue to receive envelopes. An example of the DEFINE QLOCAL command follows:

        DEFINE QLOCAL(EDIRECEIVE) INITQ(CICS1.TRIGGER)
            PROCESS(EDIPROCESS) TRIGGER TRIGTYPE(FIRST)
            TRIGDATA('CRPROF=MQ1 MQPROF=RECV1')

    The TRIGDATA field contains two mandatory keyword-value combinations. The first, CRPROF, indicates the name of the WebSphere Data Interchange continuous receive profile member to use when data is received. The second, MQPROF, relates a WebSphere Data Interchange MQSeries profile member to this specific MQSeries queue. The order in which you specify the two keyword-value combinations is not important.

    In this example, only one MQSeries queue is used for event-driven EDI but you can define multiple queues. As long as the WebSphere Data Interchange and MQSeries information is set up correctly, WebSphere Data Interchange will process any number of queues.

4. In WebSphere Data Interchange, you must define an MQSeries queue profile member. In this example, the name of the member is **RECV1**, and the Full Queue Name field is set to **EDIRECEIVE**. Other fields are set accordingly.

5. In WebSphere Data Interchange, you must define a continuous receive profile member. In this example, the name of the member is **MQ1**. Leave the Requestor ID and other Selection fields blank, as they are ignored. Follow the directions in "WebSphere Data Interchange processing after data is received" on page 379 to tell WebSphere Data Interchange what processing is required once data is received from MQSeries.

Continuous receive considerations

When the above steps have been completed successfully, WebSphere Data Interchange automatically processes all data written to the MQSeries queue as soon as MQSeries dispatches the trigger event messages.

Continuous receive selection criteria

You can send and receive data from a single mailbox and to process network acknowledgments through continuous receive. By defining selection criteria, you can set up continuous receive members to receive and process EDI data, and also set up another member to receive and process network acknowledgments, all with the same requestor ID.

Note: When using MQSeries trigger queues, there is no real selection criteria available through the WebSphere Data Interchange continuous receive facility. Rather, the continuous receive profile (CONTRECV) is used to tell WebSphere Data Interchange how to handle data once an MQSeries trigger event has occurred. So, if you are using MQSeries for continuous receive processing, skip to the next section, "WebSphere Data Interchange processing after data is received" on page 379.

You do not need to dedicate mailboxes for each direction. The same mailbox can be used for both sending and receiving. Each active continuous receive profile member is associated with unique continuous receive selection criteria. Setting the `ACTIVE` flag to **Y** in Continuous Receive (CONTRECV) makes the profile member active. Table 108 defines the settings in the profile fields that combine to make the selection criteria unique.

Table 108. Continuous receive selection criteria

Profile field	Description
REQUESTOR ID	The mailbox (requestor) profile member from which you wish to receive data and which contains the Information Exchange account, user ID, and password information. You can have many continuous receive profile members with the same requestor ID, but each member must be unique. You can also have different requestor IDs in each continuous receive profile member. This makes each member unique without having to further qualify the continuous receive.
TP NICKNAME	A trading partner from which you can receive data. If you complete this field, you must also supply the account and user ID fields for the associated trading partner profile member. If you use this field, only incoming data for this specific trading partner is received and processed.
MESSAGE USER CLASS	A code that you and your trading partners agree to use. If you use this field, only incoming data with a matching message user class is received and processed.
NETWORK ACKS ONLY	If you use this field, only network acknowledgments are received and processed. WebSphere Data Interchange issues a continuous receive with the sender account of ***SYSTEM***, a user ID of ***ERRMSG***, and a data type of **A**. This tells Expedite/CICS that only network acknowledgments (and no data) are to be received. The default is **N**.

WebSphere Data Interchange processing after data is received

The previous section describes how to dictate the selection criteria for each continuous receive. When the data has been received, WebSphere Data Interchange must be told what processing to perform. This is controlled by the profile fields listed in order of preference in Table 109.

Table 109. Continuous receive selection criteria

Profile field	Description
NETWORK ACKS ONLY	If the value of this field is **Y**, the data received is always a network acknowledgment and is processed as such. The actual processing is done with a CICS START command, and is issued for transaction EDIB (WebSphere Data Interchange Utility). The PROCESS NETWORK ACKS command is given to the WebSphere Data Interchange Utility along with the acknowledgment. If a response program is supplied in the continuous receive profile, it is given control and the name of the TS queue containing the acknowledgment is passed. Once control is returned to the WebSphere Data Interchange Utility, the TS queue is deleted. A value of **Y** in this field overrides a value of **Y** in the `TRANSLATE` field.
TRANSLATE	If the value of this field is **Y**, the interchange received by Expedite/CICS and given to WebSphere Data Interchange is deenveloped and translated to the application format. This process might also generate functional acknowledgments because they are created during the deenvelope process. Processing is performed in a separate CICS transaction (EDIB). The DEENVELOPE AND TRANSLATE command and the TS queue holding the interchange are passed. A value of **Y** in this field overrides a value of **Y** in the DEENVELOPE ONLY field.
DEENVELOPE ONLY	If the value of this field is **Y**, the interchange received by Expedite/CICS and given to WebSphere Data Interchange are only deenveloped. This process could also generate functional acknowledgments because they are created during the deenvelope process. The processing is performed in a separate transaction, EDIB (the WebSphere Data Interchange Utility). The DEENVELOPE command and the TS queue holding the interchange are passed.

If you leave these fields blank, or set them to **N**, WebSphere Data Interchange invokes only the continuous receive response program. This processing executes in the IMR1 transaction, and a separate EDIB transaction is not initiated.

Effects of defining the EDI1 TD queue

Using the EDI1 TD queue is highly recommended to ensure no data is lost during continuous receive processing, but it is not mandatory. If the TD queue is defined as an intrapartition TD queue, WebSphere Data Interchange uses this queue to pass information from transaction IMR1 to transaction EDIB. When the queue is defined, the IMR1 transaction waits for the EDIB transaction to complete. Having IMR1 wait for EDIB to complete closes a recovery exposure and ensures all interchanges received by Expedite/CICS are processed by WebSphere Data Interchange. If EDI1 is not defined, there ia a short window when the interchange image is not contained in a recoverable resource. In the event of a failure (such as a power outage), interchanges could be lost.

Continuous receive considerations

By defining EDI1 as non-recoverable, you eliminate the possibility of lost interchanges, and Expedite/CICS and WebSphere Data Interchange can recover appropriately. You must define EDI1 as non-recoverable.

The EDI1 TD queue can also be used with WebSphere Data Interchange's non-Expedite/CICS continuous receive interface.

Sent to Network status

When Expedite/CICS Version 3.2 (or higher) is used and the IINCICS network profile communication routine name is VANEXPV4, transaction status is updated from **Send requested** to **Sent to network** once the transactions have successfully been sent by Expedite/CICS. If, for some reason, a transaction is not sent successfully, its status is changed from **Send requested** to **Not sent - network error**. WebSphere Data Interchange and Expedite/CICS typically send interchanges asynchronously. When WebSphere Data Interchange requests a send, Expedite/CICS will acknowledge receipt of the request immediately but might not perform the send at the same time. Once the message has actually been sent to the network, Expedite/CICS (transaction IST1) will LINK back to WebSphere Data Interchange to update the status of the transaction appropriately.

WebSphere Data Interchange and Expedite/CICS can send interchanges synchronously. The Expedite/CICS facility (transaction LG01) is used to indicate whether processing is done synchronously or asynchronously. The VANEXPV4 routine updates transaction status in either case. During the VANEXPV4 status update process:

- WebSphere Data Interchange Transaction Store statuses are updated.
- The unique ID assigned by the network is inserted into the WebSphere Data Interchange database and used as an index for processing subsequent network acknowledgments. This will improve performance for the network acknowledgment processing.

Note: WebSphere Data Interchange supports compress and delivery priority for Expedite/CICS.

Using continuous receive outside Expedite/CICS

Thus far, the continuous receive facility has been described as it works with Expedite/CICS and Information Exchange only. You can also take advantage of the continuous receive profile with your own communication routines by using a WebSphere Data Interchange API to develop applications that interact with WebSphere Data Interchange in a continuous receive mode. See "Continuous receive interface (CICS only)" on page 617 for details.

You can process multiple incoming TS queues when continuous receive is used without Expedite/CICS. For more information, see "Processing multiple incoming TS queues" on page 351.

Response applications

Response applications are user-written CICS programs or transactions that WebSphere Data Interchange invokes at certain points of processing. If you are designing your application to run asynchronously with the WebSphere Data Interchange Utility, or you are using the continuous receive facility, response applications are an essential element to the entire processing structure.

Invoking your application

You can use the following methods to invoke your application:

- Specify a response type of **PG** to cause WebSphere Data Interchange to CICS LINK to your program. When the CICS LINK command is issued, the COMMAREA and LENGTH keywords are used to pass the results. Your application then obtains the COMMAREA address and processes the results.

  Note: Response programs of type **PG** should not handle abends. WebSphere Data Interchange must handle all abends in order to release ENQs and other resources. If WebSphere Data Interchange is not allowed to handle abends, subsequent user tasks may hang.

- Specify a response type of **TX** to cause WebSphere Data Interchange to CICS START your transaction. When the CICS START command is issued, the `FROM` and `LENGTH` keywords are used to pass results. Your application then obtains the results by issuing a CICS RETRIEVE.

See "WebSphere Data Interchange Utility control information" on page 368 for the format of the results control information given to your application.

Types of response applications

WebSphere Data Interchange can invoke three types of response applications:
- WebSphere Data Interchange Utility (response indicator **U)**
- Continuous receive (response indicator **C)**
- Transaction level (response indicator **T)**

For more information on the response indicator field, see "RESPFLAG" on page 372.

It is important to know at which the point your response application gains control. It is also important to know when to enter the response program name and type during product administration, because certain choices might allow you greater flexibility. Response applications can be programs (`Type` **PG**) or CICS transactions (`Type` **TX**).

WebSphere Data Interchange Utility response application (U)

The WebSphere Data Interchange Utility response application is only applicable when your main program invokes the WebSphere Data Interchange Utility. For example, you can CICS START transaction EDIB and pass control information as specified in "WebSphere Data Interchange Utility control information" on page 368. The response name and response type (**PG** or **TX**) passed in this control information are used to identify this response application.

The response application gains control after the WebSphere Data Interchange Utility completes the processing you requested when you started EDIB. This WebSphere Data Interchange Utility response application does not gain control at intermediate points during processing; it only gains control after all requested processing is complete.

Using this response application might not be necessary, depending on how WebSphere Data Interchange was originally invoked. In the previous example, your application started transaction EDIB, and WebSphere Data Interchange processing was asynchronous with your application. It is important that you know the outcome of the function you requested. When you specify a WebSphere Data Interchange Utility response application, the results that WebSphere Data Interchange passes to it contain essential information that can be used for self-identification and to determine the success or failure of the requested function. See "WebSphere Data Interchange Utility control information" on page 368 for the format of the control information passed to the response application.

WebSphere Data Interchange sets the value of the response indicator (RESPFLAG) field to indicate the kind of response application being invoked. This information is also useful in identifying why your application is being invoked. The user field (USRFLD) can also be used to identify the application that originally invoked WebSphere Data Interchange. The user field contains the same value that was supplied when the WebSphere Data Interchange Utility was initiated, unless it has been modified by translation response applications. For more information, see "Transaction response application (T)" on page 384.

The WebSphere Data Interchange Utility response application responds to and inspects the severity code (CCBRC) and condition code (CCBERC) fields to determine success or failure. These fields reflect the highest value encountered by WebSphere Data Interchange during the entire process. Based on the outcome, your WebSphere Data Interchange Utility response application might execute another internal program, or might execute WebSphere Data Interchange to perform subsequent functions.

A WebSphere Data Interchange Utility response application can also manage your resources. The response application can complete the unit of work and clear the input file if the processing was successful. If the application you used to invoke the WebSphere Data Interchange Utility performs a CICS START EDIB and tells the WebSphere Data Interchange Utility not to issue a CICS SYNCPOINT (SYNCVAL of **-1**), the response application can update your recoverable resources and issue the CICS SYNCPOINT. This process includes all WebSphere Data Interchange Utility and response application updates in the same unit of work, as long as your response application is a program. This scenario removes the WebSphere Data Interchange Utility from your initiating application's unit of work, but the response application you develop is still in charge of the WebSphere Data Interchange Utility unit of work.

Continuous receive response application (C)

EDI interchanges and network acknowledgments are dynamically received based on the receive criteria on behalf of a specific continuous receive profile member. The name and type (**PG** or **TX**) of your continuous receive response application is specified in the continuous receive profile. For example, for EDI data, your continuous receive response program can be associated with specific interchanges that meet the criteria. Your

continuous receive response application gains control after WebSphere Data Interchange has completely processed the entire received envelope TS queue, and WebSphere Data Interchange passes the entire envelope TS queue. Normally, the received envelope TS queue contains only one interchange. The response application receives the resulting information from the WebSphere Data Interchange Utility control information. For a description of the resulting information, see "WebSphere Data Interchange Utility control information" on page 368.

The response indicator (`RESPFLAG`) and user area (`USRFLD`) fields are useful in identifying why your application is being invoked. WebSphere Data Interchange sets the value of the `Response indicator` field to indicate the kind of response application being invoked. The `User area` field contains the value specified in the `USER` field of the continuous receive profile member, unless it has been modified by translation response applications. For more information, see "Transaction response application (T)" on page 384.

The envelope TS queue name is passed in the control area. The application must further process the envelope TS queue (delete the TS queue, archive, and so on). You can inspect the severity code (`CCBRC`) and condition code (`CCBERC`) fields to determine the success or failure of the overall continuous receive process. If a transaction level response program is used, it may not be necessary to take action in this continuous receive response application for data element or segment-level translation errors. The errors can be handled in the transaction response application. The continuous receive response application must detect higher severity errors, such as communications errors, database errors, abends, and so on.

For network acknowledgments processing (`Process Network Acks`=**Y** in the continuous receive profile), your continuous receive response application gains control after WebSphere Data Interchange has processed the network acknowledgment and updated the status in the Transaction Store. The name of the TS queue containing the acknowledgment is in the `FILEID` field. A continuous receive response application is optional when receiving and processing network acknowledgments. The only reason to use a continuous receive response application is to save the network acknowledgment itself. The TS queue holding the network acknowledgment is deleted before termination. If you have no reason to save the network acknowledgment, you do not need to develop a continuous receive response application.

If the continuous receive response application is a program (`type`=**PG**), this is a good place to issue a CICS SYNCPOINT. Consider this option, especially if you set the `Allow syncpoints` field in the continuous receive profile to **N**. By doing this, your continuous receive response program gains control and the unit of work is still active. At this point, you can change your recoverable resources and issue the CICS SYNCPOINT command. The unit of work would then include the updates made to the WebSphere Data Interchange Utility and the changes made by your application to recoverable resources.

If you want the syncpoints issued more frequently, especially where you need control after each transaction is processed, see "Transaction response application (T)" on page 384

384 for more information. You can use transaction level response programs in conjunction with continuous receive processing, assuming the `Translate` value is **Y** in the continuous receive profile.

Transaction response application (T)

You can specify that control should pass to the transaction response application for each individual transaction that is translated to the application format. Enter the name of the response program in the `Application file name` field, and enter the type (**PG** or **TX**) in the `Application file type` field (on the Trading Partner Usage Override For Receiving panel, or on the Add Data Format, Copy Data Format, and Update Data Format panels. Details about where to specify the transaction response application are discussed later in this section.

Using this kind of response application is important if you are implementing an event-driven EDI system. Control is passed to the response application when one of the following occurs:

- The `Translate` field is set to **Y** in the continuous receive profile member, and a translation to the application format has occurred.
- One of the following WebSphere Data Interchange Utility PEFORM commands is invoked:
  - RECEIVE AND TRANSLATE
  - DEENVELOPE AND TRANSLATE
  - TRANSLATE TO APPLICATION
  - RETRANSLATE TO APPLICATION
- After the TRANSLATE RECEIVED TRANSACTIONS or RETRANSLATE RECEIVED TRANSACTIONS options are invoked from the online CICS Transaction Store facility.

The point at which transaction level response programs are invoked is based on the recovery level (WebSphere Data Interchange Utility parameter `RECOVERY`). If the recovery level is **E** (envelope), all transactions in the interchange are deenveloped and translated before the first response application is invoked. Once the entire interchange is deenveloped and translated, all corresponding response programs are invoked, one after the other, for each transaction.

A syncpoint is taken by WebSphere Data Interchange after all response programs are invoked. The entire interchange and response program invocation is considered one unit of work. This is the default in CICS and occurs during continuous receive processing. It is also the default when deenveloping is part of the translation process.

Processing is different when transactions are translated separately from the deenvelope process. In this case, the following processes are performed:
- A transaction is translated.
- The response program is invoked.
- A syncpoint is taken.

These steps are repeated for each transaction being translated.

The results are passed to the response application within the WebSphere Data Interchange Utility control information. The name of a unique TS queue that holds the

translated application data and assigned by WebSphere Data Interchange is passed within this control information. This is the name of the application data TS queue (APPFILE). The name of the TS queue is eight characters long and begins with **EDI**. For more information, see "WebSphere Data Interchange Utility control information" on page 368.

The response indicator (RESPFLAG) and user area (USRFLD) fields are useful in identifying why your application is being invoked. WebSphere Data Interchange sets the value of the response indicator field to specify the kind of response application being invoked. When the transaction response application is invoked, the user area will contain:

- The value specified in the User field of the continuous receive profile member if translation is occurring because of a continuous receive request.
- The original user area value provided by the application that originally invoked the WebSphere Data Interchange Utility.
- Blanks if the WebSphere Data Interchange Utility is being invoked because a TRANSLATE RECEIVED TRANSACTIONS or RETRANSLATE RECEIVED TRANSACTIONS option was requested from the online CICS Transaction Store facility.
- The user area modified by a previous invocation of a translation response application. If a translation response application modifies the user area, the next invocation of a response application will receive the updated user area value, including utility response applications and continuous receive response applications.

The modified value of the user area is reset whenever the WebSphere Data Interchange Utility is invoked, and for each EDI interchange processed by a continuous receive request.

The updated control information also contains the severity code (CCBRC) and WebSphere Data Interchange Utility condition code (CCBERC) fields. You can inspect these fields to determine the success or failure of the translation. For more information, refer to . Your transaction response application gains control any time the transaction is translated, independent of the acceptable error level defined in the receive usage/rule. However, if the error is unacceptable, the APPFILE field is filled in with a TS queue name, but the queue is empty. If you attempt to issue a CICS READQ command against this TS queue, you will receive a CICS QIDERR response. WebSphere Data Interchange passes the transaction handle in the results.

If an unacceptable translation error occurs, this response application can specifically identify the transaction in error. This can be saved for the next error-handling processes. This handle is also passed by WebSphere Data Interchange if no errors occur, and can be used as you choose.

Note: Transaction level response applications should not issue the CICS SYNCPOINT command if the WebSphere Data Interchange Utility is running with the SYNCVAL set to **-1**, because this increases the potential for deadlock.

Specifying the transaction response application

The name of the transaction response program is entered in the `Application file name` field, and its type (**PG** or **TX**) is entered in the `Application file type` field on the Trading Partner Usage Override for Receiving panel or the Add Data Format, Copy Data Format, and Update Data Format panels.

The `Application file type` field serves one of two purposes. If the `Application file type` is **MQ**, **TD**, **TM**, or **TS**, then no transaction level response program is invoked. Translation still takes place for each transaction, but rather than passing control to a response program, the translated data is appended to the application file. The transaction response application is only invoked for a transaction when the `Application file type` identifies a program or transaction (**PG** or **TX**).

A transaction response program is assumed if the `Application file type` is **PG** or **TX**. It is important to know whether the file name and type fields are taken from the trading partner receive usage/rule or the data format. WebSphere Data Interchange uses the trading partner receive usage/rule overrides first, if they are present, and then uses the data format fields. Specifying a transaction response program in the trading partner receive usage/rule allows you to identify different programs based on the trading partner that sent them. If there is no need for this level of control, the data format is sufficient to identify a transaction response application based on the application that processes the data.

Whether your response program is invoked depends on the translator extended return code returned. If the extended return code is:

2 or less
> If the translation is acceptable, the Application file (`APPFILE`) field contains the name of the TS queue containing the translated data and the `THANDLE` field is filled in. Even if the translation is unacceptable, the `THANDLE` field data is valid.

3
> - If a usage/rule is found for this transaction, the THANDLE field is filled in.
> - If a usage/rule is not found for this transaction, your transaction level response program is not invoked.

Greater than 3
> Your transaction level response program is not invoked.

If your transaction level response program is not invoked, error handling for this level of error must be managed in a higher-level response program, such as the continuous receive or the WebSphere Data Interchange Utility termination response program, whichever is applicable.

Persistent environment

The WebSphere Data Interchange persistent environment is an optional feature that can be used with CICS/ESA systems to improve WebSphere Data Interchange performance. It accomplishes this by reducing the amount of read operations needed to perform translation and other functions. As data is read from the WebSphere Data

Interchange database, some of it is saved in an z/OS data space. The next time the data is needed, it is obtained from the data space instead of the database. The data space remains functional throughout a WebSphere Data Interchange session. A WebSphere Data Interchange session starts when the first WebSphere Data Interchange function is requested in a CICS region and ends when WebSphere Data Interchange transaction EDIT is terminated.

This process allows many WebSphere Data Interchange transactions to run concurrently and over a long period of time, obtaining the same piece of data repeatedly with only one access to the WebSphere Data Interchange database.

The data saved into the data space includes control strings, receive usages/rules, send usages/rules, and maps.

Running multiple z/OS subtasks

Although there is only one WebSphere Data Interchange data space per CICS region, WebSphere Data Interchange will start up to sixteen z/OS subtasks to manage the WebSphere Data Interchange data space. The more subtasks running, the greater the throughput the data space can deliver. The value used to determine how many subtasks will be started is obtained from the CICS Performance (SYSPROF) profile for the CICS region. You can adjust the number of subtasks up or down to fit your requirements. The number of subtasks defined in the CICS Performance (SYSPROF) profile should not exceed the number of WebSphere Data Interchange threads that can execute concurrently on the CICS region.

Sizing the z/OS data space

The size of the z/OS data space is determined by a couple of factors. First is the maximum size specified in the CICS Performance (SYSPROF) profile for the CICS region. The data space will not be made larger than the value specified in the SYSPROF profile. The value in the profile cannot exceed the installation maximum. Unless altered by the installation, this value will be the network default of 239 4000-byte blocks (less than 1 MB total). Your installation can use the installation exit IEFUSI to change the IBM default.

Secondly, WebSphere Data Interchange learns based on past usage and can define a data space smaller than the maximum size specified. The format of the data space is adjusted automatically, based on past usage to provide the most efficient data space structure. An effort is made to structure the data space at initialization so that no restructuring will be needed during regular WebSphere Data Interchange processing.

However, depending on installation usage and maximum size of the data space, all or part of the data space might become full during processing. When this occurs, WebSphere Data Interchange expands the data space (if defined smaller than the maximum), and/or reorganizes the structure of the data space. If necessary, older data in the data space is removed. Ideally, you do not want any of these situations to occur since each adjustments momentarily delays WebSphere Data Interchange processing. WebSphere Data Interchange will recognize these conditions and try to adjust for them in the next session. If these conditions persist after a few sessions, it might be desirable to increase the maximum data space size. You can monitor the WebSphere Data

Persistent environment

Interchange log file for messages indicating that these scenarios have occurred. All message identifiers relating to the persistent environment begin with the qualifier **GB**.

Make sure that WebSphere Data Interchange sessions are ended normally when using the persistent environment. A WebSphere Data Interchange session is considered ended when transaction EDIT is removed from the system. For information on removing transaction EDIT from the system, see transaction 401.

Enabling and disabling the persistent environment

To enable the persistent environment, you must be running WebSphere Data Interchange in a CICS/ESA region and the CICS Performance (SYSPROF) profile member for the region must indicate that you want the persistent environment established.

To disable the persistent environment, use the CICS Performance profile to indicate that you want the persistent environment disabled.

For more information about the SYSPROF profile member, see "CICS performance profile (SYSPROF-P2 for CICS only)" on page 276.

Using multiple regions

Certain installations might allocate a WebSphere Data Interchange database for use by a single WebSphere Data Interchange region. In these cases, when online updates are made to database records that are kept in the WebSphere Data Interchange Persistent Environment, the old copy of the record is removed from the data space.

Transaction EDIG addresses installations that have multiple WebSphere Data Interchange regions/environments sharing a single WebSphere Data Interchange database. Transaction EDIG starts automatically at set intervals to check if an update has been made recently that might affect the persistent environment. If an update has been made, then all records that match the updated record are removed from the persistent environment; that is, if a control string is deleted, then all matching control strings are removed from the persistent environment.

Note: In installations where multiple regions/environments share a single WebSphere Data Interchange database, all online updates to the database are made from the same region/environment.

Reserved TS and TD queues

By convention, WebSphere Data Interchange uses the three-character prefix EDI for its TS and TD queues. Make sure that no other application running in the same CICS region with WebSphere Data Interchange uses this TS and TD queue naming convention.

TS queues that might require additional processing

The first time WebSphere Data Interchange initializes in a CICS region, it creates a TS queue named EDITV00. This TS queue holds frequently used translation and validation tables.

If you modify any of the following tables using the WebSphere Data Interchange administrative functions, you must purge EDITV00 before the change can take effect. The tables stored in EDITV00 are:

- Alphanum
- Charset
- Filename
- Foldchar
- Langprof
- Monocase
- Prgname
- Specnum
- Tfstatus
- Tptsnrc
- Transyn

For every EDI envelope standard defined to WebSphere Data Interchange, there is a corresponding TS queue. For more information on EDI envelope standards, refer to the WebSphere Data Interchange User's Guide. The following TS queues are used for envelope standards:

EDIFACT

	EDI
ICS	EDII
UCS	EDIU
UN/TDI	EDIT
X12	EDIX

Each EDI envelope standard TS queue is created when the envelope standard is first used. If you modify an EDI envelope standard, you must delete the corresponding TS queue for the changes to become effective.

The next envelope request reads the changes into the TS queue. WebSphere Data Interchange uses the revised EDI standard until CICS system shutdown or until you repeat this process. This improves performance by cutting out the repeated reading of the EDI envelope standards from DB2.

Queues used by export and import

Currently, WebSphere Data Interchange supports up to three TS queues of data per type on import. This is approximately 96000 records (or 32000 by 96000 bytes of data per type). Data to be imported must come into WebSphere Data Interchange in TD queues. These TD queues can be defined as intra- or extra-partitioned. The WebSphere Data Interchange import facility then copies the data from the TD queues into their corresponding TS queues. It is the data in the TS queues that is actually imported.

TS queues used for export and import

These TD queue names are currently reserved by WebSphere Data Interchange for use with the export and import functions:

EDIA   Data formats
EDIB   Tables
EDIC   Control strings
EDIP   Profiles
EDIS   EDI standards
EDIT   Maps

For more information on export and import, refer to "Export/Import utility function" on page 208.

TS queues used by import only

These TS queue names are currently reserved by WebSphere Data Interchange for use with the import function.

EDIATSx
        Data formats
EDIBTSx
        Tables
EDICTSx
        Control strings
EDIPTSx
        Profiles
EDISTSx
        EDI standards
EDITTSx
        Maps

TD queues EDI2 and EDI3

WebSphere Data Interchange reserves TD queues with the names EDI1, EDI2, and EDI3.

For a description of the use of EDI1, see "Effects of defining the EDI1 TD queue" on page 379.

EDI2

If TD queue EDI2 is defined as an intrapartition TD queue, WebSphere Data Interchange will use this queue as the Expedite/CICS administrative response file. Expedite/CICS writes network acknowledgments to this file, and the file is used as input by WebSphere Data Interchange for updating network status.

If EDI2 is not defined, a unique TS queue is used as the administrative response file for each Information Exchange mailbox. These TS queues are always appended to and never cleared.

If EDI2 is defined, both Expedite/CICS and WebSphere Data Interchange use it as the administrative response file for all Information Exchange mailboxes. The destructive read property of intrapartition TD queues causes the network acknowledgments to be

deleted once they are processed by WebSphere Data Interchange. If network acknowledgments do not have to be archived, EDI2 should be used so that network acknowledgments are not reprocessed. Network acknowledgments are reprocessed when EDI2 is not defined and multiple update status requests are executed in an active CICS region.

Note: If network acknowledgments are being received and processed continuously through continuous receive, EDI2 is not used even if it is defined. In this case, Expedite/CICS passes WebSphere Data Interchange one network acknowledgment at a time in unique TS queues. On completion, WebSphere Data Interchange deletes the TS queues holding the network acknowledgments.

EDI3

In a DB2 environment, TD queue EDI3 must be defined as an intrapartition queue with trigger level one and associated with EDIE. Transaction EDIE (program EDIELAS) is responsible for all DB2 event log insertions, and is used to keep this activity out of the main commit scope. Repository module EDIELOG writes event log messages to EDI3, and EDIELAS picks them up from there and inserts them into the database. If an error occurs in EDIELAS, a message is written to the system console with message ID RS0002.

Interface between WebSphere Data Interchange, Expedite/CICS, and Information Exchange

WebSphere Data Interchange provides communication routines that interface to Expedite/CICS. EDI interchanges can be sent to and received from the Information Exchange network using this interface. Additional features include an event-driven continuous receive process and network acknowledgment reconciliation. Trading partner and mailbox (requestor) profiles should reference network profile IINCICS for this environment.

The communication flow involves these components:
• WebSphere Data Interchange
• Expedite/CICS
• Information Exchange

All three components maintain control information to track activities, so it is essential to keep them synchronized. Expedite/CICS and Information Exchange are standalone products that can be operated independently from WebSphere Data Interchange. Therefore, you can execute operational commands without the participation of WebSphere Data Interchange, preventing WebSphere Data Interchange from updating control information. Use these commands sparingly to prevent out-of-sync conditions. The two fundamental programming layers (or sessions) between the three components that must be kept in sync are: the low level Information Exchange session through which all commands and data pass, and the higher level continuous receive session which controls the environment and connectivity.

Information Exchange sessions

The Information Exchange session is the fundamental session started between an application program and a particular Information Exchange user ID (mailbox). All commands and data sent out from, or received into, a mailbox must pass through this session. You can have only one active Information Exchange session per mailbox. For example, if you are using Expedite Base/MVS, this means that you cannot have concurrent Information Exchange sessions for the same mailbox on two different Expedite/CICS regions, or one on CICS and another in batch. You can have any number of Information Exchange sessions from various environments if each session pertains to a different mailbox.

Most of the processing that occurs during the Information Exchange session is performed by Expedite/CICS and Information Exchange. WebSphere Data Interchange is involved only at the application level, but it is still important that the session be initiated and maintained by WebSphere Data Interchange to ensure proper execution of all WebSphere Data Interchange network facilities. The first time WebSphere Data Interchange receives a request from you to perform any network function for a given mailbox, it automatically issues the appropriate session start command to Expedite/CICS. Expedite/CICS then starts the session with Information Exchange and an access key is assigned. All three components now have a record of this particular Information Exchange session. After the Information Exchange session is started, WebSphere Data Interchange performs the network function you requested and returns control to you.

For efficiency reasons, WebSphere Data Interchange does not issue a session end after the requested network function is complete. This means that the next time you request a network function for the same mailbox, WebSphere Data Interchange does not issue another session start because it knows that an Information Exchange session is already active. There is no degradation in performance as a result of leaving an Information Exchange session active.

Expedite/CICS provides the necessary post-initialization PLT program, EXPOSTRT, that automatically reenables any Information Exchange sessions that were active when the CICS region was brought down, even after an immediate shutdown. For more information on PLT processing, see "Program list table considerations" on page 399.

In theory, one Information Exchange session could be used indefinitely. However, remember that you can only have one active Information Exchange session for a given mailbox, and until this Information Exchange session is ended, no other environment, not even a batch job using Expedite Base/MVS, should access this mailbox. If you have multi-environment communication needs with a particular Information Exchange mailbox, you must end the Information Exchange session from CICS before executing communications in batch. WebSphere Data Interchange will only request an Information Exchange session end when your application issues a CLOSE MAILBOX request. To keep WebSphere Data Interchange, Expedite and Information Exchange synchronized, use the WebSphere Data Interchange CLOSE MAILBOX command to end your Information Exchange sessions. If you are using the continuous receive function of WebSphere Data Interchange, executing a CLOSE MAILBOX request will also end a

continuous receive session. It is good practice to first end the continuous receive session and then issue the CLOSE MAILBOX request. See "Continuous receive sessions" on page 395 for details.

Attention: Expedite/CICS provides a facility to start and end Information Exchange sessions directly, but you should not use Expedite/CICS facilities to manage Information Exchange sessions used by the WebSphere Data Interchange application.

The following is for your information and to prevent inadvertent execution of certain Expedite/CICS functions. The Expedite/CICS terminal transaction LG01 can be used to start and end Information Exchange sessions. After you enter your mailbox account and user ID, LG01 checks internal control information. If an Information Exchange session is already active for the mailbox you identified, and the `Force User to log on` option is set to **No**, the main menu displays and Expedite/CICS does not issue a Session Start to Information Exchange. If there is no active Information Exchange session, LG01 prompts you for an Information Exchange password, and if entered, Expedite issues a Session Start to Information Exchange. Once the session starts, the main menu is displayed.

Accessing the mailbox from LG01 is one way to determine if Expedite acknowledges that there is an active Information Exchange session. You can also determine if Expedite is acknowledging an active Information Exchange session by attempting to end an Information Exchange session from LG01 (type **X** to logoff). A confirmation panel is displayed that shows any active network functions that must be completed before the Session End is issued. If one of the active functions is Continuous Receive, it ends the session, causing an out-of-sync condition between Expedite/CICS and WebSphere Data Interchange. If you confirm the request to end the session, Expedite issues a Session End to Information Exchange. (Expedite is unaware that WebSphere Data Interchange originally started the session.) For more information, see "Continuous receive sessions" on page 395.

When Expedite/CICS ends the Information Exchange session, WebSphere Data Interchange is prevented from updating its internal control information, which brings about an out-of-sync condition among the components. However, WebSphere Data Interchange has built-in logic to re-issue a Session Start if Expedite returns the message HI421: SESSION PROFILE DOES NOT EXIST. Therefore, ending an Information Exchange session only has serious implications when a continuous receive session is ended.

If you find that you inadvertently logged off and no Continuous Receives were active, do not log back on to LG01 to restart the session, because WebSphere Data Interchange must initiate Information Exchange sessions. WebSphere Data Interchange will automatically restart a session when you perform your next network function.

As a general rule, leave the session settings on LG01 alone. If you are prompted to enter your password upon entry, log off when you leave. If you were not prompted for a password, do not log off when you leave (use PF3 or type End). For more information on using LG01, refer to the Expedite/CICS Display Application User's Guide.

Interfaces with WebSphere Data Interchange

You can sign on to Information Exchange directly through a terminal connection to the Network. IE /SERV offers many useful tools and functions for managing your network activity. For example, you can directly reset a mailbox's Information Exchange session. However, you should not regularly use the IE/SERV facility to reset Information Exchange sessions used by the WebSphere Data Interchange application. It might be useful in recovery situations, but should be used carefully. Resetting an Information Exchange session from IE/SERV ends the session from the Information Exchange perspective only. Information Exchange is unaware that the session was originally started by WebSphere Data Interchange through Expedite/CICS. It causes an out-of-sync condition between the components. For more information regarding Information Exchange Administrative Services, refer to .

Information Exchange session cleanup

An Information Exchange session problem might prohibit starting or ending an Information Exchange session. The most common symptom of an Information Exchange session error is the Expedite/CICS error SDIERR RESPONSE CODE 00008. If the error is encountered by WebSphere Data Interchange, it is embedded in a logged WebSphere Data Interchange error message.

Information Exchange session problems are most often caused by a user or another program having reset the Information Exchange session. As discussed earlier, you can end an Information Exchange session directly using IE/SERV. Possibly another CICS region or another application in a different environment started a session with the same mailbox. In any event, manual intervention is required. A recovery procedure follows:

1. Sign on to IE/SERV. Choose Option 1, Profiles. Fill in the appropriate account and user ID for the mailbox in question, and choose Option 7, Reset a user's session. Does the message: USER ACCT.USER ID DOES NOT HAVE AN ACTIVE SESSION appear at the bottom of the screen?

    a. If this message appears, your Information Exchange session was previously reset by another person or application. When the application reset your session, it ended its own session. Cancel the reset request and go to Step 2.

    b. If this message does not appear, no Information Exchange session is active for your mailbox. This might mean that whatever reset your session did not end its own session. Enter **Y** to confirm the reset, go to Step 2.

    Note: If you find you must perform this procedure frequently, you might want to inspect the session trace available on IE/SERV to identify the LU name that is resetting your session.

2. Use the Expedite/CICS IDLT CICS terminal transaction to delete the Information Exchange session control information from the Expedite/CICS control file. The format of the IDLT transaction follows:

    ```
    IDLTacct user ID
    ```

    The account number (*acct*) must be eight characters long, left-justified and concatenated to the IDLT transaction ID. If your account is shorter than eight characters, pad the remaining positions with spaces.

    No validation is performed on your user ID, so it is important to enter your user ID correctly.

3. If your Information Exchange session had any active continuous receive sessions, try the failed network function again or go to the next section, Continuous receive session. As discussed earlier, WebSphere Data Interchange will automatically reissue an Information Exchange Session Start.

Continuous receive sessions

The continuous receive (CR) session is a high-level application programming layer that is added on top of an Information Exchange session. A continuous receive session cannot run without an Information Exchange session. An overview of the continuous receive process follows.

When a CR session is active, EDI messages entering the mailbox that match the continuous receive criteria are automatically received. Information Exchange automatically starts Expedite/CICS and passes the EDI envelope. Expedite/CICS then writes the envelope to a unique TS queue, linking to WebSphere Data Interchange to pass the name of this TS queue. WebSphere Data Interchange processes the envelope using the processing options specified in the applicable continuous receive profile, after which WebSphere Data Interchange returns control to Expedite/CICS. Expedite/CICS is then free to process the next envelope for this continuous receive session.

Exactly when WebSphere Data Interchange returns control to Expedite/CICS depends on whether the TD queue EDI1 is defined as intrapartitioned. If EDI1 is defined, WebSphere Data Interchange and Expedite run synchronously. In other words, Expedite/CICS will not process the next envelope until WebSphere Data Interchange returns control. If the EDI1 TD queue is not defined, Expedite/CICS and WebSphere Data Interchange run asynchronously where Expedite is free to start processing the next envelope before WebSphere Data Interchange has finished with the first. For more information, see "Effects of defining the EDI1 TD queue" on page 379. Multiple continuous receive sessions with unique receive criteria can be active for a single mailbox, in which case all sessions use the same Information Exchange session. You can also run continuous receive sessions for other mailboxes.

Starting and stopping continuous receive sessions

The continuous receive facility of WebSphere Data Interchange provides two CICS terminal transactions: EDIR to start a continuous receive session, and EDIS to stop a continuous receive session. These two CICS transactions are complemented by the WebSphere Data Interchange Utility commands START CONTINUOUS RECEIVE and STOP CONTINUOUS RECEIVE. Also, the WebSphere Data Interchange Utility provides the REPORT CONTINUOUS RECEIVE STATUS command for reporting the statuses of your continuous receives. For more information on these commands, see "Continuous receive" on page 13.

Note: This information applies only to continuous receive processing associated with Expedite/CICS and Information Exchange. Continuous receive processing driven by user-written applications is not started or stopped in this manner. For detailed information, see "Continuous receive interface (CICS only)" on page 617.

Interfaces with WebSphere Data Interchange

Both EDIR and EDIS can be issued against all continuous receive profile members or selectively to a single member. The following shows the format for entering the transaction at a terminal:

```
EDIR [membername]
EDIS [membername]
```

where *membername* identifies a member of the continuous receive profile. If a member name is not specified, all continuous receive members are requested, and a continuous receive start or stop occurs for each profile member.

During a continuous receive Session Start using EDIR, WebSphere Data Interchange invokes Expedite/CICS, passing it the continuous receive selection criteria. For criteria options, see "Continuous receive selection criteria" on page 378. Expedite/CICS assigns a key to identify the continuous receive session, passes the criteria and key to Information Exchange, and then returns control to WebSphere Data Interchange. WebSphere Data Interchange sends a message back to the issuing terminal indicating the success or failure of the continuous receive session start. After a successful start, all three components (WebSphere Data Interchange, Expedite/CICS, and Information Exchange) have saved the key assigned to the continuous receive session for subsequent identification. If the start request is unsuccessful, inspect the EDIFFS event log for information about the error.

During a continuous receive Session Stop using EDIS, WebSphere Data Interchange invokes Expedite/CICS passing the key for the particular continuous receive session. Expedite/CICS issues the end continuous receive request to Information Exchange. The continuous receive cannot be stopped at this point. If a receive is taking place. Expedite/CICS returns control to WebSphere Data Interchange where WebSphere Data Interchange and Expedite/CICS can complete asynchronously. However, WebSphere Data Interchange checks Expedite/CICS control information until an indicator shows that the continuous receive session has ended. At that point, WebSphere Data Interchange issues a message confirming a successful end.

Although Expedite/CICS provides a facility to start and stop continuous receive sessions directly, using these Expedite/CICS facilities regularly to manage your WebSphere Data Interchange continuous receive sessions is not recommended.

The following is for your information and to prevent inadvertent execution of certain functions in Expedite/CICS. The Expedite/CICS terminal transaction, LG01, can be used to start and stop any active continuous receive sessions by using Option 1, Work with Receive Data. WebSphere Data Interchange will not be aware of any activities performed in LG01, so use this feature carefully.

Continuous receive session cleanup

Before attempting any cleanup, generate a continuous receive status report. For more information, see "Reporting continuous receive status" on page 13. Based on the reported status, you can initiate a reasonable cleanup.

WebSphere Data Interchange supplies a CICS terminal transaction, EDIZ, that can be used to clean up an unrecoverable continuous receive session. This transaction deletes

the internal continuous receive session control records managed by WebSphere Data Interchange. It does not affect the state of Expedite/CICS or Information Exchange. Therefore, it is essential that Expedite/CICS (using LG01) be used to terminate all continuous receive sessions before executing EDIZ. If Expedite/CICS has not terminated the continuous receive and EDIZ is executed inadvertently, EDI envelopes will be lost because WebSphere Data Interchange is unaware of the receive requests for which Expedite/CICS is now passing data.

The format of the EDIZ command is:

```
EDIZ [membername]
```

if *membername* is not specified, all continuous receive members are requested.

Identifying unrecoverable continuous receive sessions

Table 110 describes some methods to help you to identify an unrecoverable continuous receive session.

Table 110. Identifying unrecoverable continuous receive sessions

Problem	Explanation
A continuous receive session that was once working is no longer receiving data.	Most likely, the Information Exchange session was lost after a continuous receive had already been successfully started.
A continuous receive session is receiving data from the mailbox, but WebSphere Data Interchange is not processing the data.	It appears that WebSphere Data Interchange is never getting control. This is probably caused by an inadvertent or premature use of EDIZ.
A VN1019 timeout error is logged.	When EDIS is executed, WebSphere Data Interchange will check the stop indicator for up to 2 minutes. If more than 2 minutes elapse, a timeout error is logged (VN1019) and a message is issued to the terminal. This might be caused by a delay in Expedite/CICS to end a continuous receive session.
EDIR or EDIS fails, a negative response is returned, and WebSphere Data Interchange logs an error that an SDIERR occurred.	EDIR or EDIS will fail immediately if Expedite/CICS cannot communicate with Information Exchange because of an Information Exchange session problem. WebSphere Data Interchange logs an error that an SDIERR occurred and sends a message to the terminal. This is an indication that you have an Information Exchange session that requires recovery.

Recovering continuous receives

If any of the above problems arise, use the following continuous receive recovery procedure:

1. Sign on to the mailbox using LG01.
   a. If you are not prompted to enter your password, go to Step 2.
   b. If you are prompted to enter your password, no Information Exchange session is active for your ID. Enter your password to access the LG01 main menu.

Expedite/CICS will start a new Information Exchange session, which might reactivate continuous receives that were not receiving data previously. This is expected. Continue with Step 2.

2. Choose Option 1, Work with Receive Data, and then choose Option 3, Stop Continuous Receive. If no entries are displayed, go to Step 6. If entries are displayed, go to Step 3.

3. Issue a stop request by all continuous receive sessions. Use the Enter key to refresh the screen.

   a. If all sessions change status from STARTED to STOPPED, go to Step 6.

   b. If all sessions do not change status from STARTED to STOPPED, go to Step 4.

4. Sign on to IE/SERV. Choose Option 1, Work with Profiles. Fill in the appropriate account and user ID for the mailbox in question and choose Option 8, Reset a User's Session. Enter **Y** to confirm the reset and go to Step 5.

5. Use the Expedite/CICS IDLT CICS terminal transaction to delete the Information Exchange session control information from the Expedite/CICS control file. The format of the IDLT transaction is:

   IDLT*acct* [*user ID*]

   The account number (*acct*) must be 8 characters long, left-justified and concatenated to the IDLT transaction ID. If your account is shorter than 8 characters, pad the remaining positions with spaces.

   No validation is performed on your user ID, so it is important to enter your user ID correctly.

6. You should still be in LG01 at this point. Type **=X** to log off. Confirm the session end, and go to Step 7.

7. Execute the WebSphere Data Interchange Continuous receive cleanup transaction, EDIZ, only for the continuous receive profile member(s) that are experiencing a symptom, and go to Step 8.

8. Restart the continuous receive session using EDIR.

   At this point, the continuous receive session should be recovered. If you executed EDIZ inadvertently or prematurely, envelopes might have been lost. You can use LG01 to determine which, if any, envelopes were not processed by WebSphere Data Interchange.

   Choose the Receive option from the main menu, and then Option 5, View List of Completed Receives. Each EDI envelope received by Expedite/CICS will have an entry. A status of **RECEIVED** is normal. A status of **E-HI999** means that WebSphere Data Interchange did not accept the envelope. WebSphere Data Interchange returns the HI999 error when a continuous receive control record is not found for a receive. Envelopes that encounter the HI999 error cannot be reprocessed through the Release option on this LG01 panel, because WebSphere Data Interchange and Expedite/CICS cannot be re-synchronized. However, you can use the View option to obtain the interchange control number, the interchange sender, and so on. If your mailbox has archiving turned on, you can use IE/SERV to retrieve each lost envelope. An envelope retrieved from archive is receivable just as if it had been sent.

Note: When an HI999 error is returned, WebSphere Data Interchange issues an error message to the EXPLOG1 data set associated with your CICS region: CR0120 WebSphere Data Interchange CONTROL RECORD MISSING FOR RECEIVED DATA, WRITING DATA TO EXPDERR FILE.

Program list table considerations

There are two post-initialization PLT programs that affect continuous receive processing. All customers should use the Expedite/CICS PLT program called EXPOSTRT that re-establishes both Information Exchange and continuous receive sessions automatically during CICS start-up, even after an immediate shutdown. The second program is an optional WebSphere Data Interchange PLT program called EDICRTS that issues the EDIR transaction to start all continuous receive profile members with an `Active` flag set to **Y**. For most customers, using EDICRTS is unnecessary because EXPOSTRT will reestablish continuous receive sessions for you. EDICRTS starts entirely new sessions, which is redundant if your continuous receive sessions were already active when the region was brought down.

WebSphere Data Interchange also provides an optional pre-termination PLT program named EDICRSP that issues an EDIS to stop all continuous receives, and also issues a CLOSE MAILBOX request against all involved mailboxes. During a normal shutdown, EDICRSP ends all continuous receive sessions and all Information Exchange sessions. This releases all mailboxes using continuous receive so that other communication applications requiring the same mailboxes can execute while the CICS region is down. This will prevent the Information Exchange session problems discussed earlier. However, if you have no other Information Exchange communication applications, or if the other applications you run do not use the same mailbox, you do not need to release the Information Exchange session, and PLT program EDICRSP is not needed.

As you can see, the pre-termination program EDICRSP and post-initialization program EDICRTS go hand in hand. If you use EDICRSP for pre-termination, then you should use EDICRTS for post-initialization to start those continuous receive sessions ended by EDICRSP. EXPOSTRT does not start these sessions, because they will not be active after shutdown is complete.

The following are sample post-initialization and pre-termination PLTs.

```
*--------------------------------------------------------------------*
* Sample post-initialization PLT including program EDICRTS,          *
* the continuous receive start program, and the necessary            *
* Expedite/CICS PLT program EXPOSTRT.  Also included is a            *
* sample entry for the DB2 attachment facility, DSNCCOM1, in         *
* case you are using DB2 for your repository.  The order of          *
* the following programs is important.                               *
*--------------------------------------------------------------------*
PLTAA   DFHPLT TYPE=INITIAL,SUFFIX=AA
        DFHPLT TYPE=ENTRY,PROGRAM=DSNCCOM1
        DFHPLT TYPE=ENTRY,PROGRAM=EXPOSTRT
        DFHPLT TYPE=ENTRY,PROGRAM=EDICRTS
        DFHPLT TYPE=FINAL
        END

*--------------------------------------------------------------------*
```

```
* Sample pre-termination PLT including program EDICRSP, the     *
* continuous receive stop program.  Program EDIXSOX, which      *
* terminates the long running WebSphere Data Interchange transaction,
 EDIT, *
* is shown along with DB2 attachment program,  *
DSNCCOM1, to provide the proper order for the entires.  *
*-----------------------------------------------------------------------*
PLTZZ    DFHPLT TYPE=INITIAL,SUFFIX=ZZ
         DFHPLT TYPE=ENTRY,PROGRAM=EDICRSP
         DFHPLT TYPE=ENTRY,PROGRAM=EDIXSOX
         DFHPLT TYPE=ENTRY,PROGRAM=DSNCCOM1
         DFHPLT TYPE=FINAL
         END
```

Note: If you will be using the EDICRSP PLT program, it is necessary
to add an XLT entry for Expedite/CICS transaction ISC2. A sample
entry follows:

```
*-----------------------------------------------------------------------*
*                      WebSphere Data Interchange for CICS          *
* The following XLT (CICS Transaction List Table) entries are to be *
* applied to any CICS region where WebSphere Data Interchange will  *
* execute and where program EDICRSP is included in the pre-termination*
* PLT.                                                              *
*                                                                  *
*-----------------------------------------------------------------------*
XLTAA    DFHXLT TYPE=INITIAL,SUFFIX=AA
         DFHXLT TYPE=ENTRY,TRANSID=ISC2
         DFHXLT TYPE=FINAL
         END
```

Processing program table considerations

In CICS/ESA environments, you should define the following programs in the processing
program table (PPT) with EXECKEY(CICS): EDICRIN, EDICRSP, EDICRTS, and
EDIXSOX.

WebSphere Data Interchange supplied transactions

WebSphere Data Interchange provides a set of CICS transactions to perform many
different functions. The following summary describes each transaction and its
associated function.

Table 111. WebSphere Data Interchange supplied CICS transactions

Transaction	Description
EDIA	The online administrative transaction. Use this transaction to customize in the CICS environment. **Note:** To prevent processing from being halted while EDIX waits to proceed, make sure to establish a DB2 thread pool that is used by only EDIA.

Table 111. WebSphere Data Interchange supplied CICS transactions (continued)

Transaction	Description
EDIB	The WebSphere Data Interchange Utility transaction. If you run the WebSphere Data Interchange Utility asynchronously, use EXEC CICS START to start this transaction. You can also start EDIB with WebSphere Data Interchange as part of continuous receive processing. **Note:** To prevent processing from being halted while EDIX waits to proceed, make sure to establish at least three DB2 threads that is used only by EDIB. The maximum number of threads needed for this transaction is the sum of all TRANClass values for the transactions in EDIB. A typical value is five threads.
EDID	Used internally by WebSphere Data Interchange. A background transaction used to update network status. EDID does not have a user interface.
EDIE	Used internally by WebSphere Data Interchange.A background transaction used to insert DB2 event log entries into the database (see TDQs EDI1, EDI2, and EDI3). EDIE keeps DB2 event log insertions out of the main commit scope. EDIE does not have a user interface. **Note:** To prevent processing from being halted while EDIX waits to proceed, make sure to establish a DB2 thread pool that is used only EDIX and EDIE. Two or more threads must be available for this pool.
EDIG	A transaction automatically invoked periodically when using the persistent environment.
EDIM	A transaction that executes the WebSphere Data Interchange Message Broker to transform data.
EDIQ	The transaction that gains control from MQSeries transaction CKTI when data is received into an MQSeries queue that has trigger processing (continuous receive) associated with it.
EDIR	The online or background transaction used to initiate continuous receive requests.
EDIS	The online or background transaction used to terminate continuous receive requests.
EDIT	A transaction that can be used to terminate the WebSphere Data Interchange environment. The WebSphere Data Interchange environment is usually terminated by placing EDIXSOX in the CICS pre-termination PLT, but you can also terminate the WebSphere Data Interchange environment by running EDIT. Certain pieces of information are maintained from the start of the first WebSphere Data Interchange activity within a CICS region, continuing throughout the session, until the WebSphere Data Interchange environment is terminated. You can use EDIT to reset this information. EDIT shuts down transactions EDIG and EDIX, shuts down the persistent environment, releases CSD storage, and deletes the EDICSDA and EDITV00 TS queues. Using EDIT to perform these functions is not part of standard procedures.
EDIV	The installation verification transaction. For more information on this transaction, refer to the *WebSphere Data Interchange for z/OS Installation Guide*.

WebSphere Data Interchange supplied transactions

Table 111. WebSphere Data Interchange supplied CICS transactions (continued)

Transaction	Description
EDIW	The WebSphere Data Interchange Utility invocation transaction. For more information, see "Using EDIW to invoke the WebSphere Data Interchange Utility" on page 404.
EDIX	A background transaction used to perform some Transaction Store updates for other WebSphere Data Interchange transactions. This transaction does the following acquires a transaction handle and deletes an envelope for a DEENVELOPE command with the `DUPENV`(**Y**) option. You must define EDIX in the CICS Resource Control Table for DB2 installations. EDIX processes a request and remains idle in the system for up to one minute. When the minute expires, and no other request has been issued by a WebSphere Data Interchange transaction, EDIX removes itself from the system. If another request is generated in the one-minute wait period, the request is honored and the one-minute wait is reset. Eventually, as all WebSphere Data Interchange work is quiesced, EDIX removes itself from the system. If need be, EDIX can be removed from the system manually by typing **EDIX**. No parameters are necessary. **Note:** To prevent processing from being halted while EDIX waits to proceed, make sure to establish a DB2 thread pool that is used only EDIX and EDIE. Two or more threads must be available for this pool.
EDIZ	The online or background transaction used to clean up continuous receive problems.

Performance monitor user exit

WebSphere Data Interchange can invoke a user exit during enveloping and deenveloping for performance monitoring. The exit is invoked when a user exit program name is specified in the Application Defaults (APPDEFS) profile. This function is applies only for WebSphere Data Interchange for CICS.

For performance monitoring to be meaningful, it should be done in tandem with Expedite/CICS. Typically, WebSphere Data Interchange and Expedite/CICS work asynchronously with one another (except for continuous receives). To understand resource utilization involved in data translation/transmission, the WebSphere Data Interchange and the Expedite/CICS components must be considered together. This can be achieved by using the performance monitor user exit capabilities of both products. The COMMAREA format passed to each is the same. "Format of performance monitor commarea" on page 403 describes which fields are filled in by WebSphere Data Interchange and which fields are filled in by Expedite/CICS.

On the send side, WebSphere Data Interchange could invoke the user exit during enveloping and, subsequently, Expedite/CICS could invoke it after sending. These two disparate invocations could be associated using the interchange key fields:
* Sender ID
* Receiver ID
* Control number

• Direction

On the receive side, Expedite/CICS could invoke the user exit after receiving the interchange and subsequently WebSphere Data Interchange could invoke it during deenveloping. If more than one envelope is generated or deenveloped during a single execution of WebSphere Data Interchange, the user exit would be invoked for each envelope.

By using these user exit capabilities, calculations could be done on the collective resources consumed by both products in handling an interchange. The user exit would most commonly be used to stamp SMF records (through the EXEC CICS MONITOR command) with the information passed to it. The SMF records could then be analyzed to produce the desired performance statistics. This information would be helpful for system tuning and for capacity planning.

Format of performance monitor commarea

Table 112. Format of the performance monitor commarea

Name	Offset	Length	Type	Owner	Description
RESPCODE	0	5	Char	Exp	Response code
RESPTYPE	5	8	Char	DI/Exp	WebSphere Data Interchange - ENVELOPE or DENVELOPE
DIRECT	13	1	Char	DI/Exp	Send or receive
SENDQUA	14	4	Char	DI	Interchange sender qualifier
SENDID	18	35	Char	DI/Exp	Interchange sender ID
RECVQUA	53	4	Char	DI	Interchange receiver qualifier
RECVID	57	35	Char	DI/Exp	Interchange receiver ID
CNTRLNO	92	14	Char	DI/Exp	Interchange control number
SACCT	106	8	Char	Exp	Sender's Information Exchange account
SUSERID	114	8	Char	Exp	Sender's Information Exchange user ID
ALIASTYP	122	1	Char	Exp	Receiver's alias table type
ALIASID	123	3	Char	Exp	Receiver's alias table ID
RACCT	126	8	Char	Exp	Receiver's Information Exchange account
RUSERID	134	8	Char	Exp	Receiver's Information Exchange user ID
DESTTYPE	142	1	Char	Exp	Receiver's destination type
MSGUCLAS	143	8	Char	DI	Message user class
UNIQUEID	151	8	Char	Exp	DI status update unique ID
DATE	159	8	Char	Exp	Current date (*YYYYMMDD)*
TIME	167	6	Char	Exp	Current time (*HHMMSS*)
APPLID	173	8	Char	Exp	CICS application ID

Performance monitor user exit

Table 112. Format of the performance monitor commarea (continued)

Name	Offset	Length	Type	Owner	Description
TRANID	181	4	Char	Exp	CICS transaction ID
TASKNO	185	7	Char	Exp	CICS task number
ENVSIZE	192	11	Char	DI	Envelope size
NUMTRXS	203	11	Char	DI	Number of transactions in envelope
MSGSIZE	214	11	Char	Exp	Information Exchange message size
APPREF	225	14	Char	DI	Interchange application reference
RESERVED	239	261	Char	N/A	Reserved for WebSphere Data Interchange

Using EDIW to invoke the WebSphere Data Interchange Utility

YOu can use the WebSphere Data Interchange for CICS transaction EDIW to invoke the WebSphere Data Interchange Utility. On entering transaction EDIW, a panel is displayed where you can enter Utility Control Information and Utility PERFORM commands. The fields on the EDIW panel generally relate to the fields in the Utility Control Information structure. For more information, see "WebSphere Data Interchange Utility control information" on page 368.

The EDIW transaction is useful in testing various versions of PERFORM commands until final versions can be established. In this way, your Utility invocation programs do not need to be recompiled or your command files changed to experiment with various PERFORM commands.

To invoke the Utility, fill out the appropriate fields on the panel and press Enter. EDIW allows the Utility to be invoked through an EXEC CICS LINK to program EDIFFUT or through an EXEC CICS START of transaction EDIB. When the EXEC CICS LINK method is selected, the Utility invocation results will be returned and displayed on the panel in the form of the severity and condition codes. When the EXEC CICS START method is selected, the started EDIB task might run either synchronously or asynchronously with EDIW. The actual PERFORM command to be executed by the Utility can be specified directly on the EDIW panel or can be referenced in a command file.

```
WebSphere Data Interchange for CICS  Utility Invocation
   ----------------------------------------------------------------
   Syncpoint Value......:            Utility Response Prog:
   Command File Name....:            Utility Response Type:
   Command File Type....:            Terminal ID..........:
   Command Delimiter....:            Process Net Ack File.:
   Print File Name......:            Process Net Ack Type.:
   Print File Type......:            Multiple TSQ Mode....:
   Report File Name.....:            User Area.........:
   Report File Type.....:
   Exception File Name..:
```

```
   Exception File Type..:
   Tracking File Name...:              Util Severity Code...:
   Tracking File Type...:              Util Condition Code..:
   Query File Name......:              Abend Code...........:
   Query File Type......:              Func Ack Built.......:
   Application ID.......:              Func Ack Ret Code....:
   Language ID..........:              Func Ack Ext Ret Code:
   Command Statements...:
-----------------------------------------------------------------
F3=End  F4=Dlt  F7=Bwd  F8=Fwd
 F9=Clr Link? Y  Wait? N  Caps? Y
_____
```

Using EDIW to invoke WebSphere Data Interchange

Chapter 6. Using WebSphere Data Interchange in the z/OS environment

Using sample JCL

WebSphere Data Interchange Utility (EDIUTIL) JCL

This section describes the WebSphere Data Interchange Utility JCL. Sample Utility JCLs are distributed in the JCL distribution library (EDI.V3R2M0.SEDIINS1). The DB2 version is in member EDIUTILD. The JCL is divided into subsections based primarily on function.

Note: Using the RLSE keyword to allocate certain output files might cause processing problems (such as B37 abends), especially when switching output files to separate processed data. The RLSE keyword releases "unused" space. Specifying it for the following output files might cause the file allocation to be changed to a size that is too small to accommodate the data written to it during repetitive WebSphere Data Interchange processing. If you decide to use the RLSE keyword with these files, make sure to delete and reallocate these files before each execution of WebSphere Data Interchange.

- EDIQUERY
- FFSEXCP
- FFSTRAK
- FFSWORK
- INVOICE
- PURCORD
- QDATA
- REQ1DD
- REQ2DD
- RPTFILE

Section 1 JCL modifications

The beginning of the JCL states that certain elements of the JCL must be modified to run in your environment. This is usual and customary. The minimum region size to execute the WebSphere Data Interchange Utility is 4096 K.

```
//EDIUTIL JOB (INSTALLATION DEPENDENCIES, REGION=4096K)

//*

//*********************************************************************
//*This sample JCL will invoke the WebSphere Data Interchange Utility.*
//*********************************************************************
//*    1. Change the JOB statement as necessary.                     *
//*    2. Revise ddnames and dataset names to meet your requirements.*
//*    3. If necessary, revise STEPLIB to match your libraries.      *
//*    4. If a language other then English is installed, change all  *
//*       "EDIENU." to "EDI###.", where "###" represents the proper  *
//*       language identifier value shown in the program directory.  *
//*********************************************************************
```

Section 2 WebSphere Data Interchange Utility parameters

When invoking the WebSphere Data Interchange Utility, certain parameters are passed in. These parameters are keyword-oriented. The keywords should be separated by at least one blank character. Valid keywords are:

APPLID

The application ID. The default is **EDIFFS**. This keyword also identifies the log file as specified in the Activity Log (ACTLOGS) profile.

LANGID

The language ID. The default is **ENU**. The value supplied must match an entry in the Language (LANGPROF) profile. This parameter is used to control date formats, decimal notations, and other language-specific fields.

SYSID The installation-defined WebSphere Data Interchange system ID that controls access to various components of WebSphere Data Interchange. The default is **DIENU**. The value in this field is part of the resource name defined using RACF or another resource control product.

DML The delimiter used in place of left and right parentheses to enclose WebSphere Data Interchange Utility command values.

PLAN The DB2 plan name. For DB2 installations, this parameter is required if there is no EDITSIN data set counterpart. If this parameter is specified here and in EDITSIN, the value in EDITSIN overrides the value specified here. There is no default.

SYSTEM

The DB2 subsystem ID. For DB2 installations, this parameter is required if there is no EDITSIN data set counterpart. If SYSTEM is specified here and in EDITSIN, the value in EDITSIN overrides the value specified here. There is no default.

In a DB2 environment, the parameters can be specified in multiple places:

- If the EXEC statement specifies PGM=IKJEFT01, the parameters are passed in with the PARM keyword in the RUN statement in the SYSTSIN data set.
- If the EXEC statement specifies PGM=EDIFFUT, the parameters are passed in with the PARM keyword in the EXEC statement. However, the DB2 plan and subsystem ID might come from the EDITSIN data set.

For more information, see "WebSphere Data Interchange and DB2 attachment" on page 428.

In DB2, change the parameter PLAN=EDIENU32 to match your DB2 plan name and change the parameter SYSTEM=DSN to match your DB2 subsystem ID.

```
//************************************************************************
//*    Change the PARMs as follows:                              *
//*    1. If necessary, change the "SYSID=DIENU" parm to match        *
//*       your WDI RACF installation.                       *
//*    2. If necessary, change the "APPLID=EDIFFS" parm to match      *
//*       the desired WDI application ID.                   *
```

```
//*   3. If necessary, change the "LANGID=ENU" parm to the         *
//*      proper language identifier.                               *
//*****************************************************************
//*
//XDIUTIL EXEC PGM=EDIFFUT,PARM='SYSID=DIENU
APPLID=EDIFFS LANGID=ENU'
//*
```

Section 3 STEPLIB requirements

STEPLIB accesses the WebSphere Data Interchange load modules. If you make communication requests, STEPLIB must include the library where the communication routines reside. STEPLIB is not required if you have provided access to the necessary programs in other ways (such as adding the load libraries to the LNKLST, or loading all necessary programs into the link pack area).

In DB2, STEPLIB must also include the DB2 load library.

```
//STEPLIB  DD DSN=EDI.V3R2M0.SEDILMD1,DISP=SHR   <--EDI LOAD LIBRARY

//         DD DSN=SYS1.SNA.LOADLIB,DISP=SHR      <--COMM. LOAD LIBRARY
```

Section 4 PRTFILE

PRTFILE is required for all WebSphere Data Interchange Utility functions. Error messages generated during function processing and a summary report are written to PRTFILE. This data set contains fixed block addressing (FBA) or variable block addressing (VBA) records with a logical record length of 132.

```
//*****************************************************************
//* Specify the Audit print file. In this case the summary report   *
//* generated by the utility will be returned to JES.  You might want *
//* to allocate an external file, and print it for future reference.  *
//*****************************************************************
//*
//PRTFILE  DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=132)
//*
```

Section 5 TRANSLATE TO STANDARD files

The APDATA01 and APDATA02 ddnames are examples of files containing application data that is to be translated into EDI standard format. The files can contain C and D records, or they can contain data in raw data format. The actual ddnames must match the names specified with the APPFILE keyword in the PERFORM command. A file is expected to contain C and D records unless the RAWFMTID keyword is provided. These data sets can contain fixed or variable length records.

```
//*****************************************************************
//*  If translating to standard formats:                          *
//*                                                                *
//*  Allocate all input datasets that will be translated in this step.*
//*  These ddnames should be referenced by the APPFILE keyword in   *
//*  the EDISYSIN input command language statements.               *
//*****************************************************************
//*
```

```
//APDATA01 DD DSN=PHYSICAL.FILE1.NAME,DISP=OLD
//APDATA02 DD DSN=PHYSICAL.FILE2.NAME,DISP=OLD
//*
```

Section 6 Destination files

When translating to EDI standard format, the WebSphere Data Interchange Utility writes
transactions that were not translated successfully to FFSEXCP. When translating to
application format, the Utility writes translated transactions to FFSEXCP if it can not
write them to the intended file (This is usually because the intended file could not be
opened or is full). Optional records are also written to FFSEXCP if the tracking file
(FFSTRAK) does not exist and your application data is in C and D format. FFSEXCP
must be large enough to contain the largest data record you are translating, and the
largest information record the translator might return. Use the JCL DISP option to
control whether FFSEXCP is cleared or appended to during the first use. After the first
use, WebSphere Data Interchange automatically appends to the file. FFSEXCP can
contain fixed or variable length records.

```
//*********************************************************************
//* If translating to standard or to application formats:         *
//*                                                               *
//* Specify the exception file to hold transactions in error      *
//* DCB= Allocated the same as application data files, since      *
//*      this file holds a copy of the untranslated transaction.  *
//*********************************************************************
//*
//FFSEXCP  DD DSN=BAD.TRANSACTION.FILE,DISP=MOD
//*
```

Section 7 FFSTRAK file

FFSTRAK holds the optional information records if they are requested. If you are using
C and D records and if FFSTRAK is not provided, the optional records are written to the
exception file (FFSEXCP). If your application data is in raw data format and FFSTRAK
is not provided, no optional records are written even if they are requested. FFSTRAK
must be large enough to accommodate the largest information record. FFSTRAK can
contain fixed or variable length records.

```
//*********************************************************************
//* If translating to standard formats:                           *
//*                                                               *
//* Specify the tracking file to hold IEGTQ type records if       *
//* they are to be separated from the exception file. This        *
//* is recommended if your application input data is in a raw data *
//* format. Allocate the same as FFSEXCP.                         *
//*********************************************************************
//*
//FFSTRAK  DD DSN=HOLD.TRAKING.FILE,DISP=MOD
//*
```

Section 8 FFSWORK file

FFSWORK is required only when translating from application format to EDI standard
format. FFSWORK holds the current transaction in case of translation errors. If an error

occurs, the current transaction is copied from the FFSWORK file to the exception file
(FFSEXCP). This data set can contain variable record format with a logical record
length of 32756. If your system guidelines allow, specifying UNIT=VIO on this DD
statement will drastically reduce import/export exceptions on the data set. Allocating
with a variable blocked (VB) record format will also reduce the number of I/O EXCPs on
the data set.

```
//***********************************************************************
//* If translating to standard or to application formats:        *
//*                                                              *
//* Specify the work file (temporary hold file during translation)   *
//***********************************************************************
//*
//FFSWORK  DD DSN=&&FFSWORK,DISP=(NEW,DELETE),UNIT=SYSALLDA,
//         DCB=(RECFM=V,BLKSIZE=32760),SPACE=(TRK,(1,1))
//*
```

Section 9 Envelope data file

If your Utility request involves enveloping and/or sending of transaction data, this is the
section of JCL that defines the EDI standard data file to envelope into and send from.
The ddname should match one of the following:

- The Trans data queue field value as specified in the Network Profile (NETPROF). If
  a name has not been supplied in the Network Profile member, the default is **QDATA**.
  The ddname specified holds transactions that have ISA, ICS, or GS enveloping. The
  same file name with an **E** appended (such as QDATAE) holds transactions that have
  either UNB or STX enveloping. The same file name with a **U** appended (such as
  QDATAU) holds transactions that have BG enveloping. There must be a ddname
  specified for each network that can be accessed during the run.

- The value specified in the FILEID keyword on the PERFORM command. If this
  keyword is specified, that file is used to hold all envelope types for all networks.

This data set can contain fixed-length or variable-length records with a logical record
length of 80 or greater.

```
//***********************************************************************
//* If this job step includes enveloping and/or sending data:    *
//*                                                              *
//* Specify the network transaction files to hold queued transactions *
//* The ddnames MUST match the transaction data queue field      *
//* in the network profile unless you will be using the FILEID    *
//* keyword override in the EDISYSIN input command language       *
//* statements.                                                  *
//*                                                              *
//* Use DISP=MOD to append data to the file                      *
//*     DISP=OLD to replace the file                             *
//*                                                              *
//* Suggested: DCB=(RECFM=V,LRECL=80,BLKSIZE=23440). However, this *
//*            might not satisfy YOUR network requirements, so it  *
//*            is suggested that you verify this allocation.       *
//***********************************************************************
//*
//QDATA    DD DSN=NETWORK.HOLDFILE.NAME,DISP=MOD
```

```
//*
```

Section 10 Network communications

This group of JCL statements is used when sending to or receiving from a network. The JCL shown here is required for the AT&T Global Network. Each network has its own JCL requirements.

Section 10a Communicating with Expedite Base/MVS using IEBASE: This section contains JCL statements for using the Expedite Base/z/OS network program IEBASE. The network profiles supplied by WebSphere Data Interchange are IINB41 (IEBASE Release 3.2) and IINB42 (IEBASE Release 4.2 and higher). The files described here are required by Expedite Base/z/OS. INMSG is the network input file and contains commands written by WebSphere Data Interchange for processing by Expedite Base/z/OS. OUTMSG is the network output file and contains the responses from Expedite Base/z/OS after processing. INB1STAT is used during status updating to hold network acknowledgments. For more information about these Expedite Base/z/OS files, refer to the Expedite Base/MVS Programming Guide.

```
//*************************************************************************
//* Specify the IGN Expedite Base/MVS network program (IEBASE)      *
//* required work files.  The network profile name is IINB42.       *
//*                                                                  *
//* DSNAME     DESCRIPTION                         RECFM    LRECL    *
//*                                                                  *
//* ERRORMSG   ERROR DESCRIPTIONS FILE             FB         80     *
//* ERRORTXT   EXTENDED ERROR DESCRIPTIONS FILE    FB         80     *
//* INB1STAT   STATUS INFORMATION FILE              V        255     *
//* INMSG      MESSAGE COMMAND FILE                FB         80     *
//* INPRO      PROFILE COMMAND FILE                FB         80     *
//* OUTMSG     MESSAGE RESPONSE FILE               FB         80     *
//* OUTPRO     PROFILE RESPONSE FILE               FB         80     *
//*                                                                  *
//* NOTE:  INPRO in sample JCL needs to be customized for your       *
//*        system.                                                   *
//*                                                                  *
//*************************************************************************
//*
//ERRORMSG DD DSN=SYS1.EXPBASE,EXPMSGS(ERRORMSG),DISP=SHR

//ERRORTXT DD DSN=SYS1.EXPBASE,EXPMSGS(ERRORTXT),DISP=SHR

//INB1STAT DD DSN=NETWORK.INB1STAT,DISP=OLD

//INMSG    DD DSN=&&INMSG,DISP=(NEW,DELETE),UNIT=SYSALLDA,

//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(1,1))

//INPRO    DD DSN=EDI.V1R3M0.SEDISAM1(EDIINPRO),DISP=SHR

//OUTMSG   DD DSN=NETWORK.OUTMSG,DISP=OLD

//OUTPRO   DD DSN=NETWORK.OUTPRO,DISP=OLD
```

If destination tables are needed to resolve Information Exchange (IE) mailboxes, a qualifier table (QUTTABLE) and one or more destination tables (TTABLExx) might be necessary. The ddname of the destination table(s) can be specified in the data of the

qualifier table and can be any valid ddname. If the qualifier table (QUTTABLE) is not used, the following default ddnames are used for the destination table(s):

X12   TTABLE*xx* - where *xx* is the two-character qualifier (ISA07 element of the interchange envelope)

UCS   TTABLE01

UN/TDI TTABLE

EDIFACT

TTABLE*xx* - where *xx* is the first two characters of the qualifier (UNTO:2 element of the interchange envelope)

```
//QUTTABLE DD DSN=USERFILE.IN.ANYNAME.SEQ.FILE,DISP=SHR

//TTABLExx DD DSN=USERFILE.IN.ANYNAME.SEQ.FILExx,DISP=SHR

//*
```

Section 10b Communicating with Expedite Base/z/OS using IEBASE and Comm-Press: this section contains JCL statements for using the Expedite Base/z/OS network program IEBASE with Comm-Press. In addition to the files mentioned in above, these files are required if you are using Expedite Base/z/OS with compression. For more information about these Expedite Base/z/OS files, refer to the Expedite Base/MVS Programming Guide.

```
//**********************************************************************
//* If you are using Expedite Base/MVS with Comm-Press, then specify  *
//* the following work files:                                         *
//*                                                                   *
//* DSNAME     DESCRIPTION                           RECFM    LRECL   *
//*                                                                   *
//* COMPPDS    COMPRESSION PDS FILE                  FB       80      *
//* COMPWRK    COMPRESSION WORK FILE                 FB       80      *
//* INMSGC     MESSAGE COMMAND FILE                  FB       80      *
//* INMSGR     MESSAGE RESPONSE FILE                 FB       80      *
//* OUTMSGC    MESSAGE COMMAND FILE                  FB       80      *
//* OUTMSGR    MESSAGE RESPONSE FILE                 FB       80      *
//* SYSUT1     TEMPORARY WORK FILE FOR COMPRESSION FB         80      *
//*                                                                   *
//* COMPTRC    COMPRESSION TRACE FILE                                 *
//* CPLOOKUP   COMPRESSION LOOKUP TABLE              FB       80      *
//*                                                                   *
//* NOTE:  COMPTRC is required when BASE(Y) is used on the trace      *
//*        command. CPLOOKUP is required when using COMPRESS(T).      *
//*                                                                   *
//**********************************************************************
//*
//COMPPDS  DD DSN=NETWORK.COMPPDS,DISP=(NEW,CATLG),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=0,DSORG=PO),
//            UNIT=SYSALLDA,SPACE=(CYL,(10,2,10))
//COMPWRK  DD DSN=NETWORK.COMPWRK,DISP=(NEW,CATLG),
//            DCB=(RECFM=V,LRECL=84,BLKSIZE=23440,DSORG=PS),
//            UNIT=SYSALLDA,SPACE=(TRK,(10,1))
```

```
//INMSGC   DD DSN=&&INMSGC,DISP=(NEW,DELETE),UNIT=SYSALLDA,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(1,1))
//INMSGR   DD DSN=&&INMSGR,DISP=(NEW,DELETE),UNIT=SYSALLDA,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(1,1))
//OUTMSGC  DD SYSOUT=*
//OUTMSGR  DD DSN=NETWORK.OUTMSGR,DISP=(NEW,CATLG),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//            UNIT=SYSALLDA,SPACE=(TRK,(10,1))
//SYSUT1   DD DSN=&&SYSUT1,DISP=(NEW,DELETE,DELETE),
//            UNIT=SYSDA,DCB=BLKSIZE=23476,SPACE=(CYL,(1,1))
//COMPTRC  DD SYSOUT=*
//CPLOOKUP DD DSN=NETWORK.CPLOOKUP,DISP=(NEW,CATLG),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//            UNIT=SYSALLDA,SPACE=(TRK,(10,1))
//*
```

Section 11 EDI standards

For receiving and deenveloping transaction data, this section of JCL defines the EDI standard data file to receive into and deenvelope from. The ddname should match one of the following:

- The Receive file name field value as specified in the mailbox (requestor) profile (REQPROF) associated with the REQID keyword in the PERFORM command. There must be a ddname here for each requestor that can be processed during the run.
- The value specified with the FILEID keyword in the PERFORM command. The value specified here overrides the Receive file name value from the mailbox (requestor) profile.

This data set can contain fixed-length or variable-length records with a logical record length of 80 or greater.

```
//********************************************************************
//* If this job step includes receiving and/or deenveloping of      *
//* transaction data:                                               *
//*                                                                 *
//* Specify the Network Transaction files to receive into and/or    *
//* deenvelope from.                                                *
//*                                                                 *
//* The ddname MUST match the receive file name in the Requestor    *
//*   profile or the FILEID override keyword in the EDISYSIN input   *
//*   command language statements.                                  *
//* A ddname is required for each unique ddname specified by a      *
//*   Requestor ID (receive ddname obtained from Requestor profile). *
//* Use DISP=MOD to append data to the file                         *
//*     DISP=OLD to replace the file                                *
//********************************************************************
//*
//REQ1DD    DD DSN=REQ1.TRANS,DISP=MOD
```

```
//REQ2DD    DD DSN=REQ2.TRANS,DISP=MOD
//*
```

Section 12 ddname

This section defines the ddname for receiving data. The ddname is specified in the `Application file name` field in the data format. You must specify a ddname for each data format that can be processed during the run. The name specified in the data format can be overridden with a value in the `Application file name` field in the receive usage/rule. If there is no applicable DD statement, the application data is written to the exception file (FFSEXCP). In the example below, INVOICE identifies a file that holds data in application format when translating from EDI standard format.

This data set can contain fixed or variable length records. The logical record length must be large enough to hold the largest application record.

```
//***********************************************************************
//* If translating to application formats:                           *
//*                                                                   *
//* Specify the sequential application (output) files                *
//*                                                                   *
//* ddnames MUST be supplied for every Data Format                   *
//*   that can be received in this job and they must match the       *
//*   ddname specified in defining the Data Format.                  *
//* Multiple job steps can be used if multiple network transaction   *
//*   files must be separated into unique application files.         *
//* Transactions will either replace the file or are appended to     *
//*   the file based on the DISP parameter.                          *
//* Use DISP=MOD to append data to the file                         *
//*     DISP=OLD to replace the file                                *
//* Allocation parameters should handle your largest data record     *
//* plus any informational records you might request.                *
//***********************************************************************
//*
//INVOICE   DD DSN=INVOICE.DATA.FILE,DISP=MOD
//PURCORD   DD DSN=PURCORD.DATA.FILE,DISP=MOD
//*
```

Section 13 Report file

This section defines the file that contains reports generated by the PRINT command and by various other PERFORM commands.

This data set can contain fixed block addressing (FBA) or variable block addressing (VBA) records with a logical record length of 132.

```
//***********************************************************************
//*                                                                   *
//*                   PLEASE READ!!!!                                *
//*                                                                   *
//* The following DD assignments (up to the EDISYSIN DD assignment)   *
//* are used for specific commands which most likely will not be      *
```

```
//* used too often. Please read the comments carefully to determine  *
//* if the ddnames should be allocated or removed.                    *
//**********************************************************************
//*
//**********************************************************************
//* If using any of the PRINT commands:                               *
//*                                                                    *
//* Specify the output file to write the reports out to.              *
//*                                                                    *
//* Use DISP=MOD to append to the existing file                       *
//*     DISP=OLD to replace the file                                  *
//* Allocation parameters should indicate a record length of 133.     *
//**********************************************************************
//*
//RPTFILE   DD DSN=REPORT.FILE,DISP=MOD
//*
```

Section 14 Output file

This section defines the file that contains output from command processing. EDIQUERY identifies a file that contains output from QUERY commands and from various other PERFORM commands.

This data set should contain variable length records with a block size of 32760.

```
//**********************************************************************
//* If using the following commands:                                  *
//*      TRADING PARTNER PROFILE DATA EXTRACT                         *
//*      TRADING PARTNER CAPABILITY DATA EXTRACT                      *
//*      TRANSACTION ACTIVITY DATA EXTRACT                            *
//*      NETWORK ACTIVITY DATA EXTRACT                                *
//*      TRANSACTION DATA EXTRACT                                     *
//*      ENVELOPE DATA EXTRACT                                        *
//*      QUERY                                                        *
//*                                                                    *
//* Specify the output file to write the extracted data or matching   *
//* handles to.                                                       *
//*                                                                    *
//* Use DISP=MOD to append to the existing file                       *
//*     DISP=OLD to replace the file                                  *
//*                                                                    *
//* Allocation parameters should indicate a variable record format    *
//* with a block size of 32760. This will allow any type of data      *
//* to be written to the query file without the possibility of        *
//* records being truncated.                                          *
//**********************************************************************
//*
//EDIQUERY  DD DSN=QUERY.FILE,DISP=MOD
//*
```

Section 15 Export/Import statements

This section of JCL is required for exporting and importing.

```
//***********************************************************************
//* If using the EXPORT or IMPORT commands:                            *
//*                                                                     *
//* 1) Allocate the export/import DI control file.                     *
//*                                                                     *
//* 2) Allocate the export/import user supplied control file.          *
//*    This control file describes what data is to be exported or      *
//*    imported. An allocation of fixed format and record length 84    *
//*    should be used. This file can also be allocated inline.         *
//*    Make sure to use the same ddname allocated with the             *
//*    CTLFILE keyword in the EDISYSIN input command language          *
//*    statements. In this sample CTLFILE(EXIMCTL) would be used.      *
//*                                                                     *
//* 3) Allocate the DD statements for the export/import files          *
//*    themselves. All the ddnames can point to a single file          *
//*    or they can each point to a separate file. This sample          *
//*    JCL shows the ddnames pointing to separate files.               *
//*    Allocation of the dataset(s) should have a record format        *
//*    of variable, a record length of 4089, and a block size of 4093.*
//*    Use DISP=MOD to append to the existing file(s)                  *
//*        DISP=OLD to replace the file(s)                             *
//***********************************************************************
//*
```

Section 15a Export/Import control statements: This section defines the control statements describing what data should be exported or imported. This ddname must match the value specified in the CTLFILE keyword on the PERFORM command. The export and import control statements are described in "Export/Import control file (CTLFILE)" on page 209.

The record format should be fixed with a logical record length of 84.

```
//***********************************************************************
//* Step 1 - Allocate user supplied control file
//***********************************************************************
//*
//EXIMCTL   DD DSN=EXPORT.IMPORT.CTLFILE,DISP=OLD
```

Section 15b Export/Import files: The remaining EDIEI*xxx* DD statements define files to which data is exported, or from which data is imported. You can assign all DD statements to a single physical file, or separate them as shown in this example. These data sets contain variable length records with a logical record length of 4089.

```
//***********************************************************************
//* Step 2 - Allocate export/import files
//***********************************************************************
//*
//EDIEIADF  DD DSN=EXPORT.IMPORT.ADF,DISP=MOD
//EDIEICST  DD DSN=EXPORT.IMPORT.CST,DISP=MOD
//EDIEIPRF  DD DSN=EXPORT.IMPORT.PRF,DISP=MOD
//EDIEISTD  DD DSN=EXPORT.IMPORT.STD,DISP=MOD
//EDIEITBL  DD DSN=EXPORT.IMPORT.TBL,DISP=MOD
```

```
//EDIEITPT  DD DSN=EXPORT.IMPORT.TPT,DISP=MOD
//*
```

Section 16 Functional acknowledgment overrides

This section defines override data that the translator uses to generate functional acknowledgments. FAENV is an optional file processed during deenvelope functions. For detailed information about the format of this file, see "Enveloping options file for functional acknowledgments (FAENV)" on page 203.

This data set can contain fixed or variable length records. The logical record length must be as large as the largest record in the file.

```
//**********************************************************************
//* If deenveloping data:                                             *
//*                                                                   *
//* Specify the sequential file which contains overrides              *
//* that you want used when functional acknowledgments are            *
//* generated for the data that is being deenveloped.                 *
//*                                                                   *
//* This file is optional and if not specified the envelope data      *
//* for functional acknowledgments will be taken from the standard    *
//* profile member specified for the transactions.                    *
//*                                                                   *
//* Suggested: DCB=(RECFM=V,LRECL=255)                                *
//**********************************************************************
//*
//FAENV     DD DSN=FA.OVERRIDES.FILE,DISP=SHR
//*
```

Section 17 Perform command file

This section defines the file from which the Utility PERFORM commands are read. If EDISYSIN is not defined, the Utility checks SYSIN for the PERFORM commands. For more information on these commands, see Chapter 2, "WebSphere Data Interchange commands and keywords," on page 19.

This data set can contain fixed or variable length records with any logical record length.

```
//**********************************************************************
//*            VERY IMPORTANT: EDISYSIN DD ASSIGNMENT                  *
//*                                                                   *
//* The EDISYSIN device contains all command language statements      *
//* to be processed in this job step. In this sample JCL it           *
//* has been allocated inline, but EDISYSIN can be allocated to any   *
//* dataset containing command language input. If it is allocated     *
//* to a dataset, any record length and format can be used.           *
//* Make sure sequence numbers do not appear in cols 73-80; this will *
//* cause errors.                                                     *
//*                                                                   *
//* Note: The WebSphere Data Interchange Utility will first look      *
//* for PERFORM                                                       *
```

```
//*       commands in EDISYSIN.  If there is no EDISYSIN defined,    *
//*       the Utility will look in SYSIN.                            *
//********************************************************************
//*
//EDISYSIN  DD *
* An asterisk in column one denotes a comment line..

* -------------------------------------------------------------------- *
* Here is a sample of command language input to translate and
* envelope transactions from application file APDATA01 and generate
* optional records.
* -------------------------------------------------------------------- *
PERFORM TRANSLATE AND ENVELOPE

WHERE APPFILE(APDATA01) OPTRECS(IEGTQ)

* -------------------------------------------------------------------- *
* Here is a sample of command language input to send queued
* transactions on behalf of requestor ID ROBOX
* -------------------------------------------------------------------- *
PERFORM SEND WHERE REQID(ROBOX)

* -------------------------------------------------------------------- *
* Here is a sample of command language input to receive transactions
* from requestor ID ROBOX
* -------------------------------------------------------------------- *
PERFORM RECEIVE WHERE REQID(ROBOX)

* -------------------------------------------------------------------- *
* Here is a sample of command language input to deenvelope and
* translate received transactions associated with requestor ID ROBOX
* -------------------------------------------------------------------- *
PERFORM DEENVELOPE AND TRANSLATE

WHERE REQID(ROBOX)
/*
```

Section 18 Pageable translation work file

This section identifies the pageable translation work file, EDIVAX. EDIVAX should be defined as a temporary data set. Space allocation is dependent on the amount of data that is paged. Pageable translation is specified by using the PAGE keyword available on all translate type commands, and on all envelope/deenvelope type commands. For information regarding EDIVAX space allocation and pageable translation in general, see the PAGE keyword description on 166.

```
//********************************************************************
//* Specify the Pageable Translation work file.  Uncomment the      *
//* following DD statement if you use this feature.  The space       *
//* allocation is dependent on the amount of data to be paged.       *
//********************************************************************
//*
//*EDIVAX   DD DISP=(NEW,DELETE,DELETE),UNIT=SYSDA,SPACE=(CYL,1500)
//*
```

Section 19 DB2 parameters

This section is for DB2 only. It is typical in a DB2 environment that batch application programs (such as the WebSphere Data Interchange Utility) run under the TSO Terminal Monitor Program (TMP) in background mode. This is specified by naming IKJEFT01 in the EXEC JCL statement as in the following example:

```
//XDIUTIL EXEC PGM=IKJEFT01,DYNAMNBR=50
```

The parameters passed to IKJEFT01 are specified in the SYSTSIN data set. These parameters include the DB2 subsystem ID, the DB2 plan, the name of the application program, and the application program parameter string. IKJEFT01 connects DB2 and opens the plan, runs the application program (EDIFFUT), and then closes the plan and disconnects DB2. When EDIFFUT executes via IKJEFT01 the DB2 processing mode is called DSN. This processing mode assumes that data set EDITSIN does not exist, and that the WebSphere Data Interchange Utility parameters are specified in SYSTSIN. See "Section 2 WebSphere Data Interchange Utility parameters" on page 408 for more information.

```
//***********************************************************************
//* Specify the parameters to DB2 to execute the utility              *
//*                                                                    *
//* *KEY* These are the statements which run the DB2 program.          *
//*                                                                    *
//*   1. If necessary, change "DSN" to your local DB2 subsystem ID.    *
//*   2. If necessary, change the PLAN (EDIENU32) to match the         *
//*      the plan name you used when binding the plan during install.  *
//*   3. If necessary, change the parm "LANGID=ENU" to match           *
//*      the national language you are using.                          *
//*   4. If necessary, change the parm "SYSID=DIENU" to match          *
//*      your WDI RACF installation.                                   *
//*   5. If necessary, change the parm "APPLID=EDIFFS" to match        *
//*      the desired WDI application ID.                               *
//*   6. If necessary, change the parm "PLAN=EDIENU32" to match        *
//*      the plan name you used when binding the plan during install.  *
//*   7. If necessary, change the parm "SYSTEM=DSN" to match           *
//*      your local DB2 subsystem ID.                                  *
//***********************************************************************
//*
//SYSTSIN  DD *
  DSN SYSTEM (DSN)
  RUN PROG (EDIFFUT) -
  PLAN (EDIENU32) -
  PARM('LANGID=ENU SYSID=DIENU APPLID=EDIFFS
PLAN=EDIENU32 SYSTEM=DSN')
  END
/*
```

Section 20 XML parameters

This section is for XML processing. The EDI.XML parameters (in the last line of code) should be edited as appropriate for your installation of XML Toolkit Version 1 Release 2, replacing SYS1 with the high level qualifier for your system.

```
//STEPLIB          DD DSN=EDI.V3R2M0.SEDILMDI,DISP=SHR
              DD DSN=EDI.V3R2M0.SEDILXML1,DISP=SHR
              DD DSN=DB93.DSNEXIT,DISP=SHR
              DD DSN=DB93.DSNLOAD,DISP=SHR
              DD DSN=EDI.SNA131.LOADLIB,DISP=SHR
              DD DSN=SYS.XML.SIXMMOD1,DISP=SHR
```

Required utility data sets

Table 113 lists outlines the sections of JCL required for the different WebSphere Data Interchange commands. The numbers in the table refer to the section headings described earlier in this chapter. An asterisk (*) after the number indicates that the DD statement is optional.

Note: Sections 1 through 6 are required for all commands.

Table 113. Required utility data sets

Command	JCL sections						
DEENVELOPE	6	11	16	17			
DEENVELOPE AND TRANSLATE	6	7*	11	12	16*	17	
ENVELOPE	6	7*	8	9	17		
ENVELOPE AND SEND	6	7*	8	9	10	17	
ENVELOPE DATA EXTRACT	14	17					
EXPORT	15	17					
HOLD	17						
IMPORT	15	17					
NETWORK ACTIVITY DATA EXTRACT	14	17					
PRINT	13	17					
PURGE	17						
QUERY	14	17					
RECEIVE	10	11	17				
RECEIVE AND DEENVELOPE	6	10	11	16*	17		
RECEIVE AND SEND	6	7*	9	11	16*	17	
RECEIVE AND TRANSLATE	6	10	11	12	16*	17	
RECVFILE AND SEND	6	7*	8	9	10	11	17
REENVELOPE	6	7*	8	9	17		
RELEASE	17						
REMOVE TRANSACTIONS	17						
RETRANSLATE TO APPLICATION	6	12	17				
SEND	9	10	17				

Table 113. Required utility data sets (continued)

Command	JCL sections						
TRADING PARTNER CAPABILITY DATA EXTRACT	14	17					
TRADING PARTNER PROFILE DATA EXTRACT	14	17					
TRANSACTION ACTIVITY DATA EXTRACT	14	17					
TRANSACTION DATA EXTRACT	14	17					
TRANSFORM	5	6	17	20			
TRANSLATE TO APPLICATION	6	12	17				
TRANSLATE AND ENVELOPE	5	6	7*	8	9	17	
TRANSLATE TO STANDARD	5	6	7*	8	17		
TRANSLATE AND SEND	5	6	7*	8	9	10	17
UNPURGE	17						
UPDATE STATUS	10	17					

Archive DB2 event log entries (EDIELARD)

This section shows the DB2 event log archive/removal JCL from the JCL distribution library (EDI.V3R2M0.SEDIINS1). For additional information, see "Reporting and extracting data" on page 8.

```
//EDIELARD JOB (INSTALLATION DEPENDENCIES)
//*
//**********************************************************************
//* THIS SAMPLE JCL WILL ARCHIVE AND PURGE LOG ENTRIES FROM A DB2     *
//* EVENT LOG. THIS JCL MUST BE MODIFIED FOR YOUR LOCAL ENVIRONMENT.  *
//**********************************************************************
//* 1. CHANGE JOB CARD STATEMENT TO LOCAL REQUIREMENTS               *
//* 2. CHANGE ARCHIVE FILE (ARCFILE) AND THE HOLD FILE (HLDFILE) DD  *
//* STATEMENTS AS NECESSARY.                                         *
//* 3. IF NECESSARY, CHANGE THE PARM "DIENU" IN THE STEP "ARCHIVE"   *
//* FOR YOUR WDI RACF INSTALLATION.                                  *
//* 4. IF NECESSARY, CHANGE DB2 "SYSTEM", "PLAN", AND "LIB"          *
//* PARAMETER VALUE IN SYSTSIN DATA STREAMS.                         *
//**********************************************************************
//*
//**********************************************************************
//* THE ENTRIES SELECTED FOR REMOVAL WILL BE COPIED TO THE ARCHFILE, *
//* AND ALL OTHER ENTRIES WILL BE COPIED TO THE HOLDFILE.            *
//**********************************************************************
//CREATE EXEC PGM=IEFBR14
//ARCFILE DD DSN=EDIENU.V3R2M0.ARCFILE,DISP=(NEW,CATLG),
// DCB=(RECFM=V,LRECL=4096,BLKSIZE=0),
// UNIT=SYSALLDA,SPACE=(CYL,(10,5))
//HLDFILE DD DSN=EDIENU.V3R2M0.HLDFILE,DISP=(NEW,CATLG),
```

```
// DCB=(RECFM=V,LRECL=4096,BLKSIZE=0),
// UNIT=SYSALLDA,SPACE=(CYL,(10,5))
//*
//ARCHIVE EXEC PGM=IKJEFT01,DYNAMNBR=50
//*
//STEPLIB DD DSN=DB2.SDSNLOAD,DISP=SHR <--DB2
LOAD LIBRARY
// DD DSN=EDI.V3R2M0.SEDILMD1,DISP=SHR <--EDI
LOAD LIBRARY
//*
//***********************************************************************
//* SPECIFY EDI REQUIRED FILES                                         *
//* THE PHYSICAL FILE NAMES ARE INSTALLATION DEPENDENT.                *
//* LOGFFS IS THE FLAT FILE SUPPORT PRIVATE LOG FILE                   *
//***********************************************************************
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//PRTFILE DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=132)
//ARCFILE DD DSN=EDIENU.V3R2M0.ARCFILE,DISP=SHR
//HLDFILE DD DSN=EDIENU.V3R2M0.HLDFILE,DISP=SHR
//*
//SYSIN DD *
PERFORM UNLOAD LOG ENTRIES WHERE APPLID(EDIFFS)
LOGDATE(10/01/01) TO(12/31/01)
ARCHIVEFILE(ARCFILE) HOLDFILE(HLDFILE)
/*
//SYSTSIN DD *
DSN SYSTEM (DSN)
RUN PROG (EDIFFUT) -
PLAN (EDIENU32) -
PARM('LANGID=ENU SYSID=DIENU APPLID=EDIFFS
PLAN=EDIENU32 SYSTEM=DSN')
END
/*
//***********************************************************************
//* REORG THE DB2 TABLE EDIELOG (EVENT LOG)                            *
//***********************************************************************
//REORG EXEC PGM=DSNUTILB,PARM='DSN,DSNTEX',COND=(4,LT,ARCHIVE)
//STEPLIB DD DSN=DB2.SDSNLOAD,DISP=SHR <--DB2 LOAD LIBRARY
//SYSPRINT DD SYSOUT=*
//UTPRINT DD SYSOUT=*
//SORTOUT DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
//SORTWK01 DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
```

Archive DB2 event log entries

```
//UNLD DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
//WORK DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
//SYSUT1 DD UNIT=SYSALLDA,SPACE=(4000,(20,20),,,ROUND)
//SYSIN DD *
REORG TABLESPCE (EDIENU32.EDIELOG)
UNLDDN (UNLD)
WORKDDN (UNLD)
REORG INDEX (EDIENU32.EDIELOGX)
/*
```

Application Program Interfaces (API's)

You can request WebSphere Data Interchange services directly from an application program. There are application program interfaces (APIs) defined for the following services:

- "Environmental services" on page 438
- "Translation services" on page 443
- "Enveloping services" on page 500
- "Data extraction services" on page 535
- "Communication services" on page 540
- "Update status services" on page 561
- "SYNCPOINT services" on page 568

The API for each of these services is described in detail later in this chapter. The information common to all the APIs is described in "API languages" on page 426.

The following control blocks are common to all service requests:

- Common control block (CCB)

  WebSphere Data Interchange uses the CCB to maintain status across all services requested by the application and to let your application know if a service request was successful. The return code and extended return code fields in the CCB indicate the degree of success. The CCB is initialized by environmental services. After initialization, the CCB must not be altered by the application program. You must use the same CCB on all service requests. For a complete description of the CCB, see "Common Control Block (CCB)" on page 636.

- Service name block (SNB)

  The service being requested. Where each API is described in this chapter, the logical name associated with the API is provided. You must place this logical name in the SNB before issuing a service request. The SNB is also used to indicate the number of parameters provided to the service. Applications using the API must define an SNB for each service requested. For a complete description of the SNB, see "Service Name Block (SNB)" on page 632.

- Function control block (FCB)

The particular service function being requested. Where each API is described in this chapter, the function value associated with the block is provided. You must place this function value in the FCB when requesting a service. For a complete description of the FCB, see "Function control block (FCB)" on page 640.

API languages

You can access WebSphere Data Interchange services with a simple call statement to a WebSphere Data Interchange-provided *stub* program. WebSphere Data Interchange provides four stub programs, one for each application programming language directly supported by WebSphere Data Interchange. These stub programs and associated languages are:
* FXXZCBL for COBOL programs
* FXXZPLI for PL/I programs
* FXXZC for IBM C/370 programs (z/OS only)
* FXXZASM for Assembler programs

Although WebSphere Data Interchange directly supports only these languages, any language that can create an operating-system-style parameter list, and that uses operating system linkage and register conventions, can request WebSphere Data Interchange services by using the FXXZASM stub program.

API link edit

The load library distributed with WebSphere Data Interchange contains a load module for each stub program. These stub programs are linked with the application program requesting the services. WebSphere Data Interchange is not physically part of the application load module. You must use the stub programs to access WebSphere Data Interchange.

When linking an application program, the WebSphere Data Interchange distribution load library must be part of the //SYSLIB concatenation so the linkage editor can resolve references to the stub programs. You can also use a separate DD statement for the WebSphere Data Interchange load library, and a specific INCLUDE statement in the linkage editor control cards to pull in the language stub that is needed.

There are no special residency or addressing requirements for an application program requesting WebSphere Data Interchange services. The application program can be above or below the 16 MB line, and all parameters passed to WebSphere Data Interchange through the stub program can also be above or below the 16 MB line. The relationship between the application load module, the stub programs, and WebSphere Data Interchange is illustrated in .
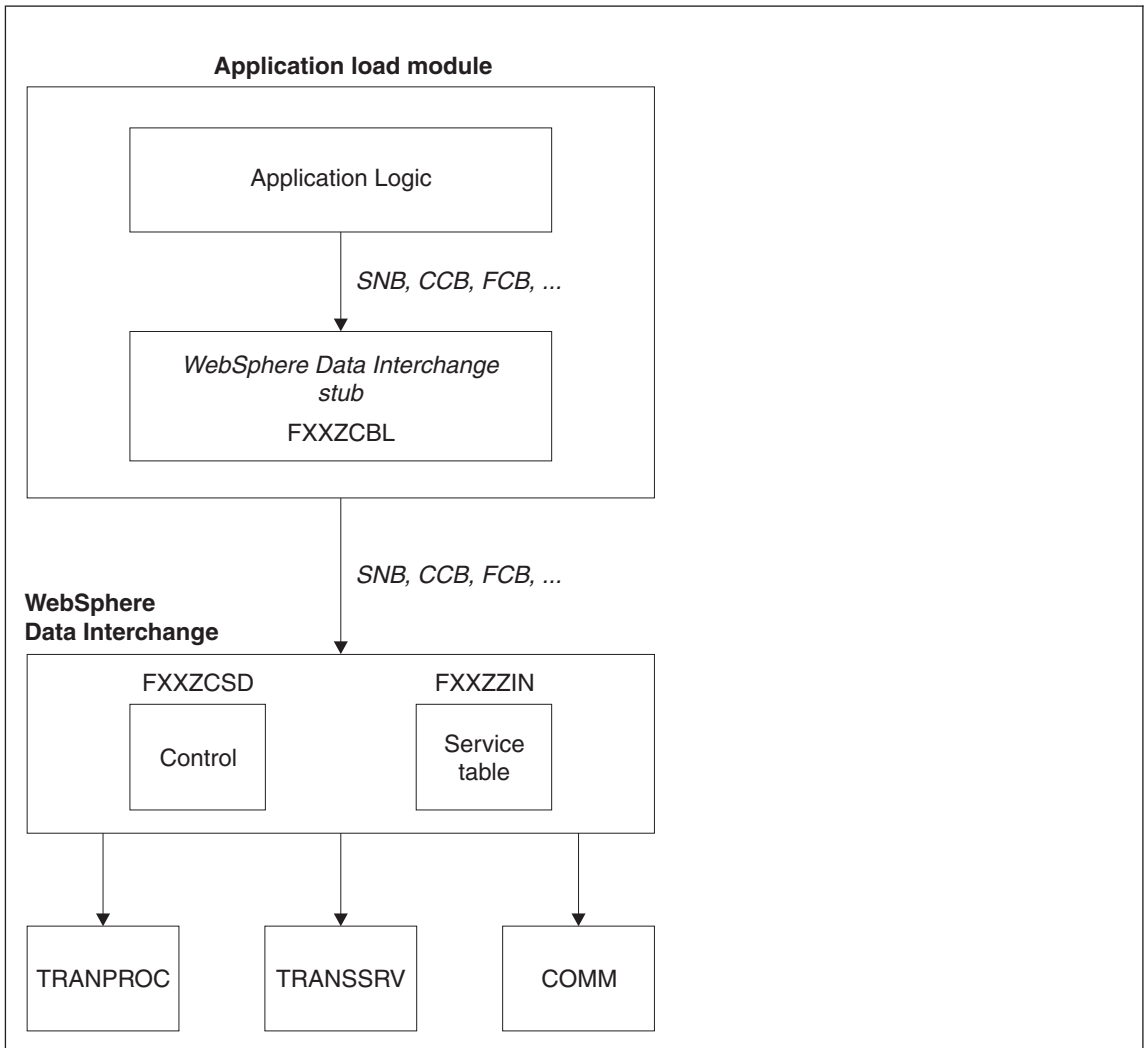
Figure 22. Load Module Relationships

WebSphere Data Interchange and DB2 attachment

For the most part, the DB2 processing mode that WebSphere Data Interchange uses is determined by the existence of a file named EDITSIN. If EDITSIN exists, the DB2 processing mode is **CAF**.

- If EDITSIN does not exist, the DB2 processing mode is **DSN,** and WebSphere Data Interchange does not attempt to attach or detach DB2 (an attachment is assumed to have taken place prior to WebSphere Data Interchange initialization).

- If EDITSIN exists, OPEN is not **N**, CAF is not **Y**, and WebSphere Data Interchange detects that DB2 is already attached while trying to make the DB2 attachment, the DB2 processing mode becomes **DSN**. This exception is true of the WebSphere Data Interchange facility CLIST.

- If OPEN is not **N** and values are supplied for SYSTEM and PLAN, WebSphere Data Interchange attempts to attach DB2.

- If WebSphere Data Interchange finds that DB2 is already attached and CAF is not **Y**, the DB2 processing mode becomes **DSN**.

- If CLOSE is not **N** and if values are supplied for SYSTEM and PLAN, WebSphere Data Interchange termination will detach DB2.

The keywords used in EDITSIN are:

SYSTEM
The name of the DB2 subsystem (or group, if data sharing). This value can be up to four characters long and is required if EDITSIN exists.

PLAN The DB2 plan name. This value can be up to eight characters long and is required if EDITSIN exists.

OPEN Indicates whether WebSphere Data Interchange must attempt to attach to DB2 during initialization. The default is **Y**.

CLOSE
Indicates whether WebSphere Data Interchange must detach from DB2 during WebSphere Data Interchange termination. The default is **Y**.

CAF If WebSphere Data Interchange initialization detects while trying to make a DB2 attachment that DB2 is already attached, the WebSphere Data Interchange DB2 processing mode becomes **DSN**. A value of **Y** for this keyword acts as an override and tells WebSphere Data Interchange to keep the DB2 processing mode CAF. The default is **N**.

EDITSIN examples

Example 1:
Your DB2 subsystem is named DB93, your DB2 plan is named **EDIENU32**, and you want WebSphere Data Interchange to handle the DB2 attachment and detachment. Include the following keywords and values in EDITSIN. The **OPEN** and **CLOSE** keywords are not needed because the default action is for WebSphere Data Interchange to handle DB2 attachment.

```
SYSTEM(DB93) PLAN(EDIENU32)
```

Example 2:

Your DB2 subsystem is named DB93, your DB2 plan is named EDIENU32, and you do not want WebSphere Data Interchange to attach or detach DB2. Include the following keywords and values in EDITSIN.

```
SYSTEM(DB93) PLAN(EDIENU32) OPEN(N) CLOSE(N)
```

Example 3:

Your DB2 subsystem is named DB93, your DB2 plan is named EDIENU32, DB2 might be attached, but, regardless, you do not want WebSphere Data Interchange to detach from DB2 on termination, and you want WebSphere Data Interchange to process in **CAF** mode regardless of whether DB2 was previously attached or not. Include the following keywords and values in EDITSIN.

```
SYSTEM(DB93) PLAN(EDIENU32) CLOSE(N) CAF(Y)
```

WebSphere Data Interchange abend return codes

When a CICS abend is detected, WebSphere Data Interchange returns **-8** in the ZCCBRC field and the EBCDIC representation of the CICS abend code in the ZCCBERC field. This return code combination is global and is not mentioned in upcoming tables.

COBOL calls

When using COBOL, you invoke WebSphere Data Interchange as a normal call to an external function. The call is made to FXXZCBL and is coded as follows:

```
CALL FXXZCBL USING SNB CCB FCB [other parms]
```

WebSphere Data Interchange assumes that all parameters are pointers to control blocks. COBOL passes control block pointers as part of normal processing.

Code fragments for initializing WebSphere Data Interchange with COBOL might look like the following samples.

SNB-COBOL

```
01 SNB-DATA.
   03 ZSNBLL    PIC S9(4) COMP-4.
   03 ZSNBID    PIC S9(4) COMP-4.
   03 ZSNBEYE   PIC X(8).
   03 ZSNBNAME   PIC X(8).
   03 ZSNBNDX   PIC S9(9) COMP-4.
   03 ZSNBPC    PIC S9(4) COMP-4.
   03 ZSNBFLG0   PIC X(1).
   03 ZSNBFLG1   PIC X(1).
  03 ZSNBFANC   PIC S9(9) COMP-4.
```

CCB-COBOL

```
01 CCB-DATA.
   03 ZCCBLL    PIC S9(4) COMP-4.
   03 ZCCBID    PIC S9(4) COMP-4.
   03 ZCCBEYE   PIC X(8).
   03 ZCCBRC    PIC S9(9) COMP-4.
   03 ZCCBERC   PIC S9(9) COMP-4.
```

```
        03 ZCCBSID    PIC X(8).
        03 ZCCBUID    PIC X(8).
        03 ZCCBAID    PIC X(8).
        03 ZCCBCID    PIC X(8).
        03 ZCCBXFID   PIC S9(4) COMP-4.
        03 ZCCBLPID   PIC X(6).
        03 ZCCBCPID   PIC S9(9) COMP-4.
        03 ZCCBRSV    PIC S9(9) COMP-4.
        03 ZCCBCCXP   PIC S9(9) COMP-4.
        03 ZCCBCABP   PIC S9(9) COMP-4.
        03 ZCCBDBID   PIC X(4).
        03 ZCCBDBPL   PIC X(8).
        03 ZCCBDBUI   PIC X(8).
        03 ZCCBDBPW   PIC X(18).
        03 ZCCBDSV1   PIC X(26).
        03 ZCCBRSV2   PIC S9(9) COMP-4 OCCURS 117 TIMES.
```

FCB-COBOL

```
 01 FCB-DATA.
    03 ZFCBLL    PIC S9(4) COMP-4.
    03 ZFCBFUNC  PIC S9(4) COMP-4.
```

INIT-COBOL

```
    01   APPLID  PIC X(08).
    01   SYSID   PIC X(08).
    MOVE LOW-VALUES    TO SNB-DATA CCB-DATA FCB-DATA.
    MOVE 'ENVSERV '    TO ZSNBNAME.
    MOVE 5    TO ZSNBPC.
    MOVE 'ENU '        TO ZCCBLPID.
    MOVE 1    TO ZFCBFUNC.
    MOVE 'MYAPPL '     TO APPLID.
    MOVE 'SYSTEM '     TO SYSID.
    CALL FXXZCBL USING SNB-DATA CCB-DATA FCB-DATA APPLID SYSID.
    IF ZCCBRC EQUAL ZERO
*  initialization worked
    ELSE
*  initialization failed
    END-IF.
```

PL/I calls

When using PL/I, you invoke WebSphere Data Interchange as a normal call to an
external function. The call is made to FXXZPLI and is coded as follows:

```
CALL FXXZPLI(SNB,CCB,FCB,other parms)
```

WebSphere Data Interchange assumes that all parameters are pointers to control
blocks. PL/I programs would not normally pass on control block pointers to an external
routine. PL/I, unless told otherwise, assumes that the call is being made to another
program written in PL/I and passes special PL/I parameter information along with the
parameters. Because WebSphere Data Interchange does not understand this
information, you must notify the compiler that FXXZPLI is an assembler program with
the following statement:

```
DCL FXXZPLI EXTERNAL ENTRY OPTIONS (ASSEMBLER,INTER)
```

Code fragments for initializing WebSphere Data Interchange with PL/I might look like the following samples.

SNB-PL/I
```
DCL 1 SNB,
    3 ZSNBLL    FIXED BIN(15),              /* SNB block length        */
    3 ZSNBID    FIXED BIN(15),              /* Reserved for future use  */
    3 ZSNBEYE   CHAR(8),                    /* SNB block name          */
    3 ZSNBNAME  CHAR(8),                    /* Service name             */
    3 ZSNBNDX   FIXED BIN(31),              /* Service index           */
    3 ZSNBPC    FIXED BIN(15),              /* Service parameter count  */
    3 ZSNBFLG0  CHAR(1),                    /* First flag byte         */
    3 ZSNBFLG1  CHAR(1),                    /* Second flag byte        */
    3 ZSNBFANC  POINTER;                    /* First anchor pointer    */
```

CCB-PL/I
```
DCL 1 CCB,
    3 ZCCBLL    FIXED BIN(15),          /* CCB block length           */
    3 ZCCBID    FIXED BIN(15),          /* Reserved for future use    */
    3 ZCCBEYE   CHAR(8),                /* CCB block name             */
    3 ZCCBRC    FIXED BIN(31),          /* Return code                */
    3 ZCCBERC   FIXED BIN(31),          /* Extended return code       */
    3 ZCCBSID   CHAR(8),                /* System ID                  */
    3 ZCCBUID   CHAR(8),                /* User ID                    */
    3 ZCCBAID   CHAR(8),                /* Application ID             */
    3 ZCCBCID   CHAR(8),                /* Error module ID            */
    3 ZCCBXFID  FIXED BIN(15),          /* Component function ID      */
    3 ZCCBLPID  CHAR(6),                /* Language profile ID        */
    3 ZCCBCPID  FIXED BIN(31),          /* Code page ID               */
    3 ZCCBRSV   FIXED BIN(31),          /* Reserved for future use    */
    3 ZCCBCCXP  POINTER,                /* CCB extension pointer      */
    3 ZCCBCABP  POINTER,                /* Common area block pointer*/
    3 ZCCBDBID  CHAR(4),                /* DB2 subsystem ID           */
    3 ZCCBDBPL  CHAR(8),                /* DB2 plan / AIX alias       */
    3 ZCCBDBUI  CHAR(8),                /* AIX DB2 user ID            */
    3 ZCCBDBPW  CHAR(18),               /* AIX DB2 password           */
    3 ZCCBRSV1  CHAR(26),               /* DI internal use only       */
    3 ZCCBRSV2(117)   FIXED BIN(31);    /* DI internal use only       */
```

FCB-PL/I
```
DCL 1 FCB,
    3 ZFCBLL    FIXED BIN(15),              /* FCB block length        */
    3 ZFCBFUNC  FIXED BIN(15);              /* Service function code   */
```

INIT-PL/I
```
DCL    APPLID   CHAR(8) INIT('MYAPPL ');
DCL    SYSID    CHAR(8) INIT('SYSTEM  ');

SNB = ";CCB = "; FCB = ";
SNB.ZSNBNAME = 'ENVSERV ';
```

```
SNB.ZSNBPC   = 5;
CCB.ZCCBLPID = 'ENU   ';
FCB.ZFCBFUNC = 1;
CALL FXXZPLI(SNB,CCB,FCB,APPLID,SYSID);
IF CCB.ZCCBRC = 0 THEN DO;
/* initialization worked    */
END;
ELSE DO;
/* initialization failed    */
END;
```

C calls

The FXXZC stub program is intended for use with the IBM C/370 compiler and is only supported in the z/OS environment. Other environments are not supported. When using C, you invoke WebSphere Data Interchange as a normal function call. The call is made to FXXZC and is coded as follows.

```
FXXZC(&MYSNB,&MYCCB,&MYFCB,other parms)
```

WebSphere Data Interchange assumes that all parameters are pointers to control blocks. The C call uses the control block address instead of a control block reference (&MYCCB instead of MYCCB).

Code fragments for initializing WebSphere Data Interchange using C might look like the following samples.

SNB-C

```
typedef struct SNB snb;          /*
Provide "snb" data type    */
struct SNB {
    short    zsnbll;                      /* SNB block length        */
    short    zsnbid;                      /* Reserved for future use      */
    char     zsnbeye[8];                  /* SNB block name        */
    char     zsnbname[8];                 /* Service name          */
    long     zsnbndx;                     /* Service index        */
    short    zsnbpc;                      /* Service parameter count      */
    char     zsnbflg0;                    /* First flag byte          */
    char     zsnbflg1;                    /* Second flag byte        */
    void     *zsnbfanc;                   /* First anchor pointer      */
};
```

CCB-C

```
typedef struct CCB ccb;     /*
Provide "ccb" data type    */
struct CCB {

    short    zccbll;                      /* CCB block length        */
    short    zccbid;                      /* Reserved for future use      */
    char     zccbeye[8];                  /* CCB block name        */
    long     zccbrc;                      /* Return code          */
    long     zccberc;                     /* Extended return code        */
    char     zccbsid[8];                  /* System ID          */
    char     zccbuid[8];                  /* User ID          */
```

```
    char    zccbaid[8];                    /* Application ID        */
    char    zccbcid[8];                    /* Error module ID       */
    short   zccbxfid;                      /* Component function ID    */
    char    zccblpid[6];                   /* Language profile ID    */
    long    zccbcpid;                      /* Code page ID          */
    long    zccbrsv;                       /* Reserved for future use */
    void    *zccbccxp;                     /* CCB extension pointer    */
    void    *zccbcabp;                     /* Common area block pointer*/
    char    zccbdbid[4];                   /*  DB2 subsystem ID      */
    char    zccbdbpl[8];                   /*  DB2 plan /AIX alias    */
    char    zccbdbui[8];                   /* AIX DB2 user ID       */
    char    zccbdbpw[18];                  /* AIX DB2 password       */
    char    zccbrsv1[26];                  /* DI internal use only    */
    long    zccbrsv2[117];                 /* DI internal use only    */
};
```

FCB-C

```
typedef struct FCB fcb;             /*Provide "fcb" data type   */
struct FCB {
    short   zfcbll;                        /* FCB block length       */
    short   zfcbfunc;                      /* Service function code     */
};
```

INIT-C

```
snb mysnb;
ccb myccb;
fcb myfcb;
memset(&mysnb,'0',sizeof(mysnb));
memset(&myccb,'0',sizeof(myccb));
memset(&myfcb,'0',sizeof(myfcb));
memcpy(mysnb.zsnbname,"ENVSERV",8);   /* service
wanted    */
mysnb.zsnbpc = 5;    /* parms on fxxzc call */
memcpy(myccb.zccblpid,"ENU  ",6);   /* language
to be used */
myfcb.zfcbfunc = 1;    /* INITIALIZE function */
fxxzc(&mysnb,&myccb,&myfcb,"MYAPPL
 ","SYSTEM  ");
if (!myccb.zccbrc) {
/* initialization worked    */
} else {
/* initialization failed    */
}
```

Assembler calls

When using Assembler, you invoke WebSphere Data Interchange as a normal call to an external function. The call is made to FXXZASM using standard operating system parameter lists, register conventions, and linkage conventions. The register conventions are:

Note on formatting:

Check termwidth

R0	Not defined
R1	Address of a parameter list
R2 to R12	Not defined
R13	Address of a 72–byte area used by the called program to save the registers of the calling program
R14	Return address to the program making the call
R15	Entry point for the routine being called

The parameter list pointed to by Register 1 consists of a series of pointers to the parameters and is coded as follows:

```
R1------------>    +0:    Address of the SNB
   +4:    Address of the CCB
   +8:    Address of the FCB
   +C:    Address of service parm 1
   +10:    Address of service parm 2
   and so on for as many parameters as the service requires
```

Code fragments for initializing WebSphere Data Interchange using Assembler might look like the following samples.

SNB-Assembler

```
SNB     DSECT ,
    DS    0D
ZSNBLL    DS    H    SNB block length
ZSNBID    DS    H    Reserved for future use
ZSNBEYE    DS    CL8    SNB block name
ZSNBNAME    DS    CL8    Service name
ZSNBNDX    DS    F    Service index
ZSNBPC    DS    H    Service parameter count
ZSNBFLG0    DS    CL1    First flag byte
ZSNBFLG1    DS    CL1    Second flag byte
ZSNBFANC    DS    F    First anchor pointer
ZSNBLEN    EQU    *-ZSNBLL    Length of SNB
```

CCB-Assembler

```
CCB     DSECT ,
    DS    0D
ZCCBLL    DS    H    CCB block length
ZCCBID    DS    H    Reserved for future use
ZCCBEYE    DS    CL8    CCB block name
ZCCBRC    DS    F    Return code
ZCCBERC    DS    F    Extended return code
ZCCBSID    DS    CL8    System ID
ZCCBUID    DS    CL8    User ID
ZCCBAID    DS    CL8    Application ID
ZCCBCID    DS    CL8    Error module ID
ZCCBXFID    DS    H    Component function ID
ZCCBLPID    DS    CL6    Language profile ID
ZCCBCPID    DS    F    Code page ID
ZCCBRSV    DS    F    Reserved for future use
ZCCBCCXP    DS    F    CCB extension pointer
```

```
ZCCBCABP    DS    F    Common area block pointer
ZCCBDBID    DS    CL4   DB2 subsystem ID
ZCCBDBPL    DS    CL8   DB2 plan / AIX alias
ZCCBDBUI    DS    CL8   AIX DB2 user ID
ZCCBDBPW    DS    CL18  AIX DB2 password
ZCCBRSV1    DS    CL26  DI internal use only
ZCCBRSV2    DS    117F  DI internal use only
ZCCBLEN     EQU   *-ZCCBLL   Length of CCB
```

FCB-Assembler

```
FCB    DSECT ,
    DS    0D
ZFCBLL    DS    H    FCB block length
ZFCBFUNC    DS    H    Service function code
ZFCBLEN    EQU   *-ZFCBLL   Length of FCB
```

USER-DSECT for initialization sample

```
USERDS    DSECT   ,
SAVAREA    DS    18F
SNBADDR    DS    A
CCBADDR    DS    A
FCBADDR    DS    A
APPADDR    DS    A
SYSADDR    DS    A
SNBAREA    DS    CL(SNBLEN)
CCBAREA    DS    CL(CCBLEN)
FCBAREA    DS    CL(FCBLEN)
```

INIT-Assembler

```
*********************************************************
* It is assumed storage for the USERDS DSECT           *
* has been allocated (initialized with binary zeros)   *
* and is addressable.                                  *
*********************************************************
APPLID    DC    CL8'MYAPPL '
SYSID    DC    CL8'SYSTEM 'LA    13,SAVAREA
LA    6,SNBAREA
LA    7,CCBAREA
LA    8,FCBAREA
USING    SNB,6
USING    CCB,7
USING    FCB,8
MVC    ZSNBNAME(8),=CL8'ENVSERV '
MVC    ZSNBPC(2),=X'0005'
MVC    ZCCBLPID(6),=CL6'ENU  '
MVC    ZFCBFUNC(2),=X'0001'
ST    6,SNBADDR
ST    7,CCBADDR
ST    8,FCBADDR
LA    9,APPLID
ST    9,APPADDR
LA    9,SYSID
ST    9,SYSADDR
```

Chapter 6. Using WebSphere Data Interchange in the z/OS environment   **435**

```
LA    1,SNBADDR
L     15,=V(FXXZASM)
BALR   14,15
ICM    9,15,ZCCBRC
BNZ    INITERR

**************************************************
*   WebSphere Data Interchange initialization successful    *
**************************************************

INITERR EQU    *

**************************************************
*   WebSphere Data Interchange initialization unsuccessful    *
**************************************************
```

API business tasks

No one API service performs an entire business function. To accomplish a business task, you must call a combination of services in a specific order. For example, if you want to write a program that reads an application file, translates the file into an EDI standard format, and sends the data to all trading partners, the following sequence of API service calls is required:

1. Call the environmental service to initialize WebSphere Data Interchange. For detailed information, see "Initializing the environmental API" on page 438.

2. Make repeated calls to the translation service to translate and envelope data at the same time. For detailed information, see "Translate-to-standard API" on page 446.

3. Call the translation service to signal the end of translation. For detailed information, see "End translation/enveloping API" on page 517.

4. Call the communication service to send the transactions. For detailed information, see "Send transactions and restart send transactions API" on page 544.

5. Call the environmental service to terminate WebSphere Data Interchange. For detailed information, see "Terminating the API" on page 441.

You can accomplish all of these calls using the TRANSLATE AND SEND command. The WebSphere Data Interchange utility issues the same API requests that you would issue in an application program.

You must base the decision on whether to use the API or a WebSphere Data Interchange Utility request (PERFORM command) on the control and availability of data rather than availability of WebSphere Data Interchange Utility functions. The WebSphere Data Interchange Utility provides most of the functions available in the API. However, the WebSphere Data Interchange Utility requires that the data be in specific formats (such as C and D records or raw data) and accessed using specific access methods (such as QSAM in z/OS). This method only provides overall result status (JCL condition codes). The following are a few reasons you might consider writing an API program:

- You have more direct control when you write your own API program. At each step, the system provides detailed information about the status of each API request. The WebSphere Data Interchange Utility provides a condition code indicating the most severe error and an audit trail of all errors that occurred. If you want to automatically

process these errors, you could write a program to parse the AUDIT file, but it could become very complex. Instead, you must consider writing a program to request API services directly.

* You might want to synchronize the updating of your data with the results of WebSphere Data Interchange processing. For example, if you are translating, you might want to update your application record to indicate whether the translation was successful, and synchronize changes to your data with the WebSphere Data Interchange databases to verify that the business transaction has been processed.

    Note: In CICS, you can synchronize application data with WebSphere Data Interchange data without writing an API program. For more information, see Chapter 5, "Using WebSphere Data Interchange in the CICS environment," on page 347.

* The application data might not meet the RAWDATA requirements of WebSphere Data Interchange or might be formatted in C and D records. As a result, your data cannot be processed by WebSphere Data Interchange. When you understand the data, you can instruct WebSphere Data Interchange how to perform translation.

* Your application data might not meet the access type requirements of WebSphere Data Interchange (QSAM in z/OS, TS queue or TD queue in CICS). For example, if the application data is stored in DB2 tables, the data must be moved into one of the storage mechanisms mentioned above before the WebSphere Data Interchange utilities can be used.

* The sequence of transactions in interchanges and groups that is created by the WebSphere Data Interchange Utility might not meet your requirements.

The API functions described in the following sections include the equivalent PERFORM commands. After reading the command descriptions, determine whether you need an API program or can use the WebSphere Data Interchange Utility. For more information, see Chapter 1, "Using The Utility," on page 1.

You might need a combination of application-written API programs and WebSphere Data Interchange Utility services. For example, you might want to control data translation but have no need to control enveloping and sending. In this situation, your business process might include:

1. An API program that issues the following API requests:
    a. Environmental service to initialize WebSphere Data Interchange
    b. Translation service to translate data without enveloping
    c. Translation service to signal the end of translation
    d. Environmental service to terminate WebSphere Data Interchange
2. A WebSphere Data Interchange Utility step to use the ENVELOPE AND SEND command, or a two-step WebSphere Data Interchange Utility process to use the ENVELOPE command followed by the SEND command.

Environmental services

Environmental services both establishes and removes the WebSphere Data Interchange environment.

First, your application program must request initialization. When initialization is complete, you can request other services. If you request another service before requesting an initialization, either a return code of **-1** is posted in the CCB, or the program ABENDs.

Finally, your application program must request termination. The services requested between initialization and termination (such as translation or communication services) internally acquire storage, open files, and obtain control over resources. Requesting termination is necessary so WebSphere Data Interchange can release all resources that it has obtained.

Table 114 lists the functions provided by the environmental service.

Table 114. Environmental services functions

Function	Code	Sample Call Statement for Function
Initialize WebSphere Data Interchange	1	`FXXZccc(SNB,CCB,FCB,'applid','sysid')`
Terminate WebSphere Data Interchange	2	`FXXZccc(SNB,CCB,FCB)`

Initializing the environmental API

You must request the initialization function to enable your program to use the WebSphere Data Interchange environment. During request processing, WebSphere Data Interchange confirms that:

- All necessary resources for a WebSphere Data Interchange environment are available, and
- You are authorized to access the system (identified by the SYSID parameter)

WebSphere Data Interchange also verifies that the `language profile ID` (saved in the `ZCCBLPID` field of the CCB before the call) identifies a member defined in the language profile (LANGPROF).

The `APPLID` parameter must identify a member in the application definition profile (APPDEFS) or in the activity log profile (ACTLOGS). If an APPDEFS member does not exist, the following assumptions are made:

- The `APPLID` parameter identifies an ACTLOGS member. If it does not, and logging is necessary for other service requests, errors occur.
- The management reporting functions must be enabled for the current WebSphere Data Interchange session.

The syntax for the Initialization API request is.

`FXXZccc(SNB,CCB,FCB,applid,sysid)`

The same CCB used to initialize the WebSphere Data Interchange session is used for every call issued during the session. The unique parameters for this function request are defined in Table 115.

Table 115. Parameters for the initialization of Environmental services

Parameter	Description

SNB	**ZSNBNAME**
	    ENVSERV
	
	ZSNBPC
	    5 if the system ID is specified
	    4 if the system ID is not specified (CICS).

CCB	**ZCCBLPID**
	    The language profile member to be used for this
	    WebSphere Data Interchange session.

FCB	**ZFCBFUNC** 1
APPLID	The application ID. Must match either a member in the application definition profile or in the activity log profile. 8 bytes, left-justified, and padded with blanks.
SYSID	The system ID. Must match a resource name the security administrator assigned to WebSphere Data Interchange. 8 bytes, left-justified, and padded with blanks. The parameter is optional. The default is **DIENU**. If you do not specify this field, the ZSNBPC field must contain **4**. Does not apply to the CICS environment.

The results of the initialization request are posted in the return code (RC) and extended return code (ERC) fields of the CCB. Valid values are defined in Table 116.

Table 116. Environmental services initialization return codes

RC	Explanation
0	Initialization was successful.

Table 116. Environmental services initialization return codes (continued)

RC	Explanation	
4	Initialization failed. Extended return codes are:	
	4	No service table defined (failure to locate FXXZIN load module).
	8	Insufficient virtual storage to load programs and tables.
	12	Failure initializing the EDIT service. This might be caused by an incorrect ZCCBLPID value (one that cannot be matched with a LANGPROF member). In DB2 installations, this error can occur because of the inability to access the profile or translation/validation tables (in many cases, this could be a DB2 timestamp error, meaning that the timestamp in load module EDIPSMD is different from its corresponding DBRM timestamp). If a DB2 timestamp error is suspected (SQL code -818), you must rebind the DB2 plan. Besides EDIPSMD, there are two other WebSphere Data Interchange DB2 load modules: EDIRPML and (for CICS users) EDICRIN, which could generate DB2 timestamp problems if out of sync.
	16	A WebSphere Data Interchange session for the same user is already active.
	1024	Access to the system is denied.

Utility service API

The utility service API is used when implementing HOT-DI.

The syntax for the utility function request call is:

```
FXXZccc(SNB,CCB,FCB,UTILCB)
```

The unique parameters for this function request are defined in Table 117.

Table 117. Environmental services utility service parameters

Parameter	Description	
SNB	**ZSNBNAME**	
		UTILSRV
	ZSNBPC	
		4
CCB	The common control block that was used to initialize WebSphere Data Interchange.	
FCB	ZFCBFUNC	
	1	Normal execution
	2	HOT WebSphere Data Interchange mode
UTILCB	The utility service control block.	

Terminating the API

You must request the termination function for WebSphere Data Interchange to perform housekeeping activities for cleanup of all system resources acquired when processing the API service requests. The cleanup includes releasing virtual storage, closing files, and releasing locks on resources.

If an error occurs during termination, WebSphere Data Interchange returns a value other than zero in the return code field (ZCCBRC). If this occurs, your application program must request the termination function again and continue to request it until the function completes successfully. A termination failure indicates that a file could not be closed, or that storage could not be released. When this happens, the WebSphere Data Interchange component for which the error occurred attempts to log the error.

After the error occurs, WebSphere Data Interchange returns control to your program so that your program can request termination again to complete the task. WebSphere Data Interchange does not attempt to call the program where the error occurred again, but continues to call other programs that are used during cleanup.

The syntax for the termination function request is:

`FXXZ`*ccc*`(SNB,CCB,FCB)`

Environmental services

The unique parameters required for this function request are defined in Table 118

Table 118. Environmental services termination parameters

Parameter	Description
SNB	**ZSNBNAME** ENVSERV **ZSNBPC** 3
FCB	**ZFCBFUNC** 2

The results of the termination request are posted in the return code and extended return code fields of the CCB. Valid values are defined in Table 119.

Table 119. Environmental services termination return codes

RC	Explanation
0	Termination was successful.
>0	Termination failed; try again.
<0	An invalid CCB address, or an incorrect name in the ZSNBNAME field.

Translation services

Translation services, enveloping services, and data extraction services are closely related. They share an API implemented by the same logical service (TRANPROC). These services also share the Transaction Store.

Exchanging information with a trading partner is a sequence of interruptible events whose status must be maintained and tracked. The processes used to exchange data fall into two categories:
- A batch-oriented process (z/OS)
- A time-critical process (CICS

In the batch-oriented process, application data is stored in files owned by the applications until the next batch job runs, which translates the application data into an EDI standard format. This job, or a different job run later, can request that all the EDI standard data be gathered (enveloped) and sent to the trading partner. The trading partners receive the data, process the data, and then return a response.

In the time-critical process, application data is usually translated, enveloped, and sent as soon as the data is available. Although some batching of data might occur, the batches are typically much smaller than in the batch-oriented process. The trading partner is expected to process the data and generate a response immediately.

The Transaction Store database saves all EDI standard data sent or received and tracks the status of the data as it progresses from translated, to send, to received, to acknowledged. The Transaction Store is updated by all translation and enveloping operations, and is maintained and reported on by the Transaction Store Facility or the WebSphere Data Interchange Utility. Updating the Transaction Store database is optional and can be controlled using the Application Defaults (APPDEFS) profile. For more information about the Application Defaults profile, refer to the WebSphere Data Interchange User's Guide.

The Transaction Store database consists of seven DB2 tables, described in Table 120 on page 444.

Table 120. Translation services DB2 tables

Table	Description
EDIVTSTH	The transaction handle table. This is the primary table in the Transaction Store and is referenced by all other tables. An entry is created whenever a translation occurs and a transaction is deenveloped. As transactions are added to the Transaction Store, they are assigned a unique key value called a transaction handle (THANDLE). The format of the transaction handle is: `YYYYMMDDHHMMSSxxnnnn`, where:
	YYYYMMDD The date of the first transaction for the translation session.
	HHMMSS The time of the first transaction for the translation session.
	xx A random number assigned by WebSphere Data Interchange. Valid values are **00 - 99**.
	nnnn A sequential number assigned by WebSphere Data Interchange. Valid values are **0 - 9999**. If more than 9999 transactions are translated in the same session, WebSphere Data Interchange receives a new date, new time, and new random number, and starts the sequence at **0** again.
	This key value is communicated between application programs and WebSphere Data Interchange API services through the TSKEY and TSKEYU fields in the TRCB.
EDIVTSTI	The EDI standard transaction image table. Contains the EDI standard transaction image and is created whenever a translation occurs or a deenvelope takes place.
EDIVTSTO	The transaction override table. Contains override values for fields in the service segments. A transaction override table is created only when translation occurs and overrides are supplied for service segment fields. These values are captured at translation time and used when the transaction is enveloped.
EDIVTSEV	Contains information about an entire interchange, such as the network status, the date and time created, and the date and time sent. A table entry is created during enveloping or deenveloping for each interchange header created or deenveloped.
EDIVTSGP	Contains information about a group of transactions in an interchange, such as the functional acknowledgment status of the group. A table entry is created during the enveloping or deenveloping process. At least one table entry is created, even when the interchange does not contain a functional group.
EDIVTSTU	Contains information about a transaction in an interchange or group, such as the date and time the transaction was enveloped, and the acknowledgment status of the transaction. A table entry is created during the enveloping or deenveloping process. A transaction that has been enveloped more than once (REENVELOPED) will have entries in the EDIVTSTU, EDIVTSGP and EDIVTSEV tables.

Table 120. Translation services DB2 tables (continued)

Table	Description
EDIVTSA	The application usage table. Contains information about the processing of a transaction by an application. A table entry is created whenever a translation for a transaction is attempted. A transaction that has been translated more than once (RETRANSLATE) will have multiple entries in this table. A transaction that has never been translated (DEENVELOPED only) will have no entries in this table.

The syntax for the translation function request call is:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The logical name for the translation service is TRANPROC. The codes and functions provided by the translation service are:

111   Test translate-to-EDI-standard
131   Production translate-to-EDI-standard
211   Test deenvelope and translate-to-application
212   Production deenvelope and translate-to-application
213   Translate a specific transaction to application
1000   End translation

Translation service functions

The two primary functions of the translation services are:

* Translate-to-EDI-standard

  This service transforms data from the application-defined format into the EDI standard defined format as defined by a map. See "Translate-to-standard API" on page 446 for more information.

* Translate-to-application

  This service transforms data in the EDI standard defined format into the application defined format as defined by a map. See "Translate-file-to-application API" on page 486 for more information.

The following five minor functions support the two primary functions:

* End translation

  Signals that there is no more data to process. See "End translation/enveloping API" on page 517 for more information.

* Retrieve interchange header

  Retrieves the EDI standard image of the current interchange, and is used by the WebSphere Data Interchange Utility to create the optional **E** record. See "Retrieve interchange header API" on page 533 for more information.

* Retrieve group header

  Retrieves the EDI standard image of the current group, and is used by the WebSphere Data Interchange Utility to create the optional **G** record. See "Retrieve group header API" on page 533 for more information.

Translation services

- Retrieve transaction

  Retrieves the EDI standard image of the current transaction, and is used by the WebSphere Data Interchange Utility to create the optional **T** record. See "Retrieve transaction header API" on page 534 for more information.
- Close and queue interchange

  Forces an interchange to be completed and written to a file associated with the network. See "Retrieve transaction header API" on page 534 for more information.

  Note: The transform function for data transformation maps is not currently supported using the WebSphere Data Interchange API.

A number of headings in this section include a two-letter abbreviation surrounded by parentheses to make the section headings unique. The two-character abbreviations and their meanings are:

DE   Deenvelope
EV   Envelope
TA   Translate to application
TF   Translate file
TS   Translate to standard

Translate-to-standard API

The translate-to-standard API transforms data from the application format into an EDI standard format as defined by a map. You select these values through the options on the Administrator's Menu.

The WebSphere Data Interchange Utility uses this API internally when you issue any of the following PERFORM commands:

- TRANSLATE TO STANDARD
- TRANSLATE AND ENVELOPE
- TRANSLATE AND SEND

The enveloping, sending, and testing considerations to be aware of when using the translate-to-standard API are described in the following topics.

Enveloping and sending

The logical name for the enveloping service is TRANPROC. This service is described in "Environmental services" on page 438. The following codes and functions are provided by the enveloping service:

1   Retrieve interchange header
2   Retrieve group header
3   Retrieve transaction header
214 Deenvelope transaction
215 Enveloper transaction
990 Close and QUEUE the current interchange
991 Isue database COMMIT

The syntax for the enveloping and sending service is:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The following three API functions allow you to translate, envelope, and send a file to a trading partner:

- The translation service (function code **131**)translates the data and updates the Transaction Store database with the transaction information and an image of the EDI standard data produced. For more information, see "Translate-to-standard API" on page 446.
- The enveloping service (function code **215**)builds the proper service segments and retrieves the transaction image from the Transaction Store. It invokes the communication service with function code **110** which writes the interchange to a file associated with the network. For more information, see "Envelope API" on page 503.
- The communication service (function code **211**) builds the proper commands for the network, and invokes the network to send the file to your trading partner. For more information, see "Send transactions and restart send transactions API" on page 544.

You can shorten this sequence by using the translate-to-standard function of the translation service to envelope a transaction when the transaction is translated. To do this, use the TRANSLATE AND ENVELOPE command as illustrated in "Translate, envelope, and send process" on page 449.

You can control whether enveloping is done at the time of translation with the ENVLDELAY field in the TCB. You can only delay enveloping for translated transactions already stored in the Transaction Store database. For more information, refer to the Application Defaults profile description in the WebSphere Data Interchange User's Guide. For more information about the ENVLDELAY field, see "Translator Control Block (TRCB)" on page 642.

Consider the following points when performing automatic enveloping:

- All data involved in a translation, including the image of the EDI standard data produced, is saved in the Transaction Store. This occurs whether enveloping is part of the translation or is delayed. An EDI standard image is saved even when the translation is not successful, because the image might prove helpful in determining why the translation failed.
- If enveloping is done separately from translation, the enveloping service must be notified which transaction to envelope. This is done through the unique key value assigned to the transaction when it is added to the Transaction Store. The key is stored in the database as a 10-byte packed value and is returned in packed format in the TSKEY field of the translator control block. It is also returned as a 20-byte character value in the TSKEYU field.

When an envelope function is requested, either the TSKEY or TSKEYU value must be accessible to the enveloping service. The program that issues the translate-to-standard function can save the TSKEY values in a file which the program that issues the envelope calls can read to get the TSKEY values. To organize the data the way you want it enveloped, store the TSKEY values in a file with other data that can be sorted.

Translation services

If only the TSKEY values are necessary, you can use the WebSphere Data Interchange Utility's QUERY command to extract the TSKEY values that you want enveloped. The QUERY command writes the TSKEY values sequentially into the file identified by the EDIQUERY ddname.

- The enveloping service, whether invoked separately, or invoked automatically by setting ENVLDELAY to **N**, invokes the communications service to write the EDI standard data to a file associated with the network. Your program does not need to do this.

- No single API function is the equivalent of the WebSphere Data Interchange Utility's TRANSLATE AND SEND command. Using the API, translating and enveloping is one step, and sending the data is another.

Translate, envelope, and send process

Application program

WebSphere Data Interchange

```
┌── Perform │ API initialization │
│
│   Perform │ First call of session (TS) │
│
├── Do while 'application data exists'
│   ├── If 'first record of transaction'
│   │
│   │       Perform │ First call of transaction (TS) │ ◄──────────  New interchange?
│   │
│   │   ├── If TRCB.TRABORT EQ 'Y'
│   │   │       EXIT
│   │   └── ENDIF
│   │   ├── If TRCB.EJECT EQ 'Q'
│   │   │       Perform │ API-Send transaction │ ◄──────────  Network program
│   │   └── ENDIF                                              called by
│   │   ├── If TRCB.EJECT EQ 'E'                               WebSphere
│   │   │                                                      Data Interchange
│   │   │       Perform │ Interchange error processing │
│   │   │
│   │   └── ENDIF
│   ├── ELSE
│   │       Perform │ Subsequent calls │
│   └── ENDIF
│   ├── If 'last record of transaction'
│   │
│   │       Perform │ Last call of error transaction (TS) │ ◄──────  Translate data
│   │
│   │       ┌── If TRCB.TRABORT EQ 'Y'
│   │       │       EXIT
│   │       └── ENDIF
│   └── ENDIF
│   ├── If TRCB.TRACCEPT NE 'Y'
│   │       Perform │ Skip to next transaction │
│   └── ENDIF
└── ENDDO
        Perform │ API-End Translation / Enveloping │ ◄──►  A
├── If TRCB.EJECT EQ 'Q'
│       Perform │ API-Send transaction │ ◄──►  B
└── ENDIF
├── If TRCB.EJECT EQ 'E'
│       Perform │ Interchange error processing │
└── ENDIF
└── Perform │ API termination │
```

New interchange? — yes — Write to "QDATA"
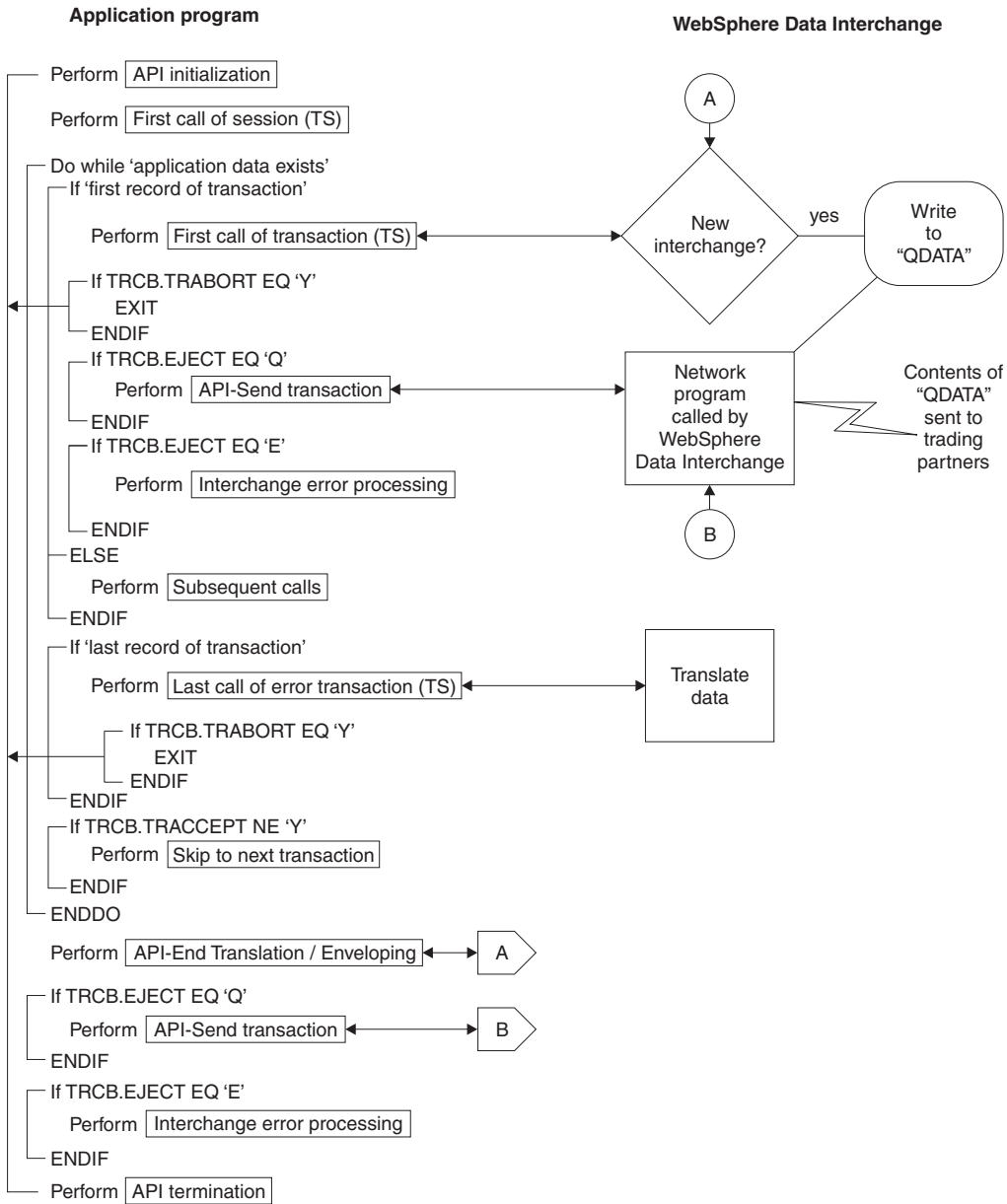
Contents of "QDATA" sent to trading partners

A

B

Figure 23. Translate, envelope, and send process

Test translate-to-standard

There are two methods for testing your transactions:

1. You can use test function code **111** to translate your transactions to EDI standard format in test mode. Your program follows the same steps as in production mode. Translation occurs and the interchange created is written to the file associated with the network, but no information is posted to the Transaction Store. To post information to the Transaction Store, you must use the method described below. You can use the results of the translation for receiving transactions in test mode.

   Note: This form of testing is available only through the API.

   When you use this method, it is assumed that:

   - You require a test usage/rule, if one is available, and the TEST field in the TRCB is set to **T**

   - You want enveloping to occur so that an EDI standard transaction image will be created and written to a file that you can inspect

   If your test translation is followed by a call to send the data, make sure to define a destination file that is in no danger of being sent accidentally. The EDI standard data produced is written to ddname EDITEST. Communications ignores any request to send data when the ddname of the file to be sent is EDITEST. In the CICS environment, EDITEST is a TS queue.

   You can verify the results of your program's efforts by examining the resulting data and the information recorded in the event log. To view or print the event log, choose EVENT LOGGING from the Administrator's Menu.

2. You can use the test usage indicator to inform trading partners that a transaction you are sending is for test purposes only. When you set the Test usage field on the Send Usage panel to **T**, translation and communications occur normally (including updating the Transaction Store) except that the envelope carries an indicator that the data is for testing. You might prefer this method because you can test the entire path from you to your trading partner, including any returned network or functional acknowledgments.

   To use this method with the WebSphere Data Interchange Utility, do one of the following:

   - Set the test indicator flag in the C record to **T** or **U** to indicate that this is a test usage/rule.

   - Set the RAWUSAGE keyword on the PERFORM command to **T** to indicate that test mode is being used with RAWDATA.

   - If you use the API, set the TEST field in the TRCB to **T** or **U**.

     Note: Only X12 and EDIFACT envelope types have a test-or-production flag defined. WebSphere Data Interchange does not mix test transactions with production transactions in the same interchange.

Translate-to-standard data modes

Multiple calls to the translator are usually required to provide the translator with the data you want translated. The final call confirms that all the data has been provided and the translation must begin.

The two modes of operation, based on the type of data provided for the translator, are:

*   Multiple unit of work mode

    In this mode, more than one application record contributes to the translation. For example, a purchase order might include a header record, many detail records, and a trailer record. With each call to the translator, you provide a single record that corresponds to a structure in the application data definition, and all subordinate structures that have not been passed separately. The data is stored in buffer files until all records for the transaction have been provided.

    Use the ATSID field to tell the translator the name of the structure provided in the TRIDB buffer. When all the data has been provided, call the translator again with an EJECT field of **Y**. This tells the translator that the translation must begin. The translator must have all the data before any translation can take place.

    Note: No data is provided when EJECT is set to **Y**. This setting indicates that all data has been provided.

*   RAWDATA mod e

    In multiple unit of work modes, the application knows the type of data being processed and communicates that information to the translator through the ATSID field. Use the RAWDATA mode only if the application does not know the type of data being processed. For example, if you are writing a general-purpose utility program, your program might not be customized to process all the different data formats you might encounter.

    RAWDATA mode reverses the roles of the application program and the translator. The translator returns the structure name to the application, and tells the application when it is time to translate using the value in the TRNSTAT field.

    For more information about RAWDATA processing, see "Special considerations (TS)" on page 466.

Translation special considerations

Many calls can be made to the translator during a translate-to-standard session, but the following require special attention:

*   The first call to the translator after a WebSphere Data Interchange initialization or after a translation service termination. This call establishes the default values that apply from the first call until a translator termination call is received.
*   The first call to the translator for a transaction. Key information concerning the transaction is provided, and the translator determines the relationship that this transaction has with the previous transaction.
*   The final call to the translator for a transaction. Translation from the application format to the EDI standard format actually takes place.
*   The final call to the translator to terminate the session. Cleanup is done and final processing occurs.

All other calls merely involve the movement of data from the application buffers into the translator buffers. The following topics describe these four critical calls in detail.

Translate-to-standard API

You can use the two following function codes to request a translate-to-standard function:

111   Requests translation in test mode
131   Requests translation in production mode

You do not need to change your program to switch from test mode to production mode. You only need to change the function code used to invoke the services. Test mode is different from production mode in the following ways:

- WebSphere Data Interchange forces the `TEST` field in the TRCB to contain a value of **T**. If a test transaction exists, it uses the test transaction when testing your program. If a test transaction does not exist, it uses the production transaction.

- WebSphere Data Interchange forces the `ENVLDELAY` field in the TRCB to contain a value of **N**. This writes the EDI standard data produced to a file.

- WebSphere Data Interchange forces the `FILEID` field in the TRCB to contain a value of **EDITEST**. If a communications service request to send transaction data is received and the file name used is EDITEST, WebSphere Data Interchange ignores the request.

- Control numbers are not taken from the trading partner profile member. Instead, they are assigned values of T*nnnnn*, where *nnnnn* is a sequential value starting with 00001.

- WebSphere Data Interchange automatically logs the EDI standard image that is produced and the application data that is received.

- WebSphere Data Interchange does not write any data to the Transaction Store, to prevent cluttering up the database with test data.

- Functional acknowledgments are not generated even if requested in the usage/rule record.

- Statistics maintained by the Management Reporting component of WebSphere Data Interchange are not updated.

The basic format of the translate-to-standard request is:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for this API request are defined in Table 121 on page 453.

Note for accessibility
Following table has spanned cols.

Table 121. Transaction services translate-to-standard parameters

Parameter	Description
SNB	**ZSNBLL** 32 **ZSNBNAME** TRANPROC **ZSNBPC** 6
FCB	ZFCBFUNC **131** production **111** test
TRCB	The translator control block. For more information about this block, see "Translator Control Block (TRCB)" on page 642.
TRIDB	The input data block. This block contains the application data to be translated. The format of the data in this block must match the format of the data defined in the data format. Minimum size is 32000 bytes.
TRODB	The output data block. WebSphere Data Interchange uses the output data block as a work buffer. Minimum size is 32000 bytes. Maximum size must be the size of the largest EDI standard segment that is produced, excluding the BIN segment.

First call of session (TS)

There are numerous fields in the control blocks defined for the translate-to-standard API function whose values only need to be established once. These values can be established before the first call, and they do not have to be refreshed or changed before any other call. Because the first call for a session also qualifies as the first call for a transaction, you must also follow the instructions in the next section, "First call for transaction (TS)." The control blocks, the fields within the control blocks, and the initialization considerations are described in Table 122, Table 123, Table 124 on page 454, and Table 125 on page 456.

Table 122. Service name block (SNB) initialization for translate-to-standard

Field name	Initialization
ZSNBLL	**32**
ZSNBNAME	**TRANPROC**
ZSNBPC	**6** (all calls for translation services have six parameters)

Table 123. Function code block (FCB) initialization for translate-to-standard

Field name	Initialization
ZFCBLL	**4**
ZFCBFUNC	**131** (production) **111** (test)

Table 124. Translator control block (TRCB) initialization for translate-to-standard

Field name	Initialization
BLKLEN	**1536**.
BLKNME	**EDITRCB.**
BLKTYPE	The format for the TRIDB and TRODB buffers. **H** Unlimited size **(other)** Limited to 32768 bytes
FASPEC	Indicates that **FUNACKFLE** is being used. Set this field to **Y** (If not provided, the value of **FUNACKFLE** will be ignored).
XPANDED	**Y**. Indicates that WebSphere Data Interchange must check the `BLKLEN` field to determine the software version and release being used.
ENVLDELAY	Indicates whether enveloping must be delayed until translation is complete. **Y** Enveloping does not occur at the same time as translation. **(other)** Enveloping occurs at the same time as translation.
SCOPE	Indicates that the point at which WebSphere Data Interchange issues a database COMMIT. **E** Interchange (envelope) level recovery. Applies only when enveloping occurs. **(other)** Transaction level recovery. **Note:** The value set in this field affects the recovery scope used during the session. For more information, see "Send Recovery Scope" on page 469.
INMEMTRANS	Indicates the maximum number of transactions that must be maintained in virtual storage before any database updates are attempted. Applies only if the value of `SCOPE` is **E**. The database is updated when either the value in this field, or the end of the interchange, is reached. This value is important in an environment where multiple application programs request translation services concurrently. The higher you set the value for INMEMTRANS, the more concurrency can be achieved. For more information, see "INMEMTRANS" on page 663. **Note:** The value set in this field affects the recovery scope used during the session. For more information, see "Send Recovery Scope" on page 469.
FILEID	For enveloping, the ddname of the file where the transaction data is written when an interchange is complete. If you do not specify a ddname, the ddname specified in the `Trans data queue` field of the network profile is used. This value does not have to be a constant for the entire session, but if specified, it must be set before an interchange is complete.

Table 124. Translator control block (TRCB) initialization for translate-to-standard (continued)

Field name	Initialization
ITPBREAK	Indicates whether a change in the internal trading partner ID (vendor number) starts a new interchange. You can use this field to create separate interchanges for the individual vendors associated with a trading partner. **Y** Starts a new interchange each time the internal trading partner ID changes. The current interchange must be closed and written. **N or (other)** Starts a new interchange only when the trading partner nickname changes. **N** is the recommended value.
BATCHID	The batch ID for a group of transactions. The Transaction Store creates an alternate key value using the batch ID. Other than using the transaction handle, searching on batch ID is the quickest way to retrieve a transaction from the Transaction Store during the selection process. If you do not specify a value for this field, the default value `DDHHMMSS` is used, where `DD` is the current day of the month, and `HHMMSS` is the current time. For more information, see "Batches and Bundles" on page 471. **Note:** This field is checked with each call to the translator. The value in this field when the last call for transaction (TS) is made will be associated with the transaction.
TRXLIFE	The amount of time a transaction remains in the Transaction Store before it is eligible for purging. Default is **30** days. **Note:** This field is checked with each call to the translator. The value in this field when the last call for transaction (TS) is made will be associated with the transaction.
IMGLIFE	The amount of time a transaction's image (the standard data produced) remains in the Transaction Store. Default is 30 days. **Note:** This field is not currently supported.
ENVLDATE	The earliest date that a transaction is considered eligible for enveloping. Applies only when delayed enveloping is used (`ENVLDELAY` is set to **Y**). If you do not specify a value in this field, the transaction is considered eligible for enveloping immediately. **Note:** This field is checked with each call to the translator. The value in this field when the last call for transaction (TS) is made will be associated with the transaction.
ERRFILTER	Indicates which error codes to filter out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more information, see "Error filtering" on page 22.
RAWDATAOUT	Indicates whether raw data output is desired for fixed-to-fixed mappings. **Y** Uses raw data output for fixed-to-fixed mappings **(other)** Uses C and D record output for fixed-to-fixed mappings

Table 124. Translator control block (TRCB) initialization for translate-to-standard (continued)

Field name	Initialization
FFILEID	The ddname of the file where the translated data for fixed-to-fixed translation is written. If you do not specify a value in this field, the ddname is formed from the concatenation of the `Application file name` from the target data format and the `File suffix` from the trading partner profile. This value does not have to be a constant for the entire session. However, if you want to use this field, it must be set before an interchange is completed.

Table 125. Translator output data block (TRODB) initialization for translate-to-standard

Field name	Initialization
BLKLEN	Set this field to the size of the data block, including the `BLKLEN` field. The minimum value for this field is 32000 bytes.
RESERVED	Zeros.

First call for transaction (TS)

On the first call to the translator for a transaction, you must provide the information required to locate the map that directs the transformation from application data to EDI standard data. Your EDI administrator creates maps for specific data format definitions. The `ATFID` field contains the definition name for which the map was defined and for which data is being provided. However, many different users can use a map. Your EDI administrator defines these users by adding trading partner send usage/rule entries to the map.

The fields for the trading partner send usage/rule are defined in

Table 126. Trading partner send usage/rule fields

Field name	Description
INTPID	The internal trading partner ID. This is the name of the trading partner as known to your application program, such as a vendor number or account name.
TEST	Indicates the type of transaction being processed. Valid values are:
	I This is an information transaction. Use an information usage/rule, if one is found. If an information usage/rule is not found, a production usage/rule is used instead. Even when a production usage/rule is used, the transaction is flagged as an information transaction.
	P This is a production transaction. Use only a production usage/rule (default).
	T This is a test transaction. Use a test usage/rule, if one is found. If a test usage/rule is not found, a production usage/rule is used instead. Even when a production usage/rule is used, the transaction is flagged as a test transaction.
	U The translator must determine if the transaction is test, information, or production based on the usage/rule found. If a test usage/rule is found, the transaction is processed as a test transaction, and a value of **T** is returned. If an information usage/rule is found, the transaction is processed as an information transaction, and a value of **I** is returned. If only a production usage/rule is found, the transaction is processed as a production transaction, and a value of **P** is returned.

WebSphere Data Interchange requires these fields to determine what translation is being requested. The TRCB fields described in Table 127 must be set by the application on the first call for translation.

Table 127. TRCB fields that you must set on the first call for translation

Field name	Initialization
ATFID	The data format ID for which data is being provided.
INTPID	The internal trading partner ID (such as vendor number or account name) for which the translated EDI standard data is destined.
TEST	Indicates the type of transaction being processed. **I** Information transaction **P** Production transaction **T** Test transaction **U** Transaction type determined by existing, active usage/rule

Table 127. TRCB fields that you must set on the first call for translation (continued)

Field name	Initialization
RAWDATA	Indicates whether the RAWDATA interface must be used. For more information on the RAWDATA interface, see "Send raw data" on page 466. **Y** Uses the RAWDATA interface **N or (other)** Does not use the RAWDATA interface
EJECT	Indicates whether all data for this structure has been received. **X** A partial structure has been provided. For more information on partial structures, see "Providing a Partial Structure" on page 469. **Y** All application data has been provided and translation must begin. For more information on this field, see "Translate-to-standard API" on page 446. For more information on the effects of the EJECT field settings, see "Last call for transaction (TS)" on page 463.
ATSID	The name of the structure for which data is being provided in the TRIDB.
BNDLFLAG	Indicates whether a bundle must be started or ended. The first transaction of a cluster is called the controlling transaction and is the transaction displayed when you use the Transaction Store Facility. WebSphere Data Interchange uses the transaction handle (THANDLE) value of the controlling transaction to associate all of the transactions in a cluster. The THANDLE value is returned in the TSKEY and TSKEYU fields. For more information, see "Batches and Bundles" on page 471. **Y** Starts a new bundle with this transaction. **N** Ends the bundle. **(blank)** Does not change bundling status. If a bundle is active, this transaction is considered part of it.
HOLDFLAG	Indicates whether this transaction is to be placed on hold when added to the Transaction Store. A transaction in hold status is not available for any other activity, such as enveloping. You must release a held transaction before it can be processed. **Y** Places this transaction in Held status. **N or (othre)** Does not place this transaction in Held status.
ROUTCODE	A three-character generic routing code provided by the application and used by WebSphere Data Interchange to select a generic send usage/rule. A blank indicates a default generic send usage/rule.

When a transaction gets enveloped, the envelope function builds the service segments that surround a transaction. These service segments consist of an interchange header that identifies the sender and the receiver, a group header that gathers all transactions of similar characteristics and can be used to identify the application sender and receiver, and a transaction header that provides the name of the transaction being sent. Most of the values for fields in these segments are established by your EDI administrator, but your program can override certain fields by providing values that you want associated with a transaction. These values must be provided with the transaction at the time of translation so that WebSphere Data Interchange can use them when the transaction is enveloped.

Table 128 describes the TRCB fields that you can use to provide overrides. If you do not use overrides, make sure these fields are blank before making the first request for a transaction. These fields are also used as return fields, so make sure your program sets the desired values before making each first request. For more information on service segments, see "Interchange layer" on page 501.

Table 128. TRCB fields that provide overrides

Field name	Overrides:
ISYNTAXID	The interchange syntax ID in the interchange header for envelope types **E** and **T**.
ISYNTAXVER	The interchange syntax version in the interchange header for envelope types **E** and **T**.
ISIDQUAL	The interchange sender ID qualifier in the interchange header for envelope types **E**, **I**, and **X**.
ISID	The interchange sender ID in the interchange header for data type **IS**.
ISENDNAME	The interchange sender name in the interchange header for envelope types **U** and **T**.
IREVROUT	The interchange reverse routing in the interchange header for envelopes type **E**.
IRIDQUAL	The interchange receiver ID qualifier in the interchange header for envelope types **E**, **I**, and **X**.
IRID	The interchange receiver ID in the interchange header for data type **IR**.
RECVNAME	The interchange receiver name in the interchange header for envelope types **U** and **T**.
IROUTEADDR	The interchange routing address in the interchange header for envelope type **E**.
IVERREL	The interchange version and release in the interchange header for data types **LV** or **VR**.
ISPW	The interchange password in the interchange header field for data type **PW**.
IAPREF	The application reference in the interchange header for data type **AP**.
ISTDID	The interchange standard ID in the interchange header for envelope type **I** and **X**.
IPRIOR	The interchange priority code in the interchange header for envelope types **E** and **T**.

Table 128. TRCB fields that provide overrides (continued)

Field name	Overrides:
ICOMMAGREE	The interchange communication agreement in the interchange header for envelope type **E** and **T**.
GSIDQUAL	The group sender ID qualifier in the group header for envelope group type **E**.
GSID	The group sender ID in the group header for data type **AS**.
GRID	The group receiver ID in the group header for data type **AR**.
GRIDQUAL	The group receiver ID qualifier in the group header for envelope group type **E**.
GAPW	The group password in the group header for data type **PW**.
GVER	The group version in the group header for data type **VR**.
GREL	The group release in the group header for data type **LV**.
GRESPAGENCY	The group responsible agency code in the group header for envelope types **E**, **U**, and **X**.
TVER	The transaction version in the transaction header for data type **VR**.
TREL	The transaction release in the transaction header for data type **LV**.

Table 129. TRIDB Initialization - First Call for Transaction

Field name	Initialization
BLKLEN	The length of the TRIDB, including this field. If you are using the same TRIDB for all calls, initialize this field only once.
RESERVED	Zeros. If you are using the same TRIDB for all calls, initialize this field only once.
DATALEN	Set this field to the number of bytes contained in the DATA field. This number must match the number of bytes defined for the structure in the data format.
DATA	The application data. The format of the data must match the format of the structure (in the `ATSID` field) defined in the data format.

When initialization is complete, the translator is invoked with the following API request:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

When the translator receives this request, it verifies that everything it needs to translate the transaction is available. The `ATFID`, `INTPID`, and `TEST` fields are used to locate the trading partner usages/rules and map, and retrieve the control string generated from the map along with the EDI standard and transaction descriptions.

Numerous errors can occur at this point. If an error occurs, the return code and extended return code for the error are posted in the `ZCCBRC` and `ZCCBERC` fields of the CCB.

The TRCB fields that indicate the status of the transaction and of the translator are defined in Table 130 on page 461.

Table 130. TRCB fields that provide transaction and translator status

Field name	Description
TRXACCEPT	Indicates whether the transaction is acceptable and can be translated.
	If the value is not **Y** and your program continues, the next call made to the translator is considered the first call for the next transaction.
	Y The transaction is acceptable and can be translated.
	(other) The transaction is not acceptable and cannot be translated.
TRABORT	Indicates whether the error is so severe that the translator has stopped processing.
	If this value is not **Y** and your program continues processing, the next call made to the translator is considered the first call for the session.
	Y The translator has stopped processing.
	(other) The translator continues processing.

The TRCB fields that are returned on the first call for a transaction and that identify the key attributes of the transaction are defined in Table 131.

Table 131. Fields in TRCB returned on first call for a transaction

Field name	Description
TRNID	The EDI standard transaction or message ID that is generated from this application data.
TEST	If you used a TEST value of **U**, this becomes an output field that indicates whether a production, test, or information usage/rule is being used.
ENVTYPE	Indicates the envelope type that is used when this transaction is enveloped. Valid values are:
	E UNB/UNZ
	I ICS/ICE
	T STX/END
	U BG/EG
	X ISA/IEA
MAPKEY	The map used to translate the application data.
TPNICK	The trading partner nickname associated with the transaction.

Enveloping During Translation:: If enveloping is performed at the same time as translation (ENVLDELAY value of **N**), the translator checks the first call to determine if the transaction just provided is placed into the current interchange, or if the transaction starts a new interchange. If a new interchange must be started, the current interchange is completed and written to the file associated with the network or to the file identified in the FILEID field. See "Fields that cause a new interchange to start" on page 515 for a description of the fields that cause a new interchange to be started.

The TRCB fields described in Table 132 on page 462 are returned when an interchange is written. The fields provide information about the interchange, the network, and the file receiving the interchange.

Table 132. Fields in TRCB returned when an interchange is written

Field name	Description
EJECT	Indicates whether an interchange was written to the file associated with the network. **Q** The interchange was written successfully. **E** The interchange was not written successfully. **Note:** When this field has a value of **E**, the QRC and QERC fields contain values indicating the error. The specific error is logged by WebSphere Data Interchange Communication services. The most likely error is that the ddname could not be opened.
QRC	The return code associated with the error.
QERC	The extended return code associated with the error.
DSNAME	The physical data set name to which the interchange was written. In CICS, this field has the same value as QDDNAME. This is the name of the TS queue containing the interchange.
QNETID	The name of the network associated with the trading partner.
QPTTOPT	Indicates whether the network identified by QNETID is a point-to-point network. **Y** The network is a point-to-point network. **(other)** The network is not a point-to-point network.
QSRPGM	Indicates whether the network identified by QNETID has a network send/receive program associated with it. If the network does not have an associated send/receive program, there is no need to send the interchange. A network without a send/receive program can be used for creating interchanges that are sent to a trading partner by other means. **Y** A send/receive program is associated with this network. **(other)** No send/receive program is associated with this network.
QDDNAME	The ddname to which the interchange is written. This field is blank if a point-to-point network is used. In CICS, this is the name of the TS queue to which the interchange was written.
QTPNICK	The trading partner nickname for which the interchange was built.
QSIZE	The total number of bytes in the interchange, stored as a 4-byte binary value.
QBT	The character representation of QSIZE.

See "Sending transaction data" on page 514 for an explanation of items to consider if your application program is using the Communication services send API for sending the data that was just written to the file. See "Send transactions and restart send transactions API" on page 544 for more information on APIs.

Subsequent calls

Calls made to the translator between the first and last calls transfer transaction data between the application program and the translator. The application data is placed in the TRIDB. The structure name describing the data is placed in the ATSID field. When the call is made, the translator copies the data from the TRIDB into the translator's internal buffers. With each call, your program must check the TRXACCEPT and TRABORT fields for error information.

When all the data has been transmitted, a final call is necessary to tell the translator to translate the data into the EDI standard format. For more information about final calls, see "Last call for transaction (TS)."

Last call for transaction (TS)

The last call for a transaction is indicated by an `EJECT` field set to **Y**. At this point, you can also set the other fields if you do not want the values from this transaction to be used in the first call for the next transaction. The fields you can set are:

- `BATCHID`
- `TRXLIFE`
- `IMGLIFE`
- `ENVLDATE`

When these fields are initialized, issue the following API request:

```
FXXZccc(CCB,SNB,FCB,TRCB,TRIDB,TRODB)
```

Note: Although no data is supplied on this request, the TRIDB and TRODB are still required.

When the translator receives this request, it translates data from the application format into the EDI standard format in accordance with the transaction instructions.

Numerous errors can occur at this point. If errors occur, the error codes associated with the most severe error are posted in the CCB's `ZCCBRC` and `ZCCBERC` fields. An acceptable transaction can have errors. An acceptable error level for the transaction is established when a usage/rule is created by your EDI Administrator. With each call, your program must check the `TRXACCEPT` and `TRABORT` fields for error information.

At this point, numerous TRCB fields are provided as output, as described in the following tables.

Translation - TRCB Fields:: The TRCB fields described in Table 133 are updated during translation, based on translation type.

Table 133. Fields in TRCB updated during translation

Field name	Description
APPCTLNUM	The application control value. The application control value is defined by the application field with an **AC** data type, or by the fields that were assigned when the map was created. This value uniquely identifies the transaction to the application. Applies only to Version 2.1 and earlier.
XACFIELD	The application control value. The application control value is defined by the application field with an **AC** data type, or by the fields that were assigned when the map was created. This value uniquely identifies the transaction to the application. Applies only to Version 3.1 and later.
TRXACCEPT	Indicates whether the transaction had an acceptable translation.
TRABORT	Indicates that an error occurred that was so severe that the translator did not continue. If this flag is set to **Y**, the translator has terminated of its own accord.

Table 133. Fields in TRCB updated during translation (continued)

Field name	Description
TSKEY	The transaction handle for this transaction in format `YYYYMMDDHHMMSSxxnnnn`. This key value is returned as a packed10-byte value in this field, which is how it is stored in the database.
TSKEYU	The transaction handle for this transaction. This is the same value as that in the `TSKEY` field, except that `TSKEYU` is an unpacked (character) 20-byte value.
ERRNUM	The total number of errors flagged during data translation.
ERRCDES	An array of the first 10 different errors flagged during data translation. For more information, see "Translator Error Codes" on page 670.

Enveloping - TRCB Fields::   Numerous TRCB fields are related to the current status of an interchange being created. The next three tables describe the fields relating to the following three items:
- Interchange header and trailer
- Group header and trailer
- Transaction header and trailer

TRCB fields related to the interchange header and trailer are described in Table 134.

Table 134. Fields in TRCB related to the interchange header and trailer

Field name	Description
NEWENV	Indicates whether the transaction started a new interchange. If you want a copy of the interchange header, you can use a function code value of **1** to obtain an exact image of the interchange header. For more information, see "Retrieve interchange header API" on page 533. **Y** Started a new interchange **(other)** Did not start a new interchange
IHCTL	The interchange control number assigned to the current interchange. The value returned in this field is right-justified with leading zeros. Applies only to Version 2.1 and earlier.
IHXCTL	The interchange control number assigned to the current interchange. The value returned in this field is right-justified with leading zeros. Applies only to Version 3.1 and later.
GRPNUM	The total number of groups in the interchange at the current time, stored as a 4-byte binary value. See `IGT` for the character representation of this value.
TRNNUM	The total number of transactions in the interchange at the current time, stored as a 4-byte binary value. See `ITT` for the character representation of this value.
SEGNUM	The total number of segments in the interchange at the current time, stored as a 4-byte binary value. See `IST` for the character representation of this value.
ESIZE	The total number of bytes in the interchange at the current time, stored as a 4-byte binary value. See `IBT` for the character representation of this value.

Table 134. Fields in TRCB related to the interchange header and trailer (continued)

Field name	Description
ISID	The interchange sender ID (interchange header with data type **IS**).
IRID	The interchange receiver ID (interchange header with data type **IR**).
IDATE	The interchange date (interchange header with data type **DT**).
ITIME	The interchange time (interchange header with data type **TM**).
IVERREL	The interchange version and release (interchange header with data types **VR** or **LV**).
IGT	The character representation of `GRPNUM`.
ITT	The character representation of `TRNNUM`.
IST	The character representation of `SEGNUM`.
IBT	The character representation of `ESIZE`.
ISPW	The interchange password (interchange header with data type **PW**).
IAPREF	The application reference (interchange header with data type **AP**).

TRCB fields related to group header and trailer are described in Table 135.

Table 135. Fields in TRCB related to group header and trailer

Field name	Description
NEWGRP	Indicates whether the transaction started a new group. If you want a copy of the group header, you can use a function code value of **2** to obtain an exact image of the group header. For more information, see "Retrieve group header API" on page 533. **Y** Started a new group **(other)** Did not start a new group
GHCTL	The group control number assigned to the current group. The value returned in this field is right-justified with leading zeros. Applies only to Version 2.1 and earlier.
GHXCTL	The group control number assigned to the current group. The value returned in this field is right-justified with leading zeros. Applies only to Version 3.1 and later.
TRNGRP	The total number of transactions in the current group at the current time, stored as a 4-byte binary value. See `GTT` for the character representation of this value.
GSID	The group sender ID (group header with data type **AS**).
GRID	The group receiver ID (group header with data type **AR**).
GDATE	The group date (group header with data type **DT**).
GTIME	The group time (group header with data type **TM**).
GAPW	The group password (group header with data type **PW**).
GVER	The group version (group header with data type **VR**).
GREL	The group release (group header with data type **LV**).

Table 135. Fields in TRCB related to group header and trailer (continued)

Field name	Description
GTT	The character representation of `TRNGRP`.

TRCB fields related to transaction header and trailer are described in Table 136.

Table 136. Fields in the translator control block (TRCB) related to transaction header and trailer

Field name	Description
NEWTRN	Indicates whether this starts a new transaction. If you want a copy of the transaction header, you can use a function code value of **3** to obtain an exact image of the transaction header. For more information, see "Retrieve transaction header API" on page 534. **Y** Started a new transaction **(other)** Did not start a new transaction
THCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros. Applies only to Version 2.1 and earlier.
THXCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros. Applies only to Version 3.1 and later.
SEGTRN	The total number of segments in the current transaction stored as a 4-byte binary value. See `TST` for the character representation of this value.
TTC	The current transaction or message ID value. See also `TRNID` on 482.
TVER	The transaction version (transaction header with data type **VR**).
TREL	The transaction release (transaction header with data type **LV**).
TST	The character representation of `SEGTRN`.

Last call of session (TS)

If the translator is called at least once and the `TRABORT` field does not have a value of **Y**, invoke the translator one last time with a termination function code.

The termination function code signals the translator that the application is finished making enveloping requests. On receiving the request, the translator releases any resources that it acquired. If an interchange is active, the translator completes and writes the current interchange.

Special considerations (TS)

The following sections list special considerations.

Send raw data: In normal processing, WebSphere Data Interchange assumes that the application has detailed knowledge of the data being processed and communicates this information to the translator by setting the appropriate fields in the TRCB. The specific values that the application must communicate to the translator are:

- The internal trading partner ID associated with this transaction in the `INTPID` field
- The structure name for the data being provided in the TRIDB in the `ATSID` field
- The end of the transaction, indicated by a value of **Y** in the `EJECT` field

If you are writing a general-purpose application program that is capable of handling numerous application files, similar to the WebSphere Data Interchange Utility, your application might not be aware of all the details for all possible files.

The translator has a RAWDATA mode that can be used in these situations. The RAWDATA mode reverses the roles of the application and the translator in a few ways. Rather than the application sending the information below to the translator, the translator sends it to the application:
- Which structure is provided
- The internal trading partner ID
- That the transaction is complete and ready for translation

RAWDATA mode is signaled on the first call to the translator for a transaction by setting the `RAWDATA` field to **Y** and setting the `ATFID` field to the data format ID that is being used for this transaction. If your application knows the internal trading partner ID, that value can be put into the `INTPID` field.

If your application does not know the internal trading partner ID, initialize the `INTPID` field with blanks, and the translator can extract the internal trading partner ID from the application data.

Note: When you use the WebSphere Data Interchange Utility, the internal trading partner ID must be included in the application data and is extracted by the translator when the appropriate structure is provided. However, an API program can set the internal trading partner ID value in the `INTPID` field, superseding the requirement that this value be contained in the application data.

For RAWDATA mode to succeed, your EDI administrator must provide the raw data specifications when the data format (`ATFID` value) is defined. The raw data specifications include:
- The name of the field that contains the internal trading partner ID.
- Either the name of a structure that starts a transaction, or the name of a structure that ends a transaction, or both. It is best to provide the name of a structure that ends a translation. The translator can then detect and process the end of the current transaction without waiting to receive the record that begins the next transaction.
- The offset into each structure where a value is located that uniquely identifies the structure.
- A unique value that identifies each structure defined in the data format.

When using generic send usages/rules, the application can select a specific generic usage/rule by generic routing code. The routing code is provided by the application in the TRCB, in the C record, or in an application field (raw data only).

After each request, the translator posts information into the TRCB that specifies the status of the transaction. The TRCB fields specified in Table 137 are updated.

Table 137. TRCB fields defining the status of the transaction

Field	Description
ATSID	The name of the structure that was received, based on the raw data specifications.
INTPID	The name of the internal trading partner ID. The translator returns the internal trading partner ID value in this field when the structure containing the value is received. At this point, the processing in the first call for a transaction takes place. If the internal trading partner ID is not in the first structure, first call processing is delayed until the internal trading partner is received. This is critical to your program, because if the transaction is not acceptable (because of missing items in the environment, such as a map), and partial structures have already been received, your API must notify the translator to discard all the previous data. This is done by making one more request with an EJECT value of **F**. For example, if the internal trading partner ID is provided in the third structure, and an error is returned on this third structure because a map was not found, you must call the translator again with an EJECT value of **F**. This causes the previously accepted structures for this transaction to be discarded. If a internal trading partner ID is not defined in the data format, or if this field contains all blanks, the value of INTPID from the first call for the transaction is used as the default.
TRNSTAT	Indicates the status of the transaction and, therefore, determines what your program must do next. Valid values are: **I** — The data that you supplied has been ignored. This might occur because the translator could not determine the structure based on the raw data specifications, or because the raw data specifications indicated that a specific structure started each transaction and that structure has not yet been supplied. **S** — The data that you supplied was recognized and has started a transaction. **C** — The data that you supplied was identified and continued the transaction. **R** — The data that you supplied was identified as starting a transaction, but a transaction is already in progress. Issue a last call for transaction to terminate the previous transaction, and call the translator again with the same data. **Y** — The data that you supplied was identified as ending a transaction. To force the translation of the current data, issue a last call for transaction.

Providing a Partial Structure: Usually, all data for a given structure is provided to the translator in a single request by moving the complete structure into the TRIDB. However, if you can not provide all the data in a single request, you can indicate that a partial structure is being provided by setting the EJECT field to **X**.

The translator usually ignores the DATALEN field of the TRIDB, because the amount of data provided is defined by the structure name (ATSID) and the structure definition in the data format. If the EJECT field is set to **X**, DATALEN must contain the amount of data provided with this request.

Continue to provide data with an EJECT value of **X** until you are ready to provide the final data for the structure. When you provide the final data for the structure, set the EJECT field to blank.

Note: In RAWDATA mode, if you send partial data for a structure that contains the internal trading partner ID field, that field must be present in the first partial record you send for the structure.

Send Recovery Scope: WebSphere Data Interchange allows you to specify how much processing must take place before WebSphere Data Interchange makes a request to the underlying system to commit all resources. Once a COMMIT request is issued, all database changes made up to this point become permanent.

The amount of processing done before a COMMIT request is issued is called the recovery scope. The translator allows a transaction level recovery scope or an interchange level recovery scope.

With interchange level recovery, you can use the INMEMTRANS field to increase concurrency. The INMEMTRANS field specifies the maximum number of transactions that must be maintained in virtual storage before any database updates are attempted. Database updating occurs when either the value in INMEMTRANS**,** or the end of the interchange, is reached. This value is important in an environment where multiple application programs are requesting translation services concurrently. The higher the value for INMEMTRANS, the more concurrency achieved. See "Translator Control Block (TRCB)" on page 642 for the description of INMEMTRANS and virtual storage considerations.

Do not use interchange level recovery scope if an interchange is to contain a large number of transactions. If the number of transactions exceeds the value in INMEMTRANS, other processes are locked out until the entire interchange is complete. Also, DB2 has a limit on the number of locks that a process can hold. A large number of transactions could cause this value to be exceeded.

You indicate the recovery level in the SCOPE field of the TRCB. Valid values are:

E   Requests interchange level recovery. WebSphere Data Interchange does not issue a database COMMIT request until an interchange is complete and has been written to the file associated with the network. Applies only when enveloping operations are taking place.

T or (other)

> Requests transaction level recovery. WebSphere Data Interchange issues a database COMMIT request at the end of every successful transaction. **T** is the recommended value.

Regardless of the value in the SCOPE field, when the translator determines that a COMMIT request must be issued, the COMMIT is issued before the translator returns control to the application. If the database changes made by the application must be synchronized with the database changes made by WebSphere Data Interchange, you can tell the translator not to issue a COMMIT by setting the NOCOMIT field to **Y**. The application can then make database updates based on the results of the translation and call the translator to issue a database commit (function code **991**). You must use the **991** function code to request the COMMIT (either directly to DB2 and/or CICS, or by using the SYNCPOINT service) rather than using the application program because the translator must know that the COMMIT has taken place. A COMMIT request issued by the application without the knowledge of the translator could result in a deadlock situation.

The translator issues the necessary call to commit all resources that have been updated. For more information, see "Issue commit API" on page 532.

Note: Setting the NOCOMIT field to **Y** prevents the translator from issuing commits, but it does not prevent WebSphere Data Interchange termination from issuing a COMMIT. Making a SYNCPOINT service call with the syncpoint interval set to **-1** prior to the WebSphere Data Interchange termination call will prevent this termination commit. For more information, see "SYNCPOINT services" on page 568.

If a system or program failure occurs, changes made to the database since the last COMMIT are removed by the file system. A guarantee of database integrity is only available for the DB2 version of WebSphere Data Interchange with the Transaction Store files defined as recoverable.

Forcing Interchange Termination: If enveloping occurs at the same time as translation, the translator attempts to build the largest interchange possible. The current interchange is completed and a new one started only when certain field values change. For more information and a complete list of items that cause a new interchange, see "Fields that cause a new interchange to start" on page 515.

You cannot control the size or content of an interchange using the WebSphere Data Interchange Utility. If you want your application to control the size or content of an interchange, you must issue a request to close and queue the current-interchange (function code **990**). You must issue this function request between the last call for the current transaction and the first call for the next transaction. For more information, see "Close and queue interchange API" on page 516.

The ESIZE field contains the current size of the interchange. This value can be compared to a threshold value that, once exceeded, signals your program to issue the request to close the current interchange. When an interchange is closed, a group trailer

segment (if groups are being used) and an interchange trailer segment are added to the end of the interchange. When setting the threshold value, take the expected sizes of these segments into account.

If you are using C and D records, you can tell the WebSphere Data Interchange Utility to force the termination of an interchange by using the Z1 record. However, you cannot limit the size of an interchange to a specific value using the WebSphere Data Interchange Utility or facility.

Batches and Bundles: You can group transactions by using the BATCHID field or the BNDLFLAG field. These two fields are not related and the transaction associations formed by each are different.

BATCHID creates an informal association provided as a fast, convenient way to select transactions from the Transaction Store. An alternate index is keyed on the BATCHID value making it the best field on which to base a selection if you do not use the THANDLE field.

For example, you could use this field to isolate all the transactions from a particular program execution. If a problem occurs with the execution, you can use BATCHID to place all the transactions from that execution on hold, preventing them from being enveloped until the problem is resolved. You can also use BATCHID to reenvelope transactions from a certain program execution if the file containing the interchanges was corrupted.

However you use it, the BATCHID association between transactions is informal and differs little from a set of transactions created using any common value, such as trading partner nickname or EDI standard ID.

On the other hand, BNDLFLAG creates a formal relationship between transactions. The transactions associated with BNDLFLAG are bundled (clustered) and are generally treated as a single unit. Your application can use bundling if you want to isolate a group of transactions and have them processed as a single unit rather than as individual transactions.

Bundling is required for UN/TDI transactions but is optional for the other EDI standards. With all other EDI standards, transactions are independent of each other and can be handled independently. For example, X12 defines a header segment, detail segments, and a trailer segment within a transaction, while UN/TDI defines a header transaction, detail transactions, and a trailer transaction. You cannot send the header transaction in one interchange, and the detail and trailer transactions in another interchange. You must send them all together in the same interchange. Do not include any additional transactions in that interchange. By bundling, you tell WebSphere Data Interchange which header belongs with which detail and trailer transactions. WebSphere Data Interchange can then envelope the appropriate transaction separately from unrelated transactions and can treat multiple transactions as a single unit.

Any action performed on any member of the cluster is done to all members of the cluster. During enveloping operations, clustered transactions are not placed in the same interchange as transactions that are not clustered. One cluster is not put in the same

interchange as another cluster. Members in a cluster can have different BATCHID values, and you can use the same BATCHID in more than one cluster.

Outbound incremental translation: Typically, during outbound translation, all the application data being translated is read into memory before translation begins. This can be a disadvantage if application files are large. With incremental translation, application data is passed into WebSphere Data Interchange in small chunks, and these chunks can then be translated and released from memory. This minimizes the use of application data memory.

Incremental translation requires application data to be grouped into headers, details, and trailers. These three groups are rigid, and data from one cannot be mapped into another. Incremental translation is only suitable for iterative detail data, such as line items on an invoice, which means the data group would consist of one header, many line items, and one trailer. During incremental translation, header data is passed into WebSphere Data Interchange first and then translated. Next, one or more detail records (line items) are passed into WebSphere Data Interchange and then translated. You can pass in and translate as many detail records as necessary. Finally, the trailer is passed into WebSphere Data Interchange and translated.

When preparing to translate data incrementally, first make sure the application data is grouped as described above. Next, make sure the map you use maintains this grouping so that data from one group is not mapped into another group. Then, you must place an &BOUNDARY special literal in the map.

You must map the &BOUNDARY literal on the data element just prior to the main detail loop. If the data element is already mapped to an application field name, create a second map with the &BOUNDARY literal. The &BOUNDARY literal acts as a switch that tells the control string generator to place a special end-of-header marker at the beginning of the next level-one loop.

Do not map the literal with an application field name. The &BOUNDARY literal must only be mapped in maps used for incremental translation. Incremental translation must never be attempted with maps that do not contain a properly mapped &BOUNDARY literal. The general rules for mapping the &BOUNDARY literal are:

- If there are loops defined in the header, &BOUNDARY can be mapped anywhere after the beginning of the last level-one header loop and before the beginning of the detail loop.
- If there are no loops defined in the header, &BOUNDARY can be mapped anywhere in the header.

After this is done, recompile the map to generate the control string. Finally, enable your API program to process the data as described next. Special considerations for using the `EJECT` and `BOUNDARY` fields with incremental translation are described below. For complete details on using an API for translating data, see "Translation services" on page 443.

Incremental translation: The header records are first passed into WebSphere Data Interchange, each with `EJECT`(**blank**) and `BOUNDARY`(**X**). After the last header record is

passed in, call the translator again with EJECT(**Y**) and BOUNDARY(**H**). This tells WebSphere Data Interchange to translate the header, and remove the header application data from memory. Make this call without data.

This next step is iterative. One or more detail records can be passed into WebSphere Data Interchange, each with EJECT(**blank**) and BOUNDARY(**X**). To start translation of these detail records, call the translator again with EJECT(**Y**) and BOUNDARY(**D**). This tells WebSphere Data Interchange to translate the detail records, and remove those records from memory. Make this final call without data. You can repeat this step as many times as necessary.

Finally, after all detail records have been passed and translated, pass all trailer records into WebSphere Data Interchange, each with EJECT(**blank**) and BOUNDARY(**X**). After the last trailer record is passed, make a subsequent call with EJECT(**Y**) and BOUNDARY(**T**). This tells WebSphere Data Interchange to translate the trailer, and remove the trailer records from memory. At this point, translation is complete and if enveloping is not delayed, an envelope will be created and queued.

Pageable translation

Pageable translation is designed to better utilize system memory during translation by transferring incoming data buffers to DASD once the allotted number (1000) of internal buffers has been exhausted. The maximum buffer size is 28,632 bytes. This means that approximately 28 MB of virtual storage can be used to hold data before pageable translation is triggered. The maximum amount of data that WebSphere Data Interchange can transfer with pageable translation is approximately twenty gigabytes. Pageable translation in CICS uses temporary storage queues with names that begin with EDI. Therefore, the size of DFHTEMP may have to be considered if pageable translation is desired in CICS.

A sample definition of EDIVAX is:

```
//EDIVAX  DD DISP=(NEW,DELETE,DELETE),UNIT=SYSDA,SPACE=(CYL,1500)
```

In z/OS, you must define a temporary work file with ddname EDIVAX. Allocate the amount of space for this file depending on the maximum amount of data to be translated. You can calculate the amount of storage needed to translate an envelope without using pageable translation by adding the following components:

```
Number of bytes in largest interchange
+
4 MB overhead
+
Number of bytes in largest application transaction image
+
Number of structures in largest interchange multiplied by 120 bytes
```

The number of structures in the largest interchange includes structures that are passed separately (records) and substructures that are not passed separately but which contain data during translation. Pageable translation deals with the first two components, and ensures that the amount of virtual storage required for them does not exceed 28 MB. The other components are not addressed by pageable translation.

Translation services

In CICS, pageable translation uses TS queues with names that begin with EDI. Therefore, you might have to modify the size of DFHTEMP if you want to use pageable translation in CICS.

To enable pageable translation using the API, set the `VAXFLAG` field in the TRCB to **X**. To enable pageable translation using the WebSphere Data Interchange Utility, use the `PAGE`(**Y**) keyword on TRANSLATE commands.

Translate-to-application API

On a translate-to-application API request, the translate-to-application service uses the settings from the Administrator's Menu to:

- Takes data in the format defined by the EDI standard (from the `EDI standards` field), and
- Transforms that data into the application-defined format (from the `Data Format` field) as defined by a map (from the `Trading partner transactions` field).

This is the same API that the WebSphere Data Interchange Utility uses internally when you issue any of the following PERFORM commands:
- DEENVELOPE AND TRANSLATE
- RECEIVE AND TRANSLATE
- RETRANSLATE TO APPLICATION
- TRANSLATE TO APPLICATION

Receiving and deenveloping

You can use the following API functions to receive a file, deenvelope the interchanges in the file received, and translate the EDI standard data into an application format for processing by various application programs:

Receiving data
Communication services (function code **232**). See "Receive and restart receive API" on page 552.

Deenveloping data
Enveloping/Deenveloping services (function code **214**). See "Deenvelope API" on page 518

Translating data
Transaction services (function code **213**). See "Translate specific API" on page 478

The Communication service invokes the network and receives the data from the network into the specified file.

The Deenveloping service:
- Parses the interchanges in the file
- Extracts each transaction
- Places details of each transaction along with the transaction image into the Transaction Store
- Generates a functional acknowledgment, if one was requested
- Reconciles any received acknowledgments with the original transactions

The Translation service retrieves the transaction image from the Transaction Store and translates the data into the application format.

You can combine the deenvelope and translate functions into one step by calling the Translation service with a function code of **212**. For more information, see "Translate-file-to-application API" on page 486.

Processing speed can be improved when you use the DEENVELOPE AND TRANSLATE combination command. This sequence is illustrated in Figure 24 on page 476.

Translation services

Application program   WebSphere Data Interchange

Perform API initialization

Perform API receive ◄

If CMCB.FILERCVD EQ 'Y'

   Perform First call of session (TF)

   Do until TRCB.TRABORT EQ 'Y'

      Perform First call of transaction (TS) ◄

     If TRCB.TRABORT NE 'Y'

      If TRCB.TRXACCEPT EQ 'Y'

       Do while TRCB.EJECT NE 'Y'

         Perform Write application record

         Perform Subsequent calls (TF/TA) ◄

       ENDDO

      ELSE

       Perform Transaction not acceptable

      END - IF

     END - IF

   ENDDO

   If CCB.ZCCBRC NE 4 or CCB.ZCCBERC NE 1

     Perform Unexpected termination

   END - IF

END - IF

Perform API termination

WebSphere Data Interchange side:

Data from trading partners

Network program called by WebSphere Data Interchange

Data written to "RECVFILE"

Locate and translate next transaction and return first structure
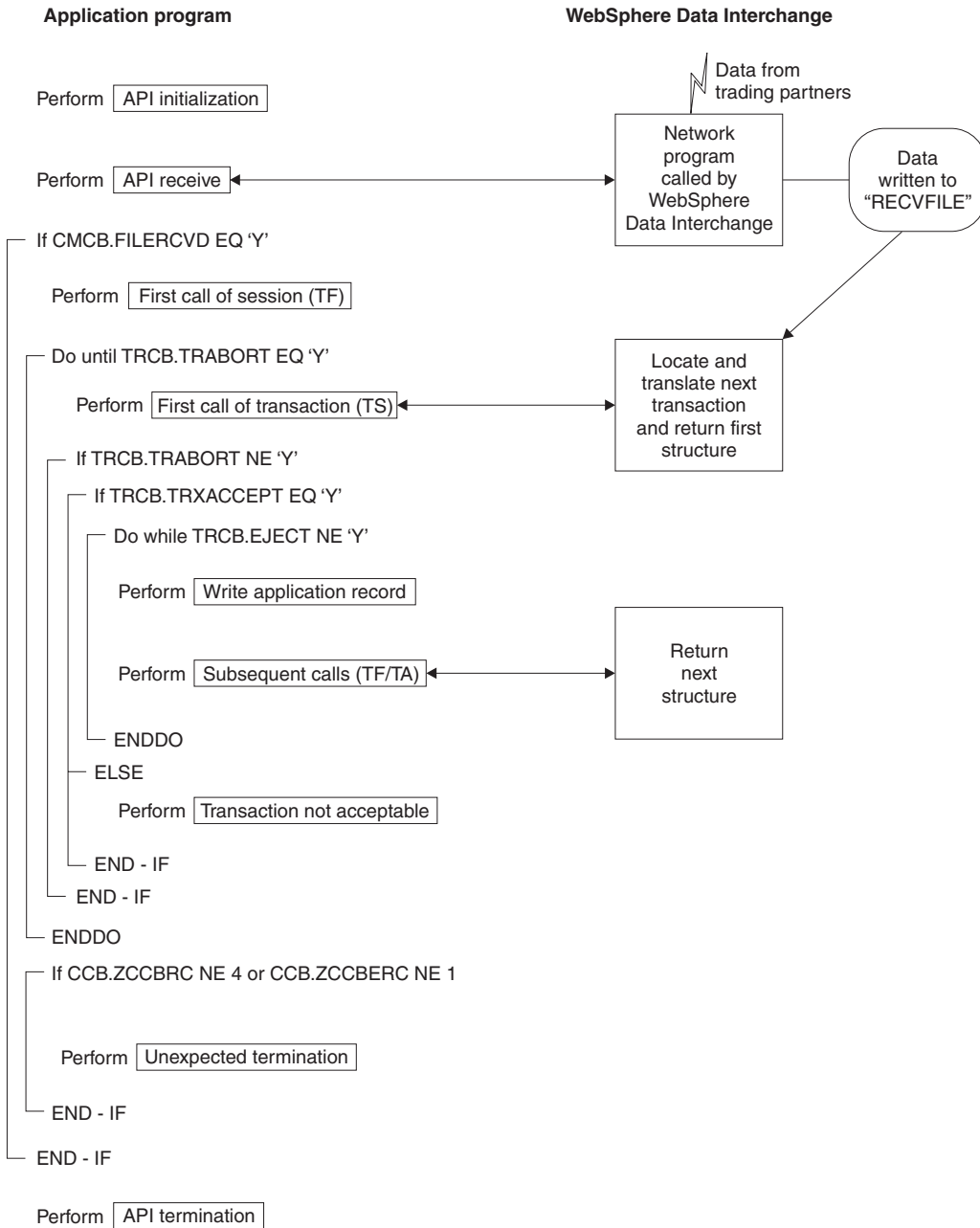
Return next structure

Figure 24. Receive, deenvelope, and translate process

Receiving and deenveloping considerations

The following considerations apply to these services:

- The communications receive function invokes a network program to receive data into a file. The data placed in this file is under the control of the network program. WebSphere Data Interchange counts the bytes in the interchanges in the file in order to update the management reporting component of WebSphere Data Interchange. No information from the interchanges is recorded in the Transaction Store.

- The deenvelope function generates the functional acknowledgment for the data received and reconciles the received functional acknowledgments with the original transactions. This is true whether code **214** (deenvelope only) or code **212** (deenvelope and translate) is used.

- Because deenveloping and translation can be separate functions, you need a way to tell the translator which transaction from the Transaction Store to process. Use the transaction handle (THANDLE) that is assigned to the transaction when it is added to the store. The program that is deenveloping the data must track the transaction handle. If the program does not track the transaction handle, you can retrieve a list of these values from the Transaction Store using the QUERY command.

- No single API function is the equivalent of the WebSphere Data Interchange Utility RECEIVE AND TRANSLATE command. Using the API, receiving the data, and deenveloping and translating the data are two separate steps.

Test Translate-to-application

You can test your transactions by sending data in either test mode or production mode. If your main purpose is to test the transaction, use test mode. If your main purpose is to test the path between your system and that of your treading partner, use production mode.

When you send test data in text mode (function code **211**), the following occurs:

- Data is translated to application format in test mode. Your program follows the same steps as for production mode. Interchanges are read from a file and parsed. Transactions are translated and presented to your program one transaction at a time.

  However, no information is written to the Transaction Store, and no functional acknowledgments are generated or reconciled. If you want information written to the Transaction Store, or if you want functional acknowledgments returned to your trading partner, you must use the second test method outlined below.

- When you use the test function code, WebSphere Data Interchange looks for a test usage/rule (if one is available) and forces test mode. You can verify the results of your program's efforts by examining the application data produced, as well as the information recorded in the event log. To view or print the event log, choose Event logging from the Administrator's Menu.

When you send test data in production mode, the following occurs:

- The data is sent with the test indicator set to **T** in the interchange header segment, if the envelope type being used has a test indicator. When you use the test indicator, translation and functional acknowledgment processing occurs as usual (including updating the Transaction Store).

- The interchange that contains the functional acknowledgment is identified as being for test purposes. You might prefer this method, rather than the first option, because you can test the entire path from your trading partner to your program, and back to your trading partner (including functional acknowledgments).

Translate-to-application API

The functions of the translate-to-application API are described in more detail in the following sections:

Translate specific

Translates a specific transaction into application format. Use this API to translate a transaction that has been deenveloped or deenveloped and translated, and is already in the Transaction Store.

Translate file

Deenvelopes and translates all the transactions in a file. Use this API when all the interchanges in a file have been processed, and all transactions in the interchanges have been deenveloped, translated, and added to the Transaction Store.

Translate specific API

The basic format of the API request to translate a specific transaction from EDI standard format to application format is:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The unique parameters for the Translate specific API request are defined in Table 138.

Table 138. Translate specific API request parameters

Parameter	Description
SNB	**ZSNBLL** 32 **ZSNBNAME** TRANPROC **ZSNBPC** 6
FCB	**ZFCBFUNC** 213
TRCB	The translator control block. See "Translator Control Block (TRCB)" on page 642 for details about this block.
TRIDB	The input data block. WebSphere Data Interchange uses this block as a work buffer. This buffer must be at least 32000 bytes in length. Its maximum size must be the size of the maximum EDI standard segment that is received, excluding the BIN segment. See "Translator Input Data Block (TRIDB)" on page 672 for a general description of this block.
TRODB	The output data block. This block contains the application data produced. The format of the `Data` field in this block must match the format of the data defined in the data format. The block has a minimum size of 32000. See "Translator Output Data Block (TRODB)" on page 674 for a general description of this block.

First call of session (TA)

There are numerous fields in the control blocks defined for the translate-to-application API function that need only be established once. These values can be established before the first call and they do not have to be refreshed or changed before another call. The following control block fields must be initialized for the first session call.

For the SNB:

ZSNBLL

> 32

ZSNBNAME

> TRANPROC

ZSNBPC

> **6** (all calls for translation services have six parameters)

For the FCB:

ZFCBLL

> 4

ZFCBFUNC

> 213

For the TRCB:

BLKLEN

> 1536

BLKNME

> EDITRCB

BLKTYPE

> The format for the TRIDB and TRODB buffers:
>
> **H**   Unlimited size
>
> **(other)**   Limited to 32768 bytes

XPANDED

> **Y**. Indicates that WebSphere Data Interchange must check the `BLKLEN` field to determine the software version and release being used.

BATCHID

> The batch ID for a group of transactions. The Transaction Store creates an alternate key value using the batch ID. Other than using the transaction handle, searching on batch ID is the quickest way to retrieve a transaction from the Transaction Store during the selection process. The default is `DDHHMMSS`, where `DD` is the current day of the month, and `HHMMSS` is the current time.
>
> > **Note:** This field is checked with each call to translator. The value in this field when the first call of transaction (TA) is made will be associated with the transaction.

ERRFILTER

> Indicates which error codes to filter out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more detailed information, see "Error filtering" on page 22.

MAPCHAIN

> Indicates whether mapchaining is in effect.

Y   Translates the current transaction again using the value in the variable DIMAPCHAIN to select the map

(other) Translates the next transaction

FORCETEST

Indicates whether the translate process is forced to select only test usages/rules. If a value of **Y** is used in this field for the DEENVELOPE command, you must also use it with the TRANSLATE TO APPLICATION command to select only deenveloped transactions.

For the TRIDB:

BLKLEN

Set this field to the size of the data block, including the BLKLEN field. The minimum value for this field is 32000 bytes.

RESERVED

Binary zeros.

For the TRODB:

BLKLEN

Set this field to the size of the data block, including the BLKLEN field. The minimum value for this field is 32000 bytes.

RESERVED

Binary zeros.

First call for transaction (TA)

The following TRCB fields are used on the first call for transaction and must be set before the first call for transaction is made:

TSKEY The transaction handle for the transaction you want to translate. The format of the handle is YYYYMMDDHHMMSSxxnnnn formatted as a 10 byte packed field. If your program does not handle packed values, initialize this field to all blanks or all binary zeros and use the TSKEYU field instead.

TSKEYU

The transaction handle for the transaction you want to translate. The format of the handle is YYYYMMDDHHMMSSxxnnnn formatted as a 20-byte character field. Use this field only if the TSKEY field does not have a value.

RAWDATA

Indicates whether the translator must automatically fill in the values for the internal trading partner ID and the record ID before returning the structure to your application.

Y   Fills in the values for internal trading partner ID and record ID

(other) Does not fill in the values for internal trading partner ID and record ID

The following TRODB fields are used on the first call for transaction and must be set before the first call for transaction is made.

BLKLEN

> The length of the TRODB, including this field. If you are using the same TRODB for all calls, initialize this field only once. The minimum length is 32000 bytes.

RESERVED

> Set this field to zeros. If you are using the same TRODB for all calls, initialize this field only once.

When initialization is complete, the translator is invoked with the following API request.

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

On the first call to the translator for a transaction, the transaction identified in the TSKEY or TSKEYU field is retrieved from the Transaction Store along with the interchange, group, and transaction header images. The interchange sender ID and sender qualifier are extracted from the interchange header. These values are used to identify the trading partner who sent the data. For a description of the information used to identify the trading partner, see "Locating sending trading partner profile members" on page 529.

The transaction/message ID value is extracted from the transaction header. At this point, the translator attempts to locate the trading partner usage/rule and map. The usages/rules and maps must be established by your EDI Administrator before API requests are made. For a description of the fields used to locate a map, see "Locating sending and receiving trading partner profile members" on page 530.

Numerous errors can occur at this point. If an error occurs, the return code and extended return code for the error are posted in the `ZCCBRC` and `ZCCBERC` fields of the CCB. With each call, your program must check the `TRXACCEPT` and `TRABORT` fields for error information.

Transaction TRCB Fields (TA):: The tables in this section describe the fields returned on the first call for a transaction. Such a large quantity of information is returned on the first call for a translate-to-application request that multiple tables are used to show the TRCB fields returned. Each table begins with a description of the data contained in the table.

The TRCB fields described in Table 139 on page 482 provide some basic attributes of the transaction just processed, and relate to the transaction side of the processing rather than the application side.

Table 139. Transaction attribute fields in the translator control block (TRCB)

Field name	Description
TRNID	The standard transaction or message ID for the transaction or message received
TEST	Indicates the type of transaction **I** Information **P** Production **T** Test
MAPKEY	The map used to translate the EDI standard data
TPNICK	The trading partner nickname associated with the transaction
DUPTRAN	Indicates whether the interchange being received has been received before
TRXACCEPT	Indicates whether the transaction had an acceptable translation and indicates that application data has been returned in the TRODB, unless the EJECT field has a value of **Y**.
TRABORT	Indicates whether an error occurred that was so severe that the translator did not continue. If this field is set to **Y**, the translator has terminated of its own accord.
TSKEY	The key value assigned to the transaction in the Transaction Store. This is called the transaction handle and has a format of YYYYMMDDHHMMSSxxnnnn. It is returned as a packed 10-byte value in this field, which is how it is stored in the database.
TSKEYU	The key value assigned to the transaction in the Transaction Store. It has the same value as the TSKEY field, except that TSKEYU is an unpacked 20-byte value.
EJECT	Indicates whether the end of the transaction has been reached. For more information, see "Partial structures (TA)" on page 499. **Y** Transaction ends. No data is returned. The next call is treated as the first call for the next transaction. **(other)** Transaction continues. The next call will return either the next structure or an EJECT value of **Y**.
ERRNUM	The total number of errors flagged during data translation.
ERRCDES	An array of the first 10 different errors flagged during data translation. For more information, see "Translator Error Codes" on page 670.

The TRCB fields described in Table 140 are application-related and provide information about the application side of the processing rather than the transaction side.

Table 140. Application related fields in the translator control block (TRCB)

Field name	Description
ATFID	The data format ID associated with this transaction.
ATSID	The name of the structure being returned on this call. The data related to the structure is in the TRODB.

Table 140. Application related fields in the translator control block (TRCB) (continued)

Field name	Description
APPFILE	The ddname for the application file to which the application data must be written. This field is taken from the data format definition, but can be overridden in the trading partner receive map rule record.
APTYPE	The type of file identified in APPFILE. Applies only to CICS and MQ (which is supported in both z/OS and CICS). Valid values are: **MQ** MQSeries queue profile member name **PG** Program identified in APPFILE **TD** TD queue name identified by the first four characters of APPFILE **TM** TS queue (main) identified in APPFILE **TS** TS queue (auxiliary) identified in APPFILE **TX** CICS transaction code identified by the first four characters of APPFILE For z/OS, if you do not specify this keyword, this field is ignored, and the ddname of a sequential file is used. For CICS, the default is **TS**.
INTPID	The internal trading partner ID taken from the map receive usage/rule.
APPCTLNUM	The application control value. The application control value is defined by the application field with an **AC** data type, or by the fields that were assigned when the map was created. This value uniquely identifies the transaction to the application. Applies only to Version 2.1 and earlier.
XACFIELD	The application control value. The application control value is defined by the application field with an **AC** data type, or by the fields that were assigned when the map was created. This value uniquely identifies the transaction to the application. Applies only to Version 2.1 and earlier.
RAWDATA	Indicates whether the raw data processing was performed. Applies only if raw data processing was requested by setting the RAWDATA field to **Y** on the first request. With raw data processing, the translator sets the record ID and internal trading partner ID field values. The format of data written to the application file by the application program is under the control of the application program. All the necessary data for C and D records is available, but need not be used if raw data output is requested. **Y** Raw data processing occurred. **N** Raw data processing did not occur because the data format did not have raw data specifications.

The translator output data block contains the application data. For each call to the translator, a single structure (record) of application data is returned. The structure returned is indicated by the ATSID field of the TRCB. Data is returned only if the transaction was acceptable (TRXACCEPT value of **Y**) and this is not the end of the transaction (EJECT value is not **Y**). See "Partial structures (TA)" on page 499 for special considerations.

The following TRODB fields are concerned with the application data:

DATELEN
> The number of characters of data being returned in the DATA field.

> **DATA** The application data with a format defined for the structure (`ATSID` field).

The TRCB fields described in Table 141 are concerned with the interchange for the current transaction.

Table 141. Interchange header/trailer fields in the translator control block (TRCB)

Field name	Description
QTPNICK	The trading partner nickname for the last transaction processed.
ENVTYPE	Indicates the envelope type of the interchange. Valid values are: **E** UNB/UNZ **I** ICS/ICE **T** STX/END **U** BG/EG **X** ISA/IEA
IHCTL	See `IHXCTL`.
IHXCTL	The interchange control number assigned to the current interchange. The value returned in this field is right-justified with leading zeros.
ISYNTAXID	The interchange syntax ID for envelope types **E** and **T**.
ISYNTAXVER	The interchange syntax version for envelope types **E** and **T**.
ISIDQUAL	The interchange sender ID qualifier for envelope types **E**, **I**, and **X**.
ISID	The interchange sender ID (interchange header with data type **IS**).
ISENDNAME	The interchange sender name for envelope types **U** and **T**.
IREVROUT	The interchange reverse routing for envelope type **E**.
IRIDQUAL	The interchange receiver ID qualifier for envelope types **E**, **I**, and **X**.
IRID	The interchange receiver ID (interchange header with data type **IR**).
RECVNAME	The interchange receiver name for envelope types **U** and **T**.
IROUTEADDR	The interchange routing address for envelope type **E**.
IDATE	The interchange date (interchange header with data type **DT**).
ITIME	The interchange time (interchange header with data type **TM**).
IVERREL	The interchange version and release (interchange header with data types **VR** or **LV**).
ISPW	The interchange password (interchange header with data type **PW**).
IAPREF	The application reference (interchange header with data type **AP**).
ISTDID	The interchange standard ID for envelope types **I** and **X**.
IPRIOR	The interchange priority code for envelope types **E** and **T**.
ICOMMAGREE	The interchange communication agreement for envelope types **E** and **T**.

The following TRCB fields are concerned with the group for the current transaction:

GHCTL
> See GHXCTL.

GHXCTL
> The group control number assigned to the current group. The value is returned in this field as a right-justified value with leading zeros.

GSIDQUAL
> The group sender ID qualifier for envelope type **E**.

GSID The group sender ID (group header with data type **AS**).

GRID The group receiver ID (group header with data type **AR**).

GRIDQUAL
> The group receiver ID qualifier for envelope type **E**.

GDATE
> The group date (group header with data type **DT**).

GTIME The group time (group header with data type of **TM**).

GAPW The group password (group header with data type **PW**).

GVER The group version (group header with data type **VR**).

GREL The group release (group header with data type **LV**).

GRESPAGENCY
> The group responsible agency code for envelope types **E**, **U**, and **X**.

The following TRCB fields are concerned with the transaction header for the current transaction:

NEWTRN
> Indicates whether this transaction had an acceptable translation. If you want a copy of the transaction header, you can use a function code value of **3** to obtain an exact image of the transaction header.
> **Y** The transaction had an acceptable translation.
> **(other)** The transaction did not have an acceptable translation.

THCTL See THXCTL.

THXCTL
> The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros.

TTC The current transaction or message ID value. See also TRNID on 482.

TVER The transaction version (transaction header with data type **VR**).

TREL The transaction release (transaction header with data type **LV**).

Subsequent calls (TF/TA)

The items listed below summarize the activity that takes place on the first call for a transaction (TA):

1. The next transaction in the file is located or the specific transaction you requested is retrieved from the Transaction Store.

2. The map associated with the transaction is found.

3. The translation from EDI standard format into application structures takes place.

4. The application structures are sorted into the order defined by the data format definition.

5. The first application structure is returned to the calling program in the TRODB.

If the first call for a transaction was successful, the `TRXACCEPT` field has a value of **Y**, and the application continues to call the translator using the following call with the same parameters as the first call of a transaction):

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The application continues to call the translator until the `EJECT` field has a value of **Y**. For each call made, the next application structure is returned in the TRODB and the name of the structure is placed in the `ATSID` field.

Last call for transaction (TA)

The translator signals that the last call for a transaction has just been made by setting the `EJECT` field to a value of **Y**. No data accompanies this call. At this point, your application might perform application-related processing relative to the end of the transaction, including updating the application databases. The next call is interpreted as the first call for the next transaction. The translator issues a COMMIT request with the next call.

Last call of session (TA)

When all the specific transactions have been processed, call the translator one last time with an end-translation function code. For more information, see "End translation/enveloping API" on page 517.

Translate-file-to-application API

You can request a translate-file-to-application in either production or test modes. Function code **212** requests translation in production mode, and function code **211** requests translation in test mode.

You do not need to change your program to switch from test mode to production mode. You only need to change the function code used to invoke the services. Test mode is different from production mode in the following ways:

- WebSphere Data Interchange assumes that the test indicator is set in the interchange, even when it is not present, or when it indicates production data. This forces the use of a test transaction, if one exists.
- WebSphere Data Interchange automatically logs the EDI standard image received and the application data produced.
- WebSphere Data Interchange does not write any data to the Transaction Store to prevent cluttering up the store with test data.
- Functional acknowledgments are not generated even if requested in the usage record.
- Statistics maintained by the Management Reporting component of WebSphere Data Interchange are not updated.

The basic format of the API request to translate-to-application format is:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The unique parameters for this API request are:

SNB   ZSNBLL 32 ZSNBNAME TRANPROC ZSNBPC 6

FCB   ZFCBFUNC

      212   production

      211   test

TRCB   The translator control block. For more information, see "Translator Control Block (TRCB)" on page 642.

TRIDB The input data block. WebSphere Data Interchange uses the input data block as a work buffer. The minimum size is 32000 bytes. Maximum size must be the size of the largest EDI standard segment that is received excluding the BIN segment. See "Translator Input Data Block (TRIDB)" on page 672 for a general description of this block.

TRODB

      The output data block. This block contains the application data that has been produced. The format of the data in this block matches the format of the data defined in the data format. The minimum size is 32000 bytes. See "Translator Output Data Block (TRODB)" on page 674 for a general description of this block.

First call of session (TF)

There are numerous fields in the control blocks defined for the translate-to-application API function whose values need to be established only once. The values can be established before the first call and they do not have to be refreshed or changed before another call. Because the first call for a session also qualifies as the first call for a transaction, you must also follow the instructions in the next section, "First call for transaction (TF)". The following tables illustrate the control blocks, the fields within the control blocks, and the initialization considerations.

SNB initialization for translate-file-to-application:

ZSNBLL

      32

ZSNBNAME

      TRANPROC

ZSNPC 6 (all calls for translation services have six parameters)

FCB initialization for translate-file-to-application:

ZFCBLL

      4

ZFCBFUNC

      212   production

      211   test

TRCB initialization for translate-file-to-application:

BLKLEN

      1536

BLKNME

   EDITRCB

BLKTYPE

   The format for the TRIDB and TRODB buffers:
   H   Unlimited size
   (other)   Limited to 32768 bytes

XPANDED

   Y Indicates that WebSphere Data Interchange must check the BLKLEN field to determine the software version and release being used.

ENVLDELAY

   Indicates whether functional acknowledgments must not be enveloped.

   Y   Does not envelope functional acknowledgments

   (other)   Envelopes functional acknowledgments

DUPTRAN

   Indicates how to process duplicate interchanges. DUPTRAN is an input field on the first call of a session and establishes if duplicate interchanges are errors. On all other requests, DUPTRAN is an output field. Valid values are:
   N   Does not process duplicate interchanges but considers them as errors. If a duplicate interchange is received, the translator issues message TR0211 and returns to the application with an interchange level error (extended return code value of 5).
   Y or (other)
            Processes duplicate interchanges and returns transactions flagged as duplicate transactions. **Y** is the recommended value.

SCOPE

   Indicates the level of recovery required:
   E   Issues a database COMMIT on the completion of every interchange. Applies only when enveloping is performed.
   (other)   Issues a database COMMIT at the start of every transaction.

   Note: This field contains values that affect the recovery scope during the session. For more information, see "Send Recovery Scope" on page 469.

INMEMTRANS

   Applies only if the value of SCOPE is **E**. Indicates the maximum number of transactions that are maintained in virtual storage before any database updates are attempted. Database updating occurs when the value in this field, or the end of the interchange, is reached.

   This value is important in an environment where multiple application programs are requesting translation services concurrently. The higher you set the value for this field, the more concurrency can be achieved. For more information, see "Translator Control Block (TRCB)" on page 642.

> **Note:** The value set in this field affects the recovery scope during the current session. For more information, see "Send Recovery Scope" on page 469.

FILEID The ddname of the file containing the interchanges to be processed. If you specify a value in this field, the value in `REQID` is ignored.

REQID A member in the mailbox (requestor) profile (REQPROF). The `Receive file name` field in the mailbox (requestor) profile identifies the file containing the interchanges to be processed. This field is required only if `FILEID` is not specified.

MRREQID

If the interchanges being deenveloped have not been recorded in the Management Reporting statistics database, this field identifies the requestor ID for which statistics must be updated. Applies only if the interchanges were not received using the Communication service receive function.

FUNACKFLE

If functional acknowledgments are being enveloped, the ddname of the file to which the transaction data is written when the interchange is complete. If you do not specify this keyword, the ddname specified in the `Trans data queue` field of the network profile (NETPROF) is used.

BATCHID

The batch ID of a group of transactions. The Transaction Store creates an alternate key using the batch ID. Other than using the transaction handle, searching on batch ID is the quickest way to retrieve a transaction from the Transaction Store during the selection process. The default is `DDHHMMSS`, where `DD` is the current day of the month, and `HHMMSSL` is the current time.

> **Note:** This field is checked with each call to the translator. The value in this field when the last call for transaction (TS) is made will be associated with the transaction.

TRXLIFE

The amount of time that a transaction remains in the Transaction Store before it is eligible for purging. The default is 30 days.

> **Note:** This field is checked with each call to the translator and if you do not specify a value, the default value is used. The value that exists when the last call for transaction (TS) is made will be associated with the transaction.

IMGLIFE

The amount of time that a transactions image (that is, the EDI standard data produced) remains in the Transaction Store. The default is **30** days.

> **Note:**
> 1. This field is checked with each call to the translator. The value that exists when the last call for transaction (TS) is made will be associated with the transaction.
> 2. This field is not currently used.

ERRFILTER

Indicates which error codes to filter out during this session. The values specified here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more information, see "Error filtering" on page 22.

MAPCHAIN

Indicates whether mapchaining is in effect. Valid values are:

Y   Translates the current transaction again (using the map identified in the DIMAPCHAIN variable.

(other)   Translates the next transaction.

FORCETEST

Indicates whether the translate process is forced to select only a test usage/rule. If you specified a value of **Y** for this keyword on the DEENVELOPE command, you must also specify a **Y** on the TRANSLATE TO APPLICATION command to select only deenveloped transactions.

TRIDB initialization for translate-file-to-application:

BLKLEN

The size of the data block, including the BLKLEN field. The minimum size is 32000 bytes.

RESERVED

Binary zeros

TRODB initialization for translate-file-to-application:

BLKLEN

The size of the data block, including the BLKLEN field. The minimum size is 32000 bytes.

RESERVED

Binary zeros.

First call for transaction (TF)

You must iniitalize certain fields before the first call for a transaction is issued.

For the TRIDB:

RAWDATA

Indicates whether the translator fills in the values for the internal trading partner ID and the record ID before returning the structure to your application.

Y   Fills in the values for the internal trading partner ID and record ID.

N   Does not fill in the values for the internal trading partner ID and record ID.

HOLDFLAG

Indicates whether the transaction is placed in HELD status when added to the Transaction Store. A transaction in HELD status is not available for any other activity until you release it.

Y   Places the transaction in hold status.

> **N (or other)**
>> Does not place the transaction in hold status.

For the TRODB:
BLKLEN
>> The length of the TRODB, including this field. If you are using the same
>> TRODB for all calls, initialize this field only once.

RESERVED
>> Zeros. If you are using the same TRODB for all calls, initialize this field only
>> once.

When initialization is complete, the translator is invoked with the following API request:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

On the first call to the translator for the session, the file containing the interchanges
(identified by the `FILEID` or `REQID` field) is opened and the first interchange in the file is
located. The entire interchange is read into virtual storage and validated. The
interchange sender ID and sender qualifier are extracted from the interchange header.
These values are used to determine which trading partner sent the data. See "Locating
sending trading partner profile members" on page 529 for an explanation of the search
for a trading partner.

On the first call to the translator for a transaction, the next transaction in the current file
is located (which might involve finding the next interchange) and the
transaction/message ID value is extracted from the transaction header. At this point, the
translator attempts to locate a map and usage/rule. Your EDI administrator must
establish the necessary maps and usages/rules before API requests are made. For
more information, see "Locating sending and receiving trading partner profile members"
on page 530 for a description of the fields used and the search to locate a transaction.

Numerous errors can occur at this point. If an error occurs, the return code and
extended return code for that error are posted in the `ZCCBRC` and `ZCCBERC` fields of
the CCB. With each call, your program must check the `TRXACCEPT` and `TRABORT` fields
for error information.

TRXACCEPT
>> Indicates whether the transaction had an acceptable translation. If this value is
>> not **Y** and your program continues processing, the next call made to the
>> translator is considered the first call for the next transaction.

TRABORT
>> Indicates whether an error was so severe that the translator could not continue
>> processing.
>>
>> If this value is **Y** and your program continues processing, the next call made to
>> the translator is considered the first call for the session. If the value in the
>> `FILEID` or `REQID` are not changed for the next call, continued processing might
>> result in an infinite loop because the same file is processed again from the
>> beginning and generates the same error.

Transaction TRCB Fields (TF):: The tables in this section describe the fields returned on the first call for a transaction. Such a large quantity of information is returned on the first call for a translate-file-to-application request that multiple tables are used to show the TRCB fields returned. There is a description of the data in the table at the beginning of each table.

The TRCB fields described in Table 142 provide information on basic attributes of the transaction. This information is about the transaction side of the processing rather than the application side.

Table 142. TRCB fields providing information on basic transaction attributes

Field name	Description
TRNID	The standard transaction or message ID for the transaction/message received.
TEST	The type of transaction: **I** Information **P** Production **T** Test
MAPKEY	The map used to translate the EDI standard data.
TPNICK	The trading partner nickname associated with the transaction.
DUPTRAN	Indicates whether this transaction has been received before: **Y** Duplicate transaction **(other)** New transaction
TRXACCEPT	Indicates that translation was acceptable and application data returned in the TRODB, unless the `EJECT` field has a value of **Y**: **Y** This transaction had an acceptable translation. **(other)** This transaction did not translate due to missing or invalid information.
TRABORT	Indicates whether an error occurred that was so severe the translator stopped processing: **Y** The error was so severe that translation was halted. **(blank)** No errors occurred.
TSKEY	The key value assigned to the transaction in the Transaction Store. This is called the transaction handle and has a format of `YYYYMMDDHHMMSSxxnnnn`. It is returned as a 10-byte packed value in this field, which is how it is stored in the database.
TSKEYU	The key value assigned to the transaction in the Transaction Store. This is the same value as the `TSKEY` field, except that this field contains an unpacked 20-byte value.
EJECT	Indicates whether the end of the transaction has been reached. See "Partial structures (TA)" on page 499 for related considerations. **Y** This transaction is complete. No data is returned and the next call is treated as a first call for transaction. **(other)** More data remains for the current transaction. The next call will return the next structure or an `EJECT` value of **Y**.
ERRNUM	The total number of errors flagged during data translation.
ERRCDES	An array of the first 10 different errors flagged during data translation. For more information, see "Translator Error Codes" on page 670.

The TRCB fields described in Table 143 are involved in the application side of the processing rather than the transaction side.

Table 143. Application related fields in TRCB

Field name	Description
ATFID	The data format ID associated with this transaction.
ATSID	The name of the structure being returned on this call. The data contained in the structure is in the TRODB.
APPFILE	The ddname name for the application file to which the application data must be written. This value is taken from the data format definition but can be overridden in the map receive usages/rules record.
APTYPE	The type of file represented by APPFILE. Applies only to CICS and MQ (which is supported in both z/OS and CICS). Valid values are: **MQ** MQSeries queue profile member name **PG** Program identified by APPFILE **TD** Transient data queue name identified by the first 4 characters of APPFILE **TM** Temporary storage queue (main) identified by APPFILE **TS** Temporary storage queue (auxiliary) identified by APPFILE **TX** CICS transaction code identified by the first 4 characters of APPFILE For z/OS, if you do not specify this keyword, this field is ignored; instead, the default is the ddname of a sequential file. For CICS, the default is **TS**.
INTPID	The internal trading partner ID taken from the map receive usages/rules.
APPCTLNUM	See `XACFIELD`.
XACFIELD	The application control value. The application control value is defined by the application field with an **AC** data type or by the fields that were assigned when the map was created. This value uniquely identifies the transaction to the application.
RAWDATA	Indicates whether the raw data processing was performed. Applies only if raw data processing was requested by setting the `RAWDATA` field to **Y** on the first request. With raw data processing, the translator sets the record ID and internal trading partner ID field values. The format of data written to the application file by the application program is under the control of the application program. All the necessary data for C and D records is available, but need not be used if raw data output is requested. **Y** Raw data processing was performed. **N** Raw data processing was not performed because the data format did not have raw data specifications.

The TRODB contains the application data. For each call to the translator, a single structure (record) of application data is returned. The type of structure returned is indicated by the `ATSID` field of the TRCB. Data is returned only if the transaction was acceptable (`TRXACCEPT` is **Y**) and the transaction is not complete (`EJECT` value is **Y**). See "Partial structures (TA)" on page 499 for special considerations.

The TRODB fields described below are related to application data:

DATALEN

The number of characters of data returned in the `DATA` field.

DATA   The application data with the format defined by the data format definition for the structure (`ATSID` field).

The TRCB fields described below are related to functional acknowledgments:

FABUILT

The envelope type of the functional acknowledgment. Valid values are:

E	UNB/UNZ
I	ICS/ICE
T	STX/END
U	BG/EG
X	ISA/IEA
G	The acknowldegement does not have an interchange header.
S	The acknowledgement was written to the Transaction Store, but was not enveloped.

FARC   The return code from the translator for the functional acknowledgment translation performed during deenveloping. For more information about return codes, refer to .

FAERC   The extended return code from the translator for the functional acknowledgment translation performed during deenveloping. For more information about return codes, refer to .

The TRCB fields described in Table 144 are associated with the interchange to which the current transaction belongs. These fields are established when the first transaction for the interchange is processed. They remain constant for all transactions in this interchange.

Table 144. Interchange header/trailer fields in TRCB

Field Name	Description
DSNAME	The physical data set name from which transactions are processed.
QTPNICK	The trading partner nickname for which the last transaction was processed.
QSIZE	The total number of bytes that were in the interchange, stored as a 4-byte binary value.
QBT	The character representation of the `QSIZE` field.
ENVTYPE	The type of envelope type received.
IHCTL	The interchange control number assigned to the current interchange. This value is returned in this field as a right-justified value with leading zeros. Applies to version 2.1 or earlier.
IHXCTL	The interchange control number assigned to the current interchange. This value is returned in this field as a right-justified value with leading zeros. Applies to version 3.1 or later.
ISYNTAXID	The interchange syntax ID for envelope types **E** and **T**.
ISYNTAXVER	The interchange syntax version for envelope types **E** and **T**.
ISIDQUAL	The interchange sender ID qualifier for envelope types **E**, **I**, and **X**.

Table 144. Interchange header/trailer fields in TRCB (continued)

Field Name	Description
ISID	The interchange sender ID (interchange header for data type **IS**).
ISENDNAME	The interchange sender name for envelope types **U** and **T**.
IREVROUT	The interchange reverse routing for envelope type **E**.
IRIDQUAL	The interchange receiver ID qualifier for envelope types **E**, **I**, and **X**.
IRID	The interchange receiver ID (interchange header for data type **IR**).
IRECVNAME	The interchange receiver name for envelope types **U** and **T**.
IROUTEADDR	The interchange routing address for envelope type **E**.
IDATE	The interchange date (interchange header for data type **DT**).
ITIME	The interchange time (interchange header for data type **TM**).
IVERREL	The interchange version and release (interchange header for data types **VR** or **LV**).
ISPW	The interchange password (interchange header for data type **PW**).
IAPREF	The application reference (interchange header for data type **AP**).
ISTDID	The interchange standard ID for envelope types **I** and **X**.
IPRIOR	The interchange priority code for envelope types **E** and **T**.
ICOMMAGREE	The interchange communication agreement for envelope types **E** and **T**.
NEWENV	Indicates whether this transaction is the first transaction of an interchange. If you want a copy of the interchange header, you can use a function code value of **1** to obtain an exact image. For more information, see "Retrieve interchange header API" on page 533. **Y**　　　Starts a new interchange **(other)**　　Does not start a new interchange
GRPNUM	The total number of groups processed so far in the current interchange, stored as a 4-byte binary value. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed. See `IGT` for the character representation of this field.
TRNNUM	The total number of transactions processed so far in the current interchange, stored as a 4-byte binary value. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed. See `ITT` for the character representation of this field.
SEGNUM	The total number of segments processed so far in the current interchange, stored as a 4-byte binary value. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed. See `IST` for the character representation of this field.
ESIZE	The total number of bytes processed so far in the current interchange, stored as a 4-byte binary value. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed. See `IBT` for the character representation of this field.

Table 144. Interchange header/trailer fields in TRCB (continued)

Field Name	Description
IGT	The character representation of GRPNUM. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.
ITT	The character representation of TRNNUM. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.
IST	The character representation of SEGNUM. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.
IBT	The character representation of ESIZE. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed.

The TRCB fields described in Table 145 are associated with the group to which the current transaction belongs.

Table 145. Group header/trailer fields in the TRCB

Field name	Description
GHCTL	The group control number assigned to the current group. The value returned in this field is right-justified with leading zeros. This field value is established when the first transaction for the group is processed. Applies to version 2.1 or earlier.
GHXCTL	The group control number assigned to the current group. The value returned in this field is right-justified with leading zeros. This field value is established when the first transaction for the group is processed. Applies to version 3.1 or later.
GSIDQUAL	The group sender ID qualifier for type **E** envelope groups.
GSID	The group sender ID (group header for data type **AS**). This value in this field is established when the first transaction for the group is processed.
GRID	The group receiver ID (group header for data type **AR** data type). This value in this field is established when the first transaction for the group is processed.
GRIDQUAL	The group receiver ID qualifier for type **E** envelope groups.
GDATE	The group date (group header for data type **DT**). This value in this field is established when the first transaction for the group is processed.
GTIME	The group time (group header for data type **TM**). This value in this field is established when the first transaction for the group is processed.
GAPW	The group password (group header for data type **PW**). This value in this field is established when the first transaction for the group is processed.

Table 145. Group header/trailer fields in the TRCB (continued)

Field name	Description
GVER	The group version (group header for data type **VR**). This value in this field is established when the first transaction for the group is processed.
GREL	The group release (group header for data type **LV**). This value in this field is established when the first transaction for the group is processed.
GRESPAGENCY	The group responsible agency code for types **E**, **U**, and **X** envelopes.
NEWGRP	Indicates whether this transaction is the first transaction of an interchange. If you want a copy of the group header, you can use a function code value of **2** to obtain an exact image. For more information, see "Retrieve group header API" on page 533. **Y** Starts a new interchange **(other)** Does not start a new interchange
TRNGRP	The total number of transactions processed so far in the current group, stored as a 4-byte binary value. This value in this field changes as transactions in the group are processed, indicating how much of the group has been processed. See GTT for the character representation of this field.
GTT	The character representation of TRNGRP. This value in this field changes as transactions in the group are processed, indicating how much of the group has been processed.

The TRCB fields described in Table 146 are associated with the transaction header and trailer for the current transaction.

Table 146. Transaction header/trailer fields in the TRCB

Field name	Description
NEWTRN	Indicates whether a transaction header is available. If you want a copy of the transaction header, you can use a function code value of **3** to obtain an exact image. For more information, see "Retrieve transaction header API" on page 534. **Y** A transaction header is available **(other)** A transaction header is not available
LASTINENV	Indicates whether this transaction is the last transaction in the current interchange.
THCTL	The transaction control number assigned to the current transaction. The value is returned in this field as a right-justified value with leading zeros. Applies to version 2.1 or earlier.
THXCTL	The transaction control number assigned to the current transaction. The value is returned in this field as a right-justified value with leading zeros. Applies to version 3.1 or later.
SEGTRN	The total number of segments in the current transaction, stored as a 4-byte binary value. See TST for the character representation of this field.

Table 146. Transaction header/trailer fields in the TRCB (continued)

Field name	Description
TTC	The current transaction or message ID value. See also `TRNID` on 482.
TVER	The transaction version (transaction header for data type **VR**).
TREL	The transaction release (transaction header for data type **LV**).
TST	The character representation of `SEGTRN`.

Subsequent calls (TF/TA)

The following activities take place on the first call for a transaction (TF):

- The next transaction in the file is located or the requested transaction is retrieved from the Transaction Store.
- The map associated with the transaction is found.
- The translation from EDI standard format into application structures takes place.
- The application structures are sorted into the order defined by the data format definition.
- The first application structure is returned to the calling program in the TRODB.

If the first call for a transaction was successful, the `TRXACCEPT` field has a value of **Y** and the application continues to call the translator until the `EJECT` field has a value of **Y**. The application uses the following call:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The parameters on this call match those on the first call for a transaction. For each call made, the next application structure is returned in the TRODB and the name of the structure is placed in the `ATSID` field.

Last call for transaction (TF)

The translator signals that the last call for a transaction has just been made by setting the `EJECT` field to a value of **Y**. No data accompanies this call. It signals the end of the current transaction. The next call is interpreted as the first call for the next transaction.

At this point, your application might perform application-related processing relative to the end of the transaction, including updating the application databases. For transaction level recovery (`SCOPE` is not **E**), the translator issues a COMMIT request on the next call.

If this transaction is the last transaction for an interchange, the `LASTINENV` field must contain a value of **Y**.

Last call of session (TF)

The translator signals the last call for a session through the CCB return codes. A `ZCCBRC` value of **4** and a `ZCCBERC` value of **1** indicate that all the data from the current file has been processed and that the translator has terminated. If there is another file that your application can process, cycle back to the first call of the session and set the `REQID` or `FILEID` field to specify the next file to be processed.

Translate-to-application processing considerations (TA)

The following sections describe special considerations for translate-to-application processing.

Partial structures (TA)

After a transaction has been translated, the application data produced is returned to the application program in the TRODB. The data is returned one structure at a time (that is, one structure for each API request made).

When all the data for a transaction has been returned, the next request results in an EJECT field with a value of **Y**, indicating that the transaction is complete. The TRODB can be no smaller than 32000 bytes, but it can be as large as you like. If a structure exceeds the size of the TRODB, as much data as can fit in the TRODB is returned and the EJECT field has a value of **X**.

The REQSIZE field contains the total structure size. When the EJECT field is returned with a value of **X**, do not change it. Subsequent calls to the translator returns the remainder of the structure. The translator continues to return a value of **X** in the EJECT field until the entire structure has been returned to the application. The EJECT field is then changed to a value of **D**.

If you do not want the remainder of the structure returned, you can set the value of the EJECT field to blank. The translator returns data for the next structure, rather than the residual data for the current structure.

Clustered transactions (TA)

During translate-to-standard requests, the application controls transaction clustering by setting the BNDLFLAG field in the TRCB. During translate-to-application or deenvelope requests, there is no control provided for clustering transactions as they are taken from the interchange. WebSphere Data Interchange automatically clusters all the transactions for a UN/TDI interchange, but does not cluster transactions from any other interchange type.

If the WebSphere Data Interchange Utility is used to deenvelope and translate a UN/TDI interchange, and if any transaction in the interchange fails to translate successfully, no data for any transaction in that interchange is returned to the application. The all or nothing aspect of processing UN/TDI interchanges is controlled by the application requesting the translation. The WebSphere Data Interchange Utility discards all data in a cluster if there is an error for any transaction in the cluster. However, because data for each transaction in the cluster is returned by the translator, your application might perform differently.

WebSphere Data Interchange also forces an interchange recovery scope (SCOPE value of **E**) when processing a UN/TDI interchange. The interchange recovery scope is forced by the translator and cannot be changed by the application program.

Enveloping services

Before you can send a transaction to a trading partner, and before a trading partner can send a transaction to you, the transaction must go through an enveloping operation. Enveloping is necessary so networks can identify the trading partner who must receive the data and so the translators can identify and verify the type of data being received. The three layers of enveloping are:

- Interchange layer
- Group layer
- Transaction layer

Each layer contains a header segment and a trailer segment. The data in these segments identifies:

- The trading partners involved (both sender and receiver)
- The applications or departments within the trading partner organizations involved (both sending and receiving)
- The exact nature of the data that is being exchanged

Each layer of enveloping has a specific purpose as described in the following sections.

Each header and trailer segment contains the following fields to help the receiving translator verify that a complete and consistent interchange has been received:

- The control numbers in the header and trailer that must match.
- A control count that indicates the number of items that must be present at the next level. For example, the control count in the interchange trailer indicates how many groups must be present; the control count in the group trailer indicates how many transactions must be present; and the control count in the transaction trailer indicates how many segments must be present in the transaction.

The sender must set these values based on the interchange created, and the receiver must verify that the data received agrees with the information contained in the header and trailer segments. These enveloping header and trailer segments are called *control* or *service* segments. Just as there are EDI standards for transactions, there are EDI standards for the service segments. WebSphere Data Interchange supports the following EDI enveloping standards:

- EDIFACT envelope standard - envelope and profile ID is **E**.

UNA	Delimiter segment
UNB	Interchange header
UNG	Group header
UNH	Transaction header
UNT	Transaction trailer
UNE	Group trailer
UNZ	Interchange trailer

- ICS envelope standard - envelope and profile ID is **I**

ICS	Interchange header
GS	Group header
ST	Transaction header
SE	Transaction trailer

    GE   Group trailer

    ICE   Interchange trailer

- UN/TDI envelope standard - envelope and profile ID is **T**
      SCH   Delimiter segment
      STX   Interchange header
      BAT   Group header
      MHD   Transaction header
      MTR   Transaction trailer
      EOB   Group trailer
      END   Interchange trailer

  Note: **BAT** and **EOB** are accepted by WebSphere Data Interchange during receive processing but are never generated by WebSphere Data Interchange.

- UCS envelope standard - envelope and profile ID is **U**
      BG   Interchange header
      GS   Group header
      ST   Transaction header
      SE GE EG

              Transaction trailer
      GE   Group trailer
      EG   Interchange trailer

- X12 envelope standard - envelope and profile ID is **X**
      ISA   Interchange trailer
      GS   Group header
      ST   Transaction header
      SE   Transaction trailer
      GE   Group trailer
      IEA   Interchange trailer

Interchange layer

Networks use the interchange header to identify both the trading partner that is sending the data, and the trading partner that must receive the data. The network uses the sender ID in the interchange header for routing error messages and network acknowledgments for the interchange. WebSphere Data Interchange uses the interchange header on the receiving end to determine:

- The trading partner, which in turn, is used to determine the map needed for translation
- The destination for any functional acknowledgments that are generated

A control number in the interchange header uniquely identifies this interchange between the sender and receiver. The control number in the interchange trailer must match the control number in the header. A count field in the interchange trailer identifies the number of groups and transactions contained in the interchange.

The interchange layer is optional in WebSphere Data Interchange when GS/GE segments are used at the group level. When the interchange layer is not present, the

GS02 and GS03 values identify the trading partners involved. GS02 must contain the trading partner nickname of the sender, and GS03 must contain the trading partner nickname of the receiver.

Group layer

The group layer is required for the ICS, UCS, and X12 enveloping standards but is optional for EDIFACT and UN/TDI. The layer is optional in some EDI standards and mandatory in others because of differences in how the standards define functional acknowledgments. For X12, UCS, and ICS, the functional acknowledgments (997 or 999 transactions) are defined at the group layer, and it is the group is being acknowledged. However, for EDIFACT and UN/TDI, acknowledgments (CONTRL message) are defined at the interchange level, and the interchange is being acknowledged. As a result, the group layer becomes optional.

The group layer identifies the sending and receiving applications or divisions within the trading partner organization. As a result, all transactions and messages that have a similar purpose or destination can be grouped together, and routed to the proper destination based on the values in the group header.

WebSphere Data Interchange uses the application sender ID and receiver ID values from the group header to locate the map needed to translate a transaction in the group. For more information, see "Locating sending and receiving trading partner profile members" on page 530. The group header also contains a control number that must be unique in the interchange. However, if 997 or 999 functional acknowledgments are being generated, the group control number must be unique for the interchange sender/receiver combination. In this case, the group control number serves the same purpose as the interchange control number. This is one reason that interchanges using GS and GE as the functional group header and trailer can be sent without the interchange level of enveloping (if the networks involved allow it). The group trailer contains a control number that must match the control number in the group header, and a count field that identifies the number of transactions in the group.

Transaction layer

The transaction layer identifies the transaction that immediately follows. The transaction ID (810, 850, 856, and so on) or message ID (ORDERS, INVOICE, CONTRL, and so on) is in the transaction header. This is the final piece of information used by WebSphere Data Interchange to locate a map for translating the transaction. The transaction header contains a control number that must be unique in the group (if groups are being used), or unique in the interchange (if groups are not being used). The transaction trailer also contains a control number that must match the control number in the header and a count field that identifies the number of segments included in the transaction. This number must match the actual number of segments received or processing is stopped and an error message is generated.

Enveloping service

The enveloping service provides the following two functions:
- An enveloping function that extracts transactions from the Transaction Store and builds transaction headers and trailers, organizing the transactions into groups, and

the groups into interchanges. The enveloping function adds the EDIVTSEV, EDIVTSGP and EDIVTSTU records to the Transaction Store database.

*   A deenveloping function that locates interchanges in a file, parses the interchange into groups and transactions, and adds the transactions to the Transaction Store. The deenveloping function also adds the EDIVTSEV, EDIVTSGP and EDIVTSTU records to the Transaction Store database.

These functions are described in detail in the following sections.

Envelope API

This section describes the details of the envelope API request. The envelope function extracts transactions from the Transaction Store and builds the transaction, group, and interchange headers and trailers that are needed to send those transactions to the appropriate trading partners.

Note: Enveloping does not have to be a separate API request. You can envelope transactions as they are translated. For more information, see the "Enveloping and sending" on page 446 and "Translate-to-standard API" on page 452.

The API that is described below is the same API that the WebSphere Data Interchange Utility uses internally when you issue any of the following PERFORM commands:
*   ENVELOPE
*   ENVELOPE AND SEND
*   REENVELOPE
*   REENVELOPE AND SEND

The envelope API is invoked once for each transaction that is enveloped. As each transaction is received, the enveloping function checks the transaction to determine whether it belongs to the current group and current interchange. If the transaction does not belong in the current group, the group is closed (a trailer is built) and a new group is started (a header is built). See "Fields that cause new groups to start" on page 515 for an explanation of the fields that will start a new group.

If the transaction does not belong in the current interchange, both the group and interchange are closed (trailers are built) and written to a file associated with the network, and a new interchange and group are started (headers are built). See "Fields that cause a new interchange to start" on page 515 for an explanation of the fields that will start a new interchange.

When WebSphere Data Interchange uses the envelope API, the transactions to be enveloped (dictated by the selection criteria) are first sorted to yield the fewest possible groups within the fewest possible interchanges.

If the transactions are not sorted first, an application program using the envelope API must allow for the fact that an interchange can be generated for each transaction. For example, if there are 10 transactions, 5 for trading partner A and 5 for trading partner B, and the sequence is A, B, A, B, A, B, A, B, A, B, 10 interchanges are created. Sorting the transactions first would yield 2 interchanges (A + B) rather than 10.

An application program using the envelope API must obtain a list of the transactions to envelope. Some ways to obtain a transaction list from the Transaction Store are:

- Set the application program that is adding the transactions to save the transaction handle values (and any other information needed for sorting) to a file that is then sorted and read by your enveloping application.

- Use the QUERY command to select transactions. The transaction handles for the selected transactions are written to the EDIQUERY file in transaction handle sequence. This does not result in optimal enveloping.

- Use the TRANSACTION DATA EXTRACT command to select transactions. As with the QUERY command, the transactions are written to the EDIQUERY file in transaction handle sequence, but this command provides complete transaction information that can be used to sort the transactions into a optimal sequence for enveloping.

Envelope API

The basic format of the API request to envelope a transaction is:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for this API request are defined in Table 147.

Table 147. Envelope API parameters

Parameter	Description
SNB	**ZSNBNAME** 　　　　TRANPROC **ZSNBPC** 　　　6
FCB	**ZFCBFUNC** 　　　215
TRCB	The translator control block. For more information, see "TRCB field descriptions" on page 647.
TRIDB	The input data block. This block is not used during enveloping operations, but a parameter must be provided. A minimum size of 16 bytes is sufficient during an enveloping operation. For more information on this block, see "TRIDB field descriptions" on page 673.
TRODB	The output data block. The output data block is used by WebSphere Data Interchange as a work buffer. This buffer must be at least 32000 bytes in length, but its maximum size must be the size of the maximum EDI standard segment (excluding the BIN segment) that is processed. For more information on this block, see "TRODB field descriptions" on page 675.

Initializing the envelope API

There are numerous fields defined in the control blocks for the enveloping API function whose values need only be established once. These values can be established before the first call and they do not have to be refreshed or changed before any other call. The

following tables describe the control blocks, the fields within the control blocks, and the considerations for initializing the envelope API.

SNB initialization for enveloping function:
ZSNBLL
> 32

ZSNBNAME
> TRANPROC

ZSNBPC
> 6 (all calls for enveloping services have six parameters)

FCB initialization for enveloping function:
ZFCBLL
> 4

ZFCBFUNC
> 215

TRCB initialization for enveloping function:

Table 148. TRCB initialization for enveloping function

Field name	Description
BLKLEN	**1536**.
BLKNME	**EDITRCB**
BLKTYPE	The format for the TRIDB and TRODB buffers. **H** Unlimited size **(other)** Limited to 32768 bytes
FASPEC	Indicates that **FUNACKFLE** is being used. Set this field to **Y** (If not provided, the value of **FUNACKFLE** will be ignored).
XPANDED	**Y**. Indicates that DataInterchange should check the `BLKLEN` field to determine the software version and release being used.
SCOPE	Indicates whether DataInterchange issues a database COMMIT after an interchange is completed or after a transaction is completed. **E** Interchange level recovery **(other)** Transaction level recovery **Note:** This value affects the recovery scope during the session.
INMEMTRANS	Applies only if the value of `SCOPE` is **E**. The maximum number of transactions that should be maintained in virtual storage before any database updates are attempted. Database updating occurs when the value in this field, or the end of the interchange, is reached. This value is important in an environment when multiple application programs request translation services at the same time. The higher you set the value for this field, the more concurrency can be achieved. For more information on this field, see "TRCB field descriptions" on page 647. **Note:** This value affects the recovery scope during the session.

Table 148. TRCB initialization for enveloping function (continued)

Field name	Description
FILEID	The ddname of the file where the transaction data is written when an interchange is complete. If you do not specify this field, the ddname specified in the `Trans data queue` field of the network profile (NETPROF) is used. This value does not have to be a constant for the entire session. However, if you want to change the value in this field, you must do so before the interchange is completed.
IUSEREXIT	The logical name of a user exit to be called by DataInterchange instead of writing the envelope to a file. This exit is used to retrieve an envelope from storage using the Get Envelope service once enveloping is complete. For more information on Get/Put Envelope exit processing, see Chapter 4, "Exit routines," on page 311.
IUSERAREA	A pointer to a user-defined area. The pointer is passed to the user exit defined in `IUSEREXIT` field.
ITPBREAK	Indicates whether a change in the internal trading partner ID starts a new interchange. Valid values are: **Y** Each interchange contains data for a single vendor number. **N or (other)** All data for a trading partner is included in a single interchange, regardless of vendor number.
ENVCHK	Indicates whether the translator checks the transaction's enveloping status before processing the transaction. See "Envelope versus reenvelope" on page 513 for more detailed information. Valid values are: **1** Verifies that the transaction has never been enveloped before. If the transaction has been enveloped before, the translator issues message TR0121 and returns to the application with a transaction level error (extended return code value of **3**). **2** Verifies that the transaction has been enveloped before. If the transaction has not been enveloped before, the translator issues message TR0121 and returns to the application with a transaction level error (extended return code value of **3**). **(other)** Does not verify the transaction's enveloping status.
ERRFILTER	Indicate which error codes to filter out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more detailed information, see "Error filtering" on page 22.
RAWDATAOUT	Indicates whether RAWDATA output is used for fixed-to-fixed mappings. **Y** Uses RAWDATA output **(other)** Uses C & D record output

Table 148. TRCB initialization for enveloping function (continued)

Field name	Description
FFILEID	The ddname of the file where the translated data for fixed-to-fixed translations is written. If this field is not specified, the ddname is formed from the concatenation of the `Application file name field` (from the target data format) and the `File suffix` field (from the trading partner profile). This value does not have to be a constant for the entire session. However, if you want to use this field, it must be set before an interchange is completed.

TRIDB initialization for enveloping function:

BLKLEN

> The size of the data block, including the `BLKLEN` field. A value of **16** is sufficient.

RESERVED

> Binary zeros.

Note: The input data block (TRIDB) is not used during an enveloping operation but must be specified in the parameter list.

TRODB initialization for enveloping function:

BLKLEN

> The size of the data block, including the `BLKLEN` field. The minimum length is 32000 bytes.

RESERVED

> Binary zeros.

Envelope transaction

This section describes the initialization required before each call, and the results returned to your program. Set the following TRCB fields before making the API request to envelope a transaction:

TSKEY The transaction handle for the transaction you want to envelope, formatted as a packed field in 10 bytes. The format of the handle is `YYYYMMDDHHMMSSxxnnnn`. If your program does not pack these values, initialize this field to all blanks or all binary zeros using the `TSKEYU` field.

TSKEYU

> The transaction handle for the transaction you want to envelope, formatted as a 20-byte character field. The format of the handle is `YYYYMMDDHHMMSSxxnnnn`. This field is only used if `TSKEY` does not have a value.

When initialization is complete, the enveloper is invoked with the following API request:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The transaction identified by the `TSKEY` or `TSKEYU` field is retrieved from the Transaction Store along with the other database records needed, such as the trading partner profile

member, and the send usage/rule and map. Everything that was needed to translate the transaction (except the control string) is also needed when the transaction is enveloped.

Numerous errors can occur at this point. If an error occurs, the return code and extended return code for the error are posted in the `ZCCBRC` and `ZCCBERC` fields of the CCB. With each call, your program must check the `TRXACCEPT` and `TRABORT` fields for error information. The fields returned in the TRCB are explained in the following tables.

Transaction-related fields (EV): The fields described in Table 149 provide some basic attributes of the transaction just processed. These fields are related to the transaction side of the processing.

Table 149. Transaction attribute fields in TRCB

Field name	Description
TRNID	The standard transaction or message ID for the transaction or message received.
TEST	The type of transaction. Valid values are: **I** Information **P** Production **T** Test
MAPKEY	The map used to translate the application data.
TPNICK	The trading partner nickname associated with the transaction.
TRXACCEPT	Indicates whether the transaction had an acceptable translation and that application data was returned in the TRODB (unless the `EJECT` field has a value of **Y**).
TRABORT	Indicates whether the error was so severe that the translator stopped processing.
TSKEY	The transaction handle for the transaction you want to envelope, formatted as a packed field in 10 bytes. The format of the handle is `YYYYMMDDHHMMSSxxnnnn`.
TSKEYU	The transaction handle for the transaction you want to envelope, formatted as a 20-byte character field. The format of the handle is `YYYYMMDDHHMMSSxxnnnn`.
ERRNUM	The total number of errors flagged during enveloping.
ERRCDES	An array of the first 10 different errors flagged during enveloping. For detailed error code information, see "Translator Control Block (TRCB)" on page 642.

Application-related fields (EV): The TRCB fields described in Table 150 are related to the application side of the processing.

Table 150. Application related TRCB fields

Field name	Description
ATFID	The data format definition ID associated with this transaction.
INTPID	The internal trading partner ID taken from the map receive usages/rules.

Table 150. Application related TRCB fields (continued)

Field name	Description
APPLTPID	The application trading partner.
APPCTLNUM	See `XACFIELD`.
XACFIELD	The application control value defined by the application field with data type of AC or by the fields that are assigned when the map is created. The application control value uniquely identifies the transaction to the application.
MAPCHAIN	Indicates whether mapchaining is in effect. **Y** Translates the current transaction again **(other)** Translates the next transaction

TRCB fields for Enveloping: Some fields in the TRCB are related to the current status of an interchange that is being created. The following tables describe the fields that relate to the following:
*   The interchange header and trailer
*   The group header and trailer
*   The transaction header and trailer

The TRCB fields described in Table 151 are related to the interchange header or trailer.

Table 151. Fields in TRCB related to the interchange header or trailer

Field name	Description
NEWENV	Indicates whether this transaction caused a new interchange to start. If you want a copy of the interchange header, you can use a function code value of **1** to obtain an exact image. For more information, see "Retrieve interchange header API" on page 533. **Y** Started a new interchange **(other)** Did not start a new interchange
IHCTL	The interchange control number assigned to the current interchange. The value returned in this field is right-justified with leading zeros. Applies to version 2.1 or earlier.
IHXCTL	The interchange control number assigned to the current interchange. The value returned in this field is right-justified with leading zeros. Applies to version 3.1 or later.
GRPNUM	The total number of groups in the interchange at the current time. This number is stored as a 4-byte binary value. See `IGT` for the character representation of this value.
TRNNUM	The total number of transactions in the current interchange at the current time. This number is stored as a 4-byte binary value. See `ITT` for the character representation of this value.
SEGNUM	The total number of segments in the current interchange at the current time. This number is stored as a 4-byte binary value. See `IST` for the character representation of this value.

Table 151. Fields in TRCB related to the interchange header or trailer (continued)

Field name	Description
ESIZE	The total number of bytes in the current interchange at the current time. This number is stored as a 4-byte binary value. See `IBT` for the character representation of this value.
ISYNTAXID	The interchange syntax identifier for envelope types **E** and **T**.
ISYNTAXVER	The interchange syntax version for envelope types **E** and **T**.
ISIDQUAL	The interchange sender ID qualifier for envelope types **E**, **I** and **X**.
ISID	The interchange sender ID (interchange header for data type **IS**).
IRECVNAME	The interchange receiver name for envelope type **T**. The application receiver code for envelope type **U** if UCS04 does not contain **IR**.
IROUTEADDR	The interchange routing address for envelope type **E**.
ISENDNAME	The interchange sender name for envelope type **T**. The application sender code for envelope type **U** if UCS03 does not contain **IS**.
IREVROUT	The interchange reverse routing for envelope type **E**.
IRIDQUAL	The interchange receiver ID qualifier for envelope types **E**, **I** and **X**.
IRID	The interchange receiver ID (interchange header for data type **IR**).
IDATE	The interchange date (interchange header for data type **DT**).
ITIME	The interchange time (interchange header for data type **TM**).
IVERREL	The interchange version and release (interchange header for data types **VR** or **LV**).
IGT	The character representation of `GRPNUM`.
ITT	The character representation of `TRNNUM`.
IST	The character representation of `SEGNUM`.
IBT	The character representation of `ESIZE`.
ISPW	The interchange password (interchange header for data type **PW**).
IAPREF	The application reference (interchange header for data type **AP**).
ISTDID	The interchange standard ID for envelope types **I** and **X**.
IPRIOR	The interchange processing priority for envelope types **E** and **T**.
ICOMMAGREE	The interchange communication agreement for envelope type **E**.

The TRCB fields described in Table 152 are related to the group header or trailer.

Table 152. Fields in TRCB related to group header or trailer

Field name	Description
NEWGRP	Indicates whether this transaction caused a new group to start. If you want a copy of the group header, you can use a function code value of **2** to obtain an exact image. For more information, see "Retrieve group header API" on page 533. **Y** Started a new group **(other)** Did not start a new group

Table 152. Fields in TRCB related to group header or trailer (continued)

Field name	Description
GHCTL	The group control number assigned to the current group. The value is returned in this field as a right-justified value with leading zeros. Applies to version 2.1 or earlier.
GHXCTL	The group control number assigned to the current group. The value is returned in this field as a right-justified value with leading zeros. Applies to version 3.1 or later.
TRNGRP	The total number of transactions in the current group at the current time. This number is stored as a 4-byte binary value. See GTT for the character representation of this value.
GSIDQUAL	The group sender ID qualifier for envelope type **E**.
GSID	The group sender ID (group header for data type **AS**).
GRIDQUAL	The group receiver ID qualifier for envelope type **E**.
GRID	The group receiver ID (group header for data type **R**).
GDATE	The group date (group header for data type **DT**).
GTIME	The group time (group header for data type **TM**).
GAPW	The group password (group header for data type **PW**).
GVER	The group version (group header for data type **VR**).
GREL	The group release (group header field with for data type **LV**).
GTT	A character representation of TRNGRP.
GRESPAGENCY	The group controlling agency for envelope type **E**. The group responsible agency for envelope types **I**, **U** and **X**.

The TRCB fields described in Table 153 are related to the transaction header or trailer

Table 153. TRCB fields related to transaction header or trailer

Field name	Description
NEWTRN	Indicates whether this transaction caused a new transaction to start. If you want a copy of the transaction header, you can use a function code value of **3** to obtain an exact image. For more information, see"Retrieve transaction header API" on page 534. **Y** Started a new transaction **(other)** Did not start a new transaction
THCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros. Applies to version 2.1 or earlier.
THXCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros. Applies to version 3.1 or later.
SEGTRN	The total number of segments in the current transaction stored as a 4-byte binary value. See TST for the character representation of this value.

Table 153. TRCB fields related to transaction header or trailer (continued)

Field name	Description
TTC	The current transaction or message ID value. See also `TRNID` on 482.
TVER	The transaction version (transaction header for data type **VR**).
TREL	The transaction release (transaction header for data type **LV**).
TST	The character representation of `SEGTRN`.

Queuing - TRCB Fields: The TRCB fields described in Table 154 are returned when an interchange is written. The value in the `EJECT` field indicates whether an interchange was written. The other fields in the table provide information about the interchange, the network, and the file receiving the interchange.

Table 154. TRCB fields returned when an interchange is written

Field name	Description
EJECT	Indicates whether an interchange was written to the file associated with the network. Valid values are: **Q** The interchange was written successfully. **E** The interchange was not written successfully. **Note:** If this field has a value of **E**, indicating an error occurred, the `QRC` and `QERC` fields contain values indicating which error occurred. The error is logged by WebSphere Data Interchange Communication services. The most likely error is that the ddname could not be opened.
QRC	The return code associated with the error.
QERC	The extended return code associated with the error.
DSNAME	The physical data set name to which the interchange is written. For CICS, this field contains the same value as the QDDNAME field, which is the name of the TS queue containing the interchange.
QNETID	The name of the network associated with the trading partner.
QPTTOPT	Indicates whether the network identified by QNETID is a point-to-point network.
QSRPGM	Indicates whether the network identified by QNETID has a network send and receive program associated with it. If no send and receive program is associated with a network, there is no need to attempt to send the interchange. (You might use a network without a send and receive program for creating interchanges that are sent to a trading partner using some other means.)
QDDNAME	The ddname to which the interchange is written. Does not apply for point-to-point networks. For CICS, this is the name of the TS queue to which the interchange is written.
QTPNICK	The trading partner nickname for which the interchange is built.
QSIZE	The total number of bytes in the interchange. This number is stored as a 4-byte binary value.
QBT	The character representation of `QSIZE`.

See"Sending transaction data" on page 514 for an explanation of items to consider if your application program is using the Communication services send API for sending the data just written to the file.

Last call of session (EV)

If the translator was called at least once and the `TRABORT` field does not have a value of **Y**, invoke the translator one last time with a termination function code. This signals the translator that the application is finished making enveloping requests. Upon receiving the request, the translator releases any resources that were acquired. If an interchange is active, the translator completes and writes the current interchange. See "End translation/enveloping API" on page 517.

Envelope processing considerations (EV)

The following sections list special considerations for envelope processing.

Envelope versus reenvelope

Both the WebSphere Data Interchange Utility and Transaction Store Facility provide a means of enveloping and reenveloping transactions. When the ENVELOPE function is used, only transactions in the Transaction Store that have never been enveloped are considered for enveloping. When the REENVELOPE function is used, only transactions in the store that have already been enveloped are considered for reenveloping.

The API services do not provide a similar distinction. The application program making the API request must distinguish whether a transaction is enveloped or reenveloped. However, you can set the `ENVCHK` field in the TRCB to indicate the intentions of the application (ENVELOPE or REENVELOPE), and the envelope service will verify that the status of the transaction matches the intentions of the application. The `ENVCHK` field has the following values and meanings:

1   Envelopes a transaction. If the transaction has ever been previously enveloped, attempting to envelope it again must be considered an error. The translator issues message TR0121 and returns a transaction level error (extended return code value of **3**) to the application.

2   Reenvelopes a transaction. If the transaction has not been previously enveloped, attempting to envelope it again must be considered an error. The translator issues message TR0121 and returns a transaction level error (extended return code value of **3**) to the application.

(other) Does not check the status of the transaction.

Clustered transactions (EV)

You can cluster (batch) transactions during translation. The Transaction Store Facility and WebSphere Data Interchange Utility ensure that any action performed against any member of a cluster is performed against all members of the cluster. If any member is enveloped, all members are enveloped. If any member is purged, all members are purged. This is a feature of the utilities and facilities and not a feature of the API. The enveloping service checks to make sure that the controlling transaction of a cluster is the first transaction that is enveloped. There is no check to ensure whether all members

are enveloped or that they are enveloped in any particular order, except to verify that the controlling transaction is enveloped first. For more information, see 458.

Sending transaction data

The value in the EJECT field in the TCB indicates when transaction data has been written to the file associated with the network. If you use the send transaction API (see "Send transactions and restart send transactions API" on page 544) to send the data, your program must determine whether the data is to be sent immediately or at the end of the translation or enveloping function.

Sending transaction data when written: Sending the transaction data when it is written is the easiest method because the application program does not have to be as aware of its environment as it does if delayed sending is used. When the EJECT field indicates that data is written, the DSNAME field contains the name of the data set to which the transaction data was written, and the QNETID field contains the network ID associated with the trading partner. You can move the value in QNETID to the NETID field in the CMCB and the value in DSNAME to the FILENAME field in the CMCB (setting DATATYP to **A**). Communications can then be invoked to send the data.

During a WebSphere Data Interchange session (between initialization and termination), the communications programs remember the names of data sets to which data has been written. The first time a data set is used, it is opened for output. Anything in the data set is overwritten with the new transaction data. However, on subsequent use, the same data set is opened for extend, meaning the first transaction data is not overlaid, and the new transaction data is written at the end of the data set. To clear the data set after the data is sent, set the CLRFILE field in the CMCB to a value of **Y**. If the file is not cleared after the data has been sent, interchanges might be sent multiple times resulting in duplicate records.

Notes:
1. You can use the QDDNAME field only if a point-to-point network is not being used. DSNAME works regardless of which network you are using.
2. You must supply a mailbox (requestor) profile ID on the communications request, and if multiple networks are being used, there must be a way to associate a requestor ID with a network.

Sending transaction data later: Deferring the sending of the data until translation or enveloping is complete is more complex than sending the data immediately. The application program must keep track of all data sets used and the networks associated with the data sets so each can be sent to the appropriate network.

Use the data set name as an identifier rather than the ddname, because multiple ddnames (such as QDATA and QDATAE) can be associated with the same physical data set. Using the ddname can result in data being sent twice. If the FILEID field is used and sending is deferred, the application must verify that all interchanges created are for a single network. If point-to-point networks are being used, all interchanges created must also be for a single trading partner.

Fields that cause a new interchange to start

During enveloping, transactions are checked against the current interchange to determine whether the transaction belongs in the current interchange or whether the interchange must be closed and a new one started. The following fields are checked for changes to determine if a new interchange must be started:

- EDI trading partner nickname
- Application trading partner nickname
- Trading partner network ID
- Envelope type
- Bundle status
- Any delimiter
- Decimal notation
- Release character
- Usage indicator
- Who is assigning transaction control numbers
- Internal trading partner ID, depending on the value of `ITPBREAK` in the TRCB
- Any of the following interchange override values in the TRCB
    - Syntax ID (ISYNTAXID)
    - Syntax version (ISYNTAXVER)
    - Sender ID qualifier (ISIDQUAL)
    - Sender ID (ISID)
    - Sender name (ISENDNAME))
    - Reverse routing (IREVROUT)
    - Receiver ID qualifier (IRIDQUAL)
    - Receiver ID (IRID)
    - Receiver name (RECVNAME)
    - Routing address (IROUTEADDR)
    - Password (ISPW)
    - Application reference (IAPREF)
    - Standard ID (ISTDID)
    - Priority (IPRIOR)
    - Communication agreement (ICOMMAGREE)
    - Version/release (IVERREL)

Fields that cause new groups to start

During enveloping, transactions are checked against the current group to determine whether the transaction belongs in the current group or whether the current group must be closed and a new one started. The following fields are checked for changes to determine if a new group must be started:

- Functional group ID assigned to the transaction
- Network security profile member being used
- Group encryption key name
- Group authentication key name
- Envelope profile member being used
- Application sender ID in the send usage/rule
- Application receiver ID in the send usage/rule
- Application password in the send usage/rule
- Any of the following override values in the TRCB
    - Application sender ID qualifier (GSIDQUAL)
    - Application sender ID (GSID)

- – Application receiver ID (GRID)
- – Application receiver ID qualifier (GRIDQUAL)
- – Group password (GAPW)
- – Group version (GVER)
- – Group release (GREL)
- – Group responsible agency (GRESPAGENCY)

Close and queue interchange API

You can use the close and queue interchange function to your application program direct control over the content and size of interchanges being created. In normal processing, the translator/enveloper continues to add transactions to the current interchange until there is a change in one of the fields that signals a new interchange. If your application has special requirements for the interchanges created, you can issue a close-and-queue envelope request at any time. If C and D records are being used as input to the WebSphere Data Interchange Utility, a Z1 record can be used to tell the utility to terminate the current interchange. When a Z1 record is read, the utility issues an API request to close and queue the current interchange. When this API request is issued, the current interchange is closed by adding a group trailer segment (if groups are being used) and an interchange trailer segment. The complete interchange is then given to the Communication service, which writes the interchange to the file associated with the network.

The basic format of the API request to close and queue the current interchange is:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for this API request are defined in Table 155.

Table 155. Parameters for the Close and queue interchange API

Parameter	Description
SNB	**ZSNBNAME** TRANPROC **ZSNBPC** 6
FCB	**ZFCBFUNC** **990** (close and queue the current interchange)
TRCB	The translator control block. This transaction control block is used in all requests. There are no field initialization requirements for this control block unless you have special COMMIT requirements. See "TRCB field descriptions" on page 647 for details.
TRIDB	The input data block. No data is required in this block.
TRODB	The output data block. This buffer must be at least 32000 bytes in length, but its maximum size must be the size of the largest standard segment (excluding the BIN segment) that will be processed. See "TRODB field descriptions" on page 675 for a general description of this block.

Queueing - TRCB Fields:

The TRCB fields described in Table 156 on page 517 are returned when an interchange is written. The value in the `EJECT` field indicates whether an interchange is written. The

other fields in the table provide information about the interchange, the network, and the file receiving the interchange.

Table 156. TRCB fields returned when an interchange is written

Field name	Description
EJECT	Indicates whether an interchange was written to the file associated with the network. Valid values are: **Q**　　The interchange was written correctly **E**　　The interchange was not written successfully **Note:** If this field has a value of **E**, the QRC and QERC fields have values indicating which error occurred. The error is logged by WebSphere Data Interchange communications services. The most likely error is that the ddname could not be opened.
QRC	The return code associated with the error.
QERC	The extended return code associated with the error.
DSNAME	The physical data set name to which the interchange was written. For CICS, this field has the same value as QDDNAME, which is the name of the TS queue containing the interchange.
QNETID	The name of the network associated with the trading partner.
QPTTOPT	Indicates whether the network identified by QNETID is a point-to-point network. Valid values are: **Y**　　The network is a point-to-point network **(other)**　The network is not a point-to-point network
QSRPGM	Indicates whether the network identified by QNETID has a network send and receive program associated with it. If there is no send and receive program associated with a network, there is no need to attempt to send the interchange. (You might use a network without a send and receive program for creating interchanges that are sent to a trading partner using some other means.) **Y**　　The network has a send and receive program. **(other)**　The network does not have a send and receive program.
QDDNAME	The ddname to which the interchange was written. Does not apply for point-to-point networks. For CICS, this is the name of the TS queue to which the interchange is written.
QTPNICK	The trading partner nickname for which the interchange is built.
QSIZE	The total number of bytes that are in the interchange, stored as a 4-byte binary value.
QBT	The character representation of QSIZE.

See "Sending transaction data" on page 514 for an explanation of items to consider if your application program uses the communications services send API for sending the data just written to the file.

End translation/enveloping API

If the translator was called at least once and the TRABORT flag does not have a value of **Y**, invoke the translator one last time with a termination function code. The termination function code signals the translator that the application has finished making

translation or enveloping requests. On receiving the request, the translator releases any resources that it acquired. If an interchange is active, the translator completes and writes the current interchange.

The basic format of the API request to end translation/enveloping   is:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for this API request are described in Table 157.

Table 157. Parameters for the End translation/enveloping API

Parameter	Description

SNB	**ZSNBNAME**
	        TRANPROC
	ZSNBPC
	        6

FCB	**ZFCBFUNC** 1000
TRCB	The same translator control block used in all other requests.
TRIDB	The input data block. No data is required in this block.
TRODB	The output data block. This buffer must be at least 32000 bytes in length, but its maximum size must be the size of the largest standard segment (excluding the BIN segment) that will be produced. See "TRODB field descriptions" on page 675 for a general description of this block.

If an interchange is active at the time of the termination request, certain field values in the TRCB indicate whether the interchange was written. See "Close and queue interchange API" on page 516 for an explanation of these fields.

Deenvelope API

This section describes the details of the deenvelope API request. Deenveloping extracts interchanges from a file and parses them to extract transactions to add to the Transaction Store. The syntax of the interchange or transaction is checked and functional acknowledgments are generated. Received acknowledgments are reconciled with the original transactions.

Note: Deenveloping does not need to be a separate API request. You can deenvelope and translate transactions at the same time. For more information, see "Receiving and deenveloping" on page 474 and "Translate-file-to-application API" on page 486.

The deenvelope API described in the following section is the same API used internally by the WebSphere Data Interchange Utility when you issue any of the following PERFORM commands:
- DEENVELOPE
- DEENVELOPE AND TRANSLATE
- RECEIVE AND DEENVELOPE
- RECEIVE AND TRANSLATE

Each time the deenveloping API is invoked, the next transaction from the file being processed is located and added to the Transaction Store. Sometimes locating the next transaction involves locating the next interchange in the file. Once an interchange is located, the complete interchange is read into virtual storage. A syntax check and validation are done on the service segments to verify that the interchange received has a valid format and is consistent with the specified format.

The basic format of the API request to deenvelope transaction from a file is:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for this API request are described in Table 158.

Table 158. Parameters for the deenveloping API

Parameter	Description

SNB	**ZSNBNAME**
	        TRANPROC
	ZSNBPC
	        6

FCB	**ZFCBFUNC** 214
TRCB	The translator control block. For more information on this block, see "Translator Control Block (TRCB)" on page 642.
TRIDB	The input data block. This buffer must be at least 32000 bytes in length, but its maximum size must be the size of the largest standard segment (excluding the BIN segment) that is received. See "Translator Input Data Block (TRIDB)" on page 672 for a general description of this block.
TRODB	The output data block. WebSphere Data Interchange uses the output data block as a work buffer. This buffer must be at least 32000 bytes in length. See "Translator Output Data Block (TRODB)" on page 674 for a general description of this block.

Initializing for deenvelope API

There are numerous fields in the control blocks defined for the deenvelope API function that must be established only once. These values can be established before the first call and they do not need to be refreshed or changed before any other call. Because the first call for a session also qualifies as the first call for a transaction, you must also follow the instructions in the next section, "Deenvelope transaction (DE). The control blocks, the fields within the control blocks, and the initialization considerations are described in the following tables.

SNB initialization for deenveloping:
ZSNBLL
        32
ZSNBNAME
        TRANPROC
ZSNBPC
        6 (number of calls for enveloping services)

FCB initialization for deenveloping:
ZFCBLL
> 4

ZFCBFUNC
> 214 (deenvelope)

TRCB initialization for deenveloping is defined in Table 159.

Table 159. TRCB initialization for deenveloping

Field name	Description
BLKLEN	**1536**.
BLKNME	**EDITRCB**.
BLKTYPE	The format for the TRIDB and TRODB buffers. **H** Unlimited size **(other)** Limited to 32768 bytes
XPANDED	**Y**. Indicates that WebSphere Data Interchange must check the `BLKLEN` field to determine the software version and release being used.
ENVLDELAY	Indicates whether functional acknowledgments must not be enveloped. **Y** Does not envelope functional acknowledgments **(other)** Envelopes functional acknowledgments
DUPTRAN	Indicates how to process duplicate interchanges. This field is only an input field on the first call of a session and establishes if duplicate interchanges are errors. On all other requests, this field is an output field. Valid values are: **N** Does not process duplicate interchanges but considers them errors. If a duplicate interchange is received, the translator issues message TR0211 and returns to the application with an interchange level error (extended return code value of 5). **Y** Processes duplicate interchanges and returns transactions flagged as duplicate transactions. **Y** is the recommended value.
SCOPE	Indicates whether interchange or transaction recovery level was requested. **E** Interchange level recovery **(other)** Transaction level recovery **Note:** This field contains values that affect the recovery scope during the session. For more information, see "Send Recovery Scope" on page 469.

Table 159. TRCB initialization for deenveloping (continued)

Field name	Description
INMEMTRANS	Applies only if the value of `SCOPE` is **E**. Indicates the maximum number of transactions that must be maintained in virtual storage before any database updates are attempted. Database updating occurs when the value in this field, or the end of the interchange, is reached. This value is important in an environment where multiple application programs request translation services at the same time. The higher you set the value for this field, the greater the concurrency that can be achieved. For more information on this field, see "Translator Control Block (TRCB)" on page 642. **Note:** This field contains values that affect the recovery scope during the session. For more information, see "Send Recovery Scope" on page 469.
FILEID	The ddname of the file that contains the interchanges to process. If this field is specified, `REQID` is ignored.
IUSEREXIT	The logical name of a user exit to be called by WebSphere Data Interchange instead of reading the input file. This exit is used when the envelope is to be stored into WebSphere Data Interchange through the Put Envelope service before deenveloping. For more information on the Get/Put Envelope Exit processing, see Chapter 4, "Exit routines," on page 311.
IUSERAREA	A pointer to a user-defined area. The pointer is passed to the user exit defined in IUSEREXIT field.
IUSERACCESS	Indicates whether the interchange is presented to the IUSEREXIT program in virtual storage or in a file. **M** Gives the interchange to the exit in virtual storage. Applies when the `IUSERTYPE` is **UE** (user exit). **F** Gives the interchange to the exit in a file. When you use this value, the interchange is first written to the TD queue file, and then the `IUSEREXIT` program is invoked.
IUSERTYPE	The type of user exit program specified in `IUSEREXIT`. **PG** IUSEREXIT is a program that must be linked to (EXEC CICS LINK in CICS). **UE** IUSEREXIT is a WebSphere Data Interchange user exit program defined in the User Exits profile
REQID	The requestor ID of a member in the mailbox (requestor) profile (REQPROF). This field is required only if `FILEID` is not provided.
MRREQID	If the interchanges being deenveloped have not been recorded in the Management Reporting statistics database, this value identifies the requestor ID for which statistics must be updated. This update is done only if the interchanges were not received using the Communication service receive function.

Table 159. TRCB initialization for deenveloping (continued)

Field name	Description
FUNACKFLE	If functional acknowledgments are being enveloped, this field contains the ddname of the file where the transaction data is written when an interchange is complete. If you do not specify this field, the ddname specified in the `Trans data queue` field of the network profile (NETPROF) is used. **Note: FASPEC** must be set to **Y**
TRXLIFE	The amount of time this transaction must remain in the Transaction Store before it is eligible for purging. The default is 30 days. **Note:** This field is checked with each call to the translator. The value contained in this field when the deenvelope transaction request is made will be associated with the transaction.
IMGLIFE	The length of time this transaction's image (the standard data produced) must remain in the Transaction Store. The default is 30 days. **Notes:** 1. This field is checked with each call to the translator. The value contained in this field when the last call for transaction (TS) is made will be associated with the transaction. 2. This field is not currently supported.
HOLDFLAG	Indicates whether this transaction must be placed on hold when added to the Transaction Store. A transaction in hold status is not available for any other activity until it is released. **Y** Holds the transaction **N or other** Does not hold the transaction. **N** is the recommended value. **Note:** This field is checked with each call to the translator. The value contained in this field when the deenvelope transaction request is made will be the value associated with the transaction.
ERRFILTER	Specifies which error codes to filter out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more information, see "Error filtering" on page 22.
FORCETEST	Indicates whether the deenvelope process is to be forced to select only a test usage/rule regardless of the value of the test indicator in the envelope. This flag is most useful when receiving test envelopes with no test indicators (such as the UCS BG). In this case, you can to force the translator to consider the envelopes to be tested and only look for test usages/rules. **Y** Forces test mode. If a test usage/rule is not found, an error is generated and the transaction is rejected. **(other)** Uses the test indicator from the envelope to determine the usage/rule to select. Envelopes with no test indicator are always considered production envelopes.

TRIDB initialization for deenveloping:

BLKLEN

> The size of the data block, including the `BLKLEN` field. The minimum size for this field is 32000 bytes.

RESERVED

> Binary zeros

TRODB initialization for deenveloping:

BLKLEN

> The size of the data block, including the `BLKLEN` field. The minimum value for this field is 32000 bytes.

RESERVED

> Binary zeros.

Deenvelope transaction

There are no initialization requirements before making the request to deenvelope a transaction, unless you want the next transaction to have different values than the current transaction for the `TRXLIFE`, `IMGLIFE`, or `HOLDFLAG` fields. Once initialization is complete, the translator is invoked with the following API request:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

On the first call to the translator for the session, the file containing the interchanges (identified in the `FILEID` or `REQID` field) is opened and the first interchange is located. The entire interchange is read into virtual storage and validated. The interchange sender ID and sender ID qualifier are extracted from the interchange header and used to determine which trading partner sent the data. For an explanation of the search done for a trading partner, see "Locating sending trading partner profile members" on page 529.

On the first call to the translator for a transaction, the next transaction in the file is located (this might involve finding the next interchange) and the transaction/message ID value is extracted from the transaction header. The translator attempts to locate a usage/rule and map. This search is done even though a translation does not take place. The usage/rule indicates the type of functional acknowledgment requested, and the map identifies the segments and data elements in the requested transaction.

In functional acknowledgments, only mapped segments and data elements are validated. Your EDI administrator must establish the transactions and usages/rules before API requests are made. For an explanation of the fields used and the search done to locate a transaction, see "Locating sending and receiving trading partner profile members" on page 530. The deenveloping process still occurs without a transaction or usage/rule, but functional acknowledgments are not generated.

Numerous errors can occur at this point. If an error occurs, the return code and associated extended return code are posted in the `ZCCBRC` and `ZCCBERC` fields of the CCB. With each call, your program must check the `TRXACCEPT` and `TRABORT` fields for error information.

Transaction TRCB Fields (DE): The fields in the following tables are returned on the deenvelope transaction request. Such a large amount of information is returned that multiple tables are used to show the TRCB fields.

The TRCB fields described in Table 160 are related to the transaction side of the processing rather than the application side.

Table 160. TRCB fields related to the transaction

Field name	Description
TRNID	The standard transaction or message ID for the transaction or message received.
TEST	The type of transaction. **P** Production **T** Test **I** Information
MAPKEY	The map used to translate the standard data. Applies only if a usage/rule was found.
TPNICK	The trading partner nickname associated with the transaction.
DUPTRAN	Indicates whether this transaction is a duplicate, and whether the interchange being received has been received before. **Y** The transaction is a duplicate. **(other)** The transaction is not a duplicate.
TRXACCEPT	Indicates whether the transaction had an acceptable translation. **Y** The transaction was deenveloped successfully. **(other)** The transaction was not deenveloped because something was missing.
TRABORT	Indicates whether the error was so severe that the translator stopped processing. **Y** The translator has stopped processing. **(other)** The translator continued processing. **Y** **(other)**
TSKEY	The key value assigned to the transaction in the Transaction Store. This is called the transaction handle and has a format of YYYYMMDDHHMMSSxxnnnn. It is returned as a 10-byte packed value in this field, and stored in the database.
TSKEYU	The key value assigned to the transaction in the Transaction Store. The same value as the TSKEY field, but TSKEYU is an unpacked (character) 20-byte value.
ERRNUM	The total number of errors flagged during data processing.
ERRCDES	An array of the first 10 different errors flagged during data processing. See "Translator Error Codes" on page 670 for a list of errors.

The TRCB fields described in Table 161 on page 525 are related to the application side of the processing rather than the transaction side.

Table 161. Application-related fields in TRCB

Field name	Description
ATFID	The data format definition ID associated with this transaction. Applies only if a usage/rule was found.
INTPID	The internal trading partner ID (vendor number) taken from the map receive usage/rule. Applies only if a usage/rule was found.

The TRCB fields described in Table 162 are related to the functional acknowledgments.

Table 162. Fields related to functional acknowledgements

Field name	Description
FABUILT	The envelope type for the functional acknowledgment. Valid values are: **E** UNB/UNZ **I** ICS/ICE **T** STX/END **U** BG/EG **X** ISA/IEA **G** The acknowledgment does not have an interchange header. **S** The acknowledgment was written to the Transaction Store, but was not enveloped.
FARC	The return code from the translator for the functional acknowledgment translation done during deenvelope. For more information, refer to .
FAERC	The extended return code from the translator for the functional acknowledgment translation done during deenvelope. For more information, refer to .

The TRCB service segment fields (TF) described in Table 163 are associated with the interchange header and trailer to which the current transaction belongs. The first table describes the fields that are established when the first transaction for the interchange is processed and then remain constant for all the transactions in the interchange.

Table 163. Interchange header/trailer fields in TRCB

Field name	Description
DSNAME	The physical data set name from which transactions are being processed.
QTPNICK	The trading partner nickname for which the last transaction processed.
QSIZE	The total number of bytes in the interchange, stored as a 4-byte binary value. See `QBT` for the character representation of this value.
QBT	The character representation of the `QSIZE` field.

Table 163. Interchange header/trailer fields in TRCB (continued)

Field name	Description
ENVTYPE	The envelope type being received. Valid values are: **E** UNB/UNZ **I** ICS/ICE **T** STX/END **U** BG/EG **X** ISA/IEA
IHCTL	See `IHXCTL`.
IHXCTL	The interchange control number assigned to the current interchange. The value returned in this field is right-justified with leading zeros.
ISYNTAXID	The interchange syntax ID for envelope types **E** and **T**.
ISYNTAXVER	The interchange syntax version for envelope types **E** and **T**.
ISIDQUAL	The interchange sender ID qualifier for envelope types **E**, **I**, and **X**.
ISID	The interchange sender ID (interchange header for data type **IS**).
IRECVNAME	The interchange receiver name for envelope type **T**. The application receiver code for envelope type **U** if UCS04 does not contain **IR**.
IROUTEADDR	The interchange routing address for envelope type **E**.
ISENDNAME	The interchange sender name for envelope type **T**. The application sender code envelope type **U** if UCS03 does not contain **IS**.
IREVROUT	The interchange reverse routing for envelope type **E**.
IRIDQUAL	The interchange receiver ID qualifier for envelope types **E**, **I**, and **X**.
IRID	The interchange receiver ID (interchange header for data type **IR**).
IDATE	The interchange date (interchange header for data type **DT**).
ITIME	The interchange time (interchange header for data type **TM**).
IVERREL	The interchange version and release (interchange header for data type **VR** or **LV)**.
ISPW	The interchange password (interchange header for data type **PW**).
IAPREF	The application reference (interchange header for data type **AP**)
ISTDID	The interchange standard ID for envelope types **I** and **X**.
IPRIOR	The interchange processing priority for envelope types **E** and **T**.
ICOMMAGREE	The interchange communication agreement for envelope type **E**.

The values in the TRCB fields described in Table 164 on page 527 change as transactions are processed to indicate how much of the interchange has been processed.

Table 164. TRCB fields updated to indicate interchange processing

Field name	Description
NEWENV	Indicates whether the transaction is the first transaction of an interchange. If you want a copy of the interchange header, you can use a function code value of **1** to obtain an exact image. For more information, see "Retrieve interchange header API" on page 533. **Y** Starts a new interchange **(other)** Does not start a new interchange
GRPNUM	The total number of groups processed so far in the current interchange stored as a 4-byte binary value. See IGT for a character representation of this value.
TRNNUM	The total number of transactions processed so far in the current interchange stored as a 4-byte binary value. See ITT for a character representation of this value.
SEGNUM	The total number of segments processed so far in the current interchange stored as a 4-byte binary value. See IST for a character representation of this value.
ESIZE	The total number of bytes processed so far in the current interchange stored as a 4-byte binary value. See IBT for a character representation of this value.
IGT	The character representation of GRPNUM.
ITT	The character representation of TRNNUM.
IST	The character representation of SEGNUM.
IBT	The character representation of ESIZE.

The TRCB fields described in Table 165, Table 166 on page 528, and Table 167 on page 528 are associated with the group header and trailer to which the current transaction belongs. The first table describes the fields that are established when the first transaction for the group is processed and then remain constant for all the transactions in the group.

Table 165. Group header/trailer fields in the TRCB

Field name	Description
GHCTL	See GHXCTL.
GHXCTL	The group control number assigned to the current group. The value returned in this field is right-justified with leading zeros.
GSIDQUAL	The group sender ID qualifier for envelope type **E**.
GSID	The group sender ID (group header for data type **AS**).
GRIDQUAL	The group receiver ID qualifier for envelope type **E**.
GRID	The group receiver ID (group header for data type **AR**).
GDATE	The group date (group header for data type **DT**).
GTIME	The group time (group header for data type **TM**).
GAPW	The group password (group header for data type **PW**).
GVER	The group version (group header for data type **VR**).

Table 165. Group header/trailer fields in the TRCB (continued)

Field name	Description
GREL	The group release (group header for data type **LV**).
GRESPAGENCY	The group controlling agency used in EDIFACT envelopes. The group responsible agency used in ICS, UCS, and X12 envelopes.

The values in the TRCB fields described Table 166 change as transactions are processed to indicate how much of the group has been processed.

Table 166. Transaction header/trailer fields in the TRCB

Field name	Description
NEWGRP	Indicates whether the transaction is the first transaction of a group. If you want a copy of the group header, you can use a function code value of **2** to obtain an exact image. For more information, see "Retrieve group header API" on page 533. **Y**　　　Starts a new group **(other)**　Does not start a new group
TRNGRP	The total number of transactions processed so far in the current group, stored as a 4-byte binary value (see `GTT`).
GTT	The character representation of `TRNGRP`.

The TRCB fields described in Table 167 are associated with the transaction header and trailer for the current transaction.

Table 167. Fields in TRCB associated with the transaction header and trailer

Field name	Description
NEWTRN	Indicates whether a transaction header is available. If you want a copy of the transaction header, you can use a function code value of **3** to obtain an exact image. For more information, see "Retrieve transaction header API" on page 534. **Y**　　　A transaction header is available **(other)**　A transaction header is not available
LASTINENV	Indicates whether this transaction is the last transaction in the current interchange. **Y**　　　Ends the interchange **(other)**　Does not end the interchange
THCTL	See `THXCTL`.
THXCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros.
SEGTRN	The total number of segments in the current transaction stored as a 4-byte binary value. See `TST` for a character representation of this value.
TTC	The current transaction or message ID value. See also the `TRNID` field on 482.
TVER	The transaction version (transaction header for data type **VR**).
TREL	The transaction release (transaction header for data type **LV**).

Table 167. Fields in TRCB associated with the transaction header and trailer (continued)

Field name	Description
TST	The character representation of SEGTRN.

Last call of session (DE)

The translator signals the last call for a session through the CCB return codes. A ZCCBRC value of **4** and a ZCCBERC value of **1** indicate that all the data from the current file has been processed and that the translator has terminated. If your application is to process another file, cycle back to the first call of the session and set the REQID or FILEID field to indicate the next file that must be processed.

Envelope processing and profile location considerations (E)

The following sections list special considerations for envelope processing.

Locating sending trading partner profile members

During send map processing, the following search sequence is used in an attempt to locate a member in the trading partner profile (TPPROF) for the sender.

Table 168. Search sequence to locate trading partner profile for the sender

For application data:	For C& D records:	Then:
If the Application TP Nickname field names a field and the field contains a value	If the APPLTID field of the C record contains a value	That value is used as the name of the trading partner profile member for the sender.
When neither field contains a value, then:		
If the Interchange Sender ID and ID Qualifier fields name a field and the field contains a value	If the ISID and ISIDQUAL contain values	These values are used to search the trading partner profile members to find an entry with a matching Interchange ID and ID Qualifier. The value in the associated TP Nickname field is used as the trading partner profile member for the sender.
At this point, if no matching entry is found, the sending trading partner is considered to be unknown and the default trading partner profile (with the nickname of **ANY**) is used as the trading partner profile member for the sender.		

Locating receiving trading partner profile members

The following search sequence is used in an attempt to locate a member in the trading partner profile for the receiver.

Table 169. Search sequence to locate trading partner profile for the receiver

For application data:	For C& D records:	Then:
If the Trading Partner ID field names a field and the field contains a value	If the INTPTID field of the C record contains a value	These values and the value in the Data Format Name field are used to search the receive usages/rules table for an active usage/rule. If an active usage/rule is found, the value in the associated TP Nickname field is used as the name of the trading partner profile member for the receiver.
When neither field contains a value, then:		
If the External TP Nickname field names a field and the field contains a value	If the EXTPID field contains a value	That value is used as the trading partner profile member for the sender.
When neither field contains a value, then:		
If the Interchange Receiver ID and ID Qualifier fields name a field and the field contains a value	If the ISID and ISIDQUAL contain values	These values are used to search the trading partner profile members to find an entry with a matching Interchange ID and ID Qualifier. The value in the associated TP Nickname field is used as the trading partner profile member for the receiver.
At this point, if no matching entry is found, the receiving trading partner is considered to be unknown and the default trading partner profile (with the nickname of **ANY**) is used as the trading partner profile member for the receiver.		

Deenvelope processing and profile location considerations (DE)

The following sections list special considerations for deenvelope processing.

Locating sending and receiving trading partner profile members

During receive map processing, the following search sequence is used in an attempt to locate a member in the trading partner profile (TPPROF) for the sender.

1. The sender ID qualifier and sender ID are used to search the trading partner profile members (in TPPROF) to find an entry with a matching `Interchange Qualifier` and `Interchange ID`. If a matching entry is found, the value in the TP Nickname field is used as the trading partner profile for the sender.

2. If no match is found, the trading partner sender ID is parsed as follows:

   a. If the trading partner sender ID is 15 or fewer bytes in length, the interchange ID is parsed into a 7-byte account number and an 8-byte user ID.

   b. If the sender ID is 16 bytes in length, the interchange ID is parsed into a 8-byte account number and an 8-byte user ID.

   c. If the sender ID is 35 bytes in length, the interchange ID is parsed into a 32-byte account number and a 3-byte user ID.

    d. If the sender ID contains separators (such as blanks, periods, or slashes), the value is parsed into an account number and user ID based on the separators found.

    Based on the parsed value, the trading partner profile members are searched for a matching `Account Number` and `User ID`.

3. If no match is found, the interchange ID is used to search the trading partner profile members for a matching `Contact Phone`.

At this point, if no matching entry is found, the sending trading partner is considered to be unknown and the default trading partner profile (with the nickname of **ANY**) is used as the trading partner profile member for the sender.

Once the trading partner profile for the sender has been determined, the same search sequence is performed using the Interchange Receive ID and ID Qualifier values in the data to attempt to locate the trading partner profile nickname for the receiver.

Usages retrieved

WebSphere Data Interchange makes a single call to the database to retrieve all the active usages/rules for a given trading partner and transaction ID. For production transactions, only production usages/rules are retrieved. For test transactions, both test and production usages/rules are retrieved. You can use the `FORCETEST` keyword in the batch utility to override the usage indicator and force the translator to look at only the test usages/rules. In this mode, if a test rule is not found, the transaction is rejected. Once all usages/rules have been retrieved, the data from the usages/rule is compared with the data from the transaction to find the best usage/rule. This is determined by adding up the weights that are assigned to the various fields by WebSphere Data Interchange and picking the usage/rule that has the greatest weight value. A usage/rule is eliminated from consideration if a value has been supplied in the usage/rule that does not match a value from the transaction. For example, if an `Application sender ID` was specified in the usage/rule and the usage/rule value does not match the transaction value, that usage/rule is not considered.

The fields and their weight values are:

Application sender
    4096
Application receive
    2048
Responsible agency code
    1024
Version
    512
Release
    256
Test mode matches
    128
Test mode does not match
    64
Sending trading partner is specific
    32

Receiving trading partner is specific
16
Sending trading partner is KNOWN
8
Receiving trading partner is KNOWN
4
Sending trading partner is ANY
2
Receiving trading partner is ANY
1

Note: When you are using generic receive usages/rules, WebSphere Data Interchange makes a second call to retrieve all of the generic usages/rules (using an ampersand in the trading partner nickname plus standard transaction ID) and the above weighting values are used to select the appropriate generic receive usage/rule.

Issue commit API

The only time this function call is necessary is when the previous request has a `NOCOMIT` field value of **Y**, indicating that the translator must not issue a COMMIT at a point where it usually would do so. Your program then has an opportunity to change resources (such as updating a database) and then request a COMMIT so that the changes made by WebSphere Data Interchange and the changes made by your application program can be synchronized. The COMMIT request is not honored if:

- An interchange is active and interchange level recovery scope (`SCOPE` value of **E**) is in effect.
- Transaction level recovery scope is in effect and a BUNDLE is in progress.
- The SYNCPOINT interval has not been reached. For more information on SYNCPOINT interval, see "SYNCPOINT services" on page 568.

The basic format of the COMMIT request is:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for this API request are defined in Table 170.

Table 170. Parameters for the Issue commit API

Parameter	Description
SNB	**ZSNBNAME** TRANPROC **ZSNBPC** 6
FCB	**ZFCBFUNC** 991
TRCB	**NOCOMIT** N
TRIDB	The translator input data block used in previous requests
TRODB	The translator output data block used in previous requests

Retrieve interchange header API

This API function allows your application to retrieve the image of the current interchange header during enveloping or deenveloping. Make this request when the `NEWENV` field in the TRCB is set to **Y**, indicating that a new interchange has just been started. However, the request can be made at any time until a new interchange is started.

The format of the Retrieve interchange header request is:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for this API request are defined in Table 171.

Table 171. Parameters for the Retrieve interchange header API

Parameter	Description
SNB	The same SNB used in previous requests. No further initialization is required.
FCB	**ZFCBFUNC** 1
TRCB	The same TRCB used in previous requests. No further initialization is required.
TRIDB	The same TRIDB used in previous requests. No further initialization is required.
TRODB	The same TRODB used in previous requests. The translator returns the current interchange header image in the `DATA` field. The size of the image is returned in the `DATALEN` field.

Retrieve group header API

This API function allows your application to retrieve the image of the current group header during enveloping or deenveloping. Make this request when the `wassuP` field in the TRCB is set to **Y**, indicating that a new group has just been started. However, the request can be made at any time until a new group is started.

Note: The BAT segment is only supported on receive. WebSphere Data Interchange does not create a BAT segment.

The format of this API request is:

`FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for this API request are defined in Table 172.

Table 172. Parametres for the Retrieve group header API

Parameter	Description
SNB	The same SNB used in previous requests. No further initialization is required.

Table 172. Parametres for the Retrieve group header API (continued)

Parameter	Description
FCB	**ZFCBFUNC** 　　　2
TRCB	The same TRCB used in previous requests. No further initialization is required.
TRIDB	The same TRIDB used in previous requests. No further initialization is required.
TRODB	The same TRODB used in previous requests. The translator returns the current interchange header image in the DATA field. The size of the image is returned in the DATALEN field.

Retrieve transaction header API

This API function allows your application to retrieve the image of the current transaction header during enveloping or deenveloping. Make this request when the NEWTRN field in the TRCB is set to **Y**, indicating that a new transaction has just been started. However, the request can be made at any time until the next transaction is started.

The format of the API request is:

FXXZ*ccc*(SNB,CCB,FCB,TRCB,TRIDB,TRODB)

The unique parameters for this API request are defined in Table 173.

Table 173. Parameters for the Retrieve transaction header API

Parameter	Description
SNB	The same SNB used in previous requests. No further initialization is required.
FCB	**ZFCBFUNC** 　　　3
TRCB	The same TRCB used in previous requests. No further initialization is required.
TRIDB	The same TRIDB used in previous requests. No further initialization is required.
TRODB	The same TRODB used in previous requests. The translator returns the current interchange header image in the DATA field. The size of the image is returned in the DATALEN field.

Data extraction services

Data extraction services provide functions that enable an application program to retrieve the same information from the TRODB that is extracted and written to the EDIQUERY file when a TRANSACTION DATA EXTRACT or a ENVELOPE DATA EXTRACT is performed.

The format of the API request for data extraction is:

`FXXZ`*ccc*`(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for this API request are defined in Table 174.

Table 174. Parameters for the Data extraction API

Parameter	Description

SNB	**ZSNBNAME**
	            TRANPROC
	ZSNBPC
	            6

FCB	The function control block with one of the following values in `ZFCBFUNC`: **216** Detailed transaction data **217** Transaction image **218** Transaction acknowledgment image **219** Functional acknowledgment image
TRCB	The translator control block. See "Translator Control Block (TRCB)" on page 642 for details on this control block.
TRIDB	The translator input data block. See "Translator Input Data Block (TRIDB)" on page 672 for a description of this block.
TRODB	The translator output data block. This block contains the returned detailed transaction data and has a minimum size of 32000 bytes. For more information on this block, see "Translator Output Data Block (TRODB)" on page 674.

Initialization for data extraction

There are numerous fields in the control blocks defined for data extraction functions whose values need to be established only once. These values can be established before the first call and do not have to be refreshed or changed before any other call. The control blocks, the fields within the control blocks, and the initialization considerations are shown in the following tables.

SNB initialization required for data extraction is:
ZSNBLL
        32
ZSNBNAME
        TRANPROC
ZSNBPC
        6

FCB initialization required for data extraction is:

ZFCBLL
> 4

ZFCBFUNC
> **216**, **217**, **218**, or **219**

TRCB initialization required for data extraction is:

BLKLEN
> 1536

BLKNME
> EDITRCB

BLKTYPE
> The format for the TRIDB and TRODB buffers:
> **H** Unlimited size
> **(other)** Limited to 32768 bytes

TRIDB initialization required for data extraction is:

BLKLEN
> The size of the data block, including this `BLKLEN` field.

> **Note:** Although the TRIDB is not used during a data extraction operation, it still must be specified in the parameter list. A BLKLEN value of **16** is sufficient.

RESERVED
> Binary zeros

TRODB initialization required for for data extraction is:

BLKLEN
> The size of the data block, including this `BLKLEN` field. The minimum value for this field is 32000 bytes.

RESERVED
> Binary zeros

Retrieve detailed data API

This API function allows the application to retrieve detailed information concerning a transaction from the Transaction Store and returns it as formatted records in the TRODB. For more information, see "Transaction Store data extract information categories" on page 302.

The TRCB fields described in Table 175 on page 537 are required to initialize transactions.

Table 175. TRCB fields required to initialize transactions

Field name	Description
TSKEY	The transaction handle for the target transaction. The handle is YYYYMMDDHHMMSSxxnnnn, formatted as a 10-byte packed field. If your program does not pack these values, initialize this field to all blanks or all binary zeros and use the TSKEYU field instead.
TSKEYU	The transaction handle for the target transaction. The handle is YYYYMMDDHHMMSSxxnnnn, formatted as a 20-byte character field. This field is used only if TSKEY does not have a value.
CONCATENATE	Indicates whether detailed information records are concatenated in the TRODB. Valid values are: **Y** Concatenate detailed information records in the TRODB. **N** Return detailed information records individually.
EJECT	Blank.

Once the initialization is complete, the data extraction service is invoked with the following API request:

`FXXZ`*ccc*`(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

Complete information about the transaction identified by the `TSKEY` or `TSKEYU` field is retrieved from the Transaction Store and formatted into the data extract record formats. For more information, see "Transaction Store data extract record formats" on page 303. The data extraction records are returned in the TRODB data block in the following sequence:

1. Interchange record
2. Group record, which can be followed by:
   - Functional acknowledgment interchange record
   - Functional acknowledgment group record
   - Functional acknowledgment transaction record
3. Transaction record, which can be followed by:
   - Transaction acknowledgment interchange record
   - Transaction acknowledgment group record
   - Transaction acknowledgment transaction record
4. Application record

When an outbound transaction is enveloped more than once, the preceding sequence is repeated for each interchange to which the transaction belongs.

When an inbound transaction is translated more than once, more than one application record is returned for each translation.

If there is a functional acknowledgment for the group, the interchange, group, and transaction records for the functional acknowledgment are returned with the group record even if concatenation has not been requested.

If there is a transaction acknowledgment for the transaction, the interchange, group, and transaction records for the transaction acknowledgment are returned with the transaction record even if concatenation has not been requested.

Numerous errors can occur at this point. If an error occurs, the return code and extended return code for that error are posted in the ZCCBRC and ZCCBERC fields of the CCB. With each call, your program must check the TRXACCEPT and TRABORT fields for error information.

When the translation was acceptable (TRXACCEPT value of **Y**), the contents of TRODB data block depend on the value of CONCATENATE as follows:

Y   As much information as possible is concatenated and returned in the TRODB data block.

(other) The first record concerning the transaction is returned in the TRODB data block.

The TRCB fields described in Table 176 are returned in response to the first request for detailed information about a transaction.

Table 176. Returned information on extract request

Field name	Description
TRXACCEPT	Indicates whether data was extracted for the specified transaction.
TRABORT	Indicates whether if an error was so severe that the translator stopped processing
TSKEY	The transaction handle formatted as a 10-byte packed value.
TSKEYU	The transaction handle formatted as a 20-byte character value.
EJECT	Indicates whether all data has been received for this transaction. Valid values are: **X** CONCATENATE was requested and all the data did not fit in the TRODB buffer. Issue the request again, leaving the EJECT value as **X** to get the residual data. Continue to do so until you receive an EJECT value of **Y**. **N** CONCATENATE was not requested and more data remains for this transaction. Issue the request again, leaving the EJECT value as **N** to get the next detailed record. Continue to do so until you receive an EJECT value of **Y**. **Y** The data returned on this request was the last data for the transaction.

The following fields are returned in the TRODB:

TRODB
> The translator output data block.

DATALEN
> The amount of data returned in the DATA field.

DATA   The formatted data extraction records. For more information, see "Transaction Store data extract information categories" on page 302.

Retrieve transaction image API

This API function allows your application to retrieve the transaction image for the current transaction returned from the Retrieve detailed data API. You must request the current transaction (function code **216)** before requesting an image (function code **217)**. Once the transaction detail record is returned, make the following API request:

`FXXZ`*ccc*`(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

A transaction image cannot be generated until the transaction detail record has been generated. The transaction image detail record is returned in the TRODB. If the image is too large to fit in the TRODB, the `EJECT` field of the TRCB has a value of **X**. In this case, make repeated requests (leaving the `EJECT` value as **X**) until the `EJECT` value changes to **Y**, indicating that no data remains.

Retrieve transaction acknowledgment image API

This API function allows your application to retrieve the transaction acknowledgment image for the current transaction returned from the Retrieve detailed data API. You must request the current transaction (function code **216)** before requesting an image (function code **217)**. Once the transaction detail record is returned, make the following API request:

`FXXZ`*ccc*`(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

A transaction acknowledgment image cannot be generated until the transaction detail record has been generated. The transaction image detail record is returned in the TRODB. If the image is too large to fit in the TRODB, the `EJECT` field of the TRCB has a value of **X**. In this case, make repeated requests (leaving the `EJECT` value as **X**) until the `EJECT` value changes to **Y**, indicating that no data remains.

Retrieve functional acknowledgment image API

This API function allows your application to retrieve the functional acknowledgment image for the current group returned from the Retrieve detailed data API. You must request the current group (function code **216)** before requesting an acknowledgment image (function code **219)**. Once the group detail record is returned, make the following API request:

`FXXZ`*ccc*`(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

A functional acknowledgment cannot be generated until the transaction detail record has been generated. The transaction image detail record is returned in the TRODB. If the image is too large to fit in the TRODB, the `EJECT` field of the TRCB has a value of **X**. In this case, make repeated requests (leaving the `EJECT` value as **X**) until the `EJECT` value changes to **Y**, indicating that no data remains.

Communication services

The functions provided in the Communication API enable an application program to send or receive transaction data, files, and messages to and from trading partners. This section provides some general comments about data flow and the components of the Communication services, followed by detailed data about each function.

The transaction or file data that is to transmitted is not passed directly through an API parameter. Rather, it is provided indirectly as the name of a sequential flat file that contains the data to be sent or must be used to collect the data that is being received.

Note: In CICS, the data to be sent or received is restricted to a TS queue.

Ordinarily, the functions called by an API use an external file to contain the data. Therefore, if to send transaction data to a trading partner, first place the transaction data in an sequential file using one of the following ways:

- Writing an API program to request enveloping services or translation and enveloping services. For more information, see "Envelope API" on page 503 and "Translate-to-standard API" on page 446.
- Using the ENVELOPE, REENVELOPE, or combination commands provided by the WebSphere Data Interchange Utility.
- Using the enveloping and reenveloping functions provided by the Transaction Store facility. If transactions are being received from a trading partner, the communication functions ask the appropriate network programs to collect the data in the specified file. Management reporting is called at this time to record the number of interchanges and the total number of bytes received. However, responses are generated from the transactions in the file in several ways, including:
- Writing an API program to request that the transactions in the file be deenveloped, or deenveloped and translated. For more information, see "Deenvelope transaction" on page 523 and "Translate-file-to-application API" on page 486.
- Using the DEENVELOPE or combination commands provided by the WebSphere Data Interchange Utility.
- Using the deenvelope functions provided by the Transaction Store facility.

In some cases an application program indirectly requests a communications API function from WebSphere Data Interchange by going through a network program which processes the request with WebSphere Data Interchange. Not all the programs involved in processing the request are written by or provided with WebSphere Data Interchange. The indirect path from an application program to a network program is:

1. The application program makes the Communication API request by calling the appropriate STUB program (FXXZC, FXXZCBL, FXXZPLI, or FXXZASM).
2. WebSphere Data Interchange invokes the communication component of WebSphere Data Interchange (EDICM) which reads various profiles, including the network profile (NETPROF). The network profile entry contains the names of the following programs:
   - The communication routine. This routine is the bridge between WebSphere Data Interchange and your network, because it knows the interfaces on both ends and uses data provided by WebSphere Data Interchange to build the commands

required by your network program. WebSphere Data Interchange provides communication routines for the networks directly supported by WebSphere Data Interchange.

- The network program. Network programs are not provided by WebSphere Data Interchange, but by your network provider and have defined interfaces. The purpose of the communication routine is to create this interface.

- The message handler. This program processes responses from the network program so that results can be communicated to the original program making the API request. WebSphere Data Interchange provides message handlers for the networks directly supported by WebSphere Data Interchange. However, anyone can write a message handler for an internal network or for one not directly supported by WebSphere Data Interchange.

3. After validation is performed, EDICM invokes the communication routine to process the API request.

4. The communication routine transforms the application's API requests into requests that are acceptable to your network program. Once the commands are built, the network program is invoked.

5. The network program processes the commands and sends or receives data as directed by the commands. Once the transmission is complete, the network program returns to the communication routine.

6. The communication routine invokes the message handler to process the responses from the network. Control is returned to EDICM, which returns to the application.

Note: If you are writing an API application, a network program, or a message handler, be aware that all communication requests made internally by WebSphere Data Interchange utilities and facilities use the communications API functions described in this section.

Communications service functions

Table 177 contains an overview of the functions provided by the communications service. The logical name for the communications service is COMM.

Table 177. Overview of functiondsprovided by Communication services

Function	Code	Sample Call Statement for Function
Queue standard data	**110**	`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB,DATABLK)`
Send transactions	**211**	`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`
Send files	**221**	`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`
Receive	**232**	`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`
Cancel previously sent data	**233**	`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`
Return file name	**300**	`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`
Process network acknowledgments	**252**	`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`

Trading partner profile data block (TPPDB)

The trading partner profile data block (TPPDB) is a required parameters for all functions requested from the Communication API. This does not mean that an application making an API request must populate each field in the block with data. In fact, the only field in the TPPDB that must be initialized is the `BLKLEN` field.

This block is intended as an output block, but certain fields in the block will accept data. These fields are used by the communication routine to build commands for the network program and contain the parameters that control or direct the network program. If your program does not provide values for these fields, corresponding values retrieved by the communications routine from the mailbox (requestor) profile entry or the trading partner profile entry are used. The Communication API determines if you are providing values in this block by looking at the `TPNICKNM` field. If this field contains blanks, the TPPDB is interpreted as the output block. If this field is not blank, the TPPDB is interpreted as the input block and the values specified in the TPPDB override the values from the mailbox (requestor) profile member. The TPPDB fields you can specify are described in Table 178.

Table 178. Trading partner fields used by communications

Field name	Description
BLKLEN	**1532** (the length of the trading partner profile data block).
BLKNME	EDITPPDB.
TPNICKNM	The trading partner nickname that you want to send data to or receive data from. This can also be the nickname used for getting parameters for the network program. For example, when transaction data is sent, more than one trading partner can be involved. There could be data in the file for multiple trading partners. If this field is left blank, communications uses the value from the `TPNICKNM` field of the CMCB, and the TPPDB becomes an output-only block. If this field contains a value, the TPPDB is interpreted as an input block and data from the following fields is used to construct the commands for the network program.
NETID	The ID of a network profile (NETPROF) entry that defines the network to which the request must be directed. This field is used as an input field only if the `REQID` and the `NETID` fields of the CMCB contain blanks.
ACCTNUM	The account number of the trading partner to which the current request applies.
USERID	The user ID of the trading partner to which the current request applies.
NETCLS	A code indicating any special status of the data being sent. The network specifies the acceptable codes.
NETCHG	Indicates how charges are shared between sender and receiver. The network specifies the acceptable codes.
NETACK	Indicates which network acknowledgments are requested. The network specifies the acceptable codes.
NETVCHK	Indicates whether the destination is to be verified before sending occurs. The network specifies the acceptable codes.
NETRETN	The number of days data is kept in a mailbox before it is purged. The network specifies the acceptable values.

Table 178. Trading partner fields used by communications (continued)

Field name	Description
NETEDIO	Indicates whether you want EDI segments to be stored in the receiving file as separate records. The network specifies the acceptable codes.
NETEDIP	Indicates whether EDI data you receive is to have special EDI processing. The network specifies the acceptable codes.
STGFRMTO	Indicates whether you want to use the storage format defined by the network program. The network specifies the acceptable codes.
MACHTYPE	If your trading partner is using the Personal Computer/Information Exchange (PC/IE) product to receive your data, enter **1**. Otherwise, leave blank.
STGFRMT	Indicates to the network how data is stored for non-EDI files. The network specifies the acceptable codes.
EOTID	The character that signifies the end of message text to the network. This applies only to sending or receiving free-form messages.
NETCMDS	The name of a member of the PDS that is allocated to EDINTCMD which contains network commands that WebSphere Data Interchange must pass to the network.
TPDATALINE	The phone number to dial, to connect directly to your trading partner's computer to pass data.
TIMEOUT	The maximum allowable amount of time that the data line for communications can be idle without being dropped. The network specifies the acceptable values.

Common CMCB output fields

Some CMCB fields are established by the communications component of WebSphere Data Interchange (EDICM). They contain input data for the communication routine, and output data for the application program making the communications API requests. The values in these fields define the network's capabilities and are set based on the values found in the FSUPPORT network command entry (Network Commands (NETOP) profile) for the network. These fields are described in the following table and are not repeated in the individual API descriptions. A value of **Y** in these fields indicates the function described by the field is supported by the designated network.

Table 179. Common CMCB output fields

Field name	If this field contains a value of Y:
FFILE	Non-EDI files are supported.
FEDIX	X12 interchanges are supported.
FEDIE	EDIFACT interchanges are supported.
FEDIU	BG/EG interchanges are supported.
FEDIG	GS/GE interchanges are supported.
FEDII	ICS interchanges are supported.
FEDIT	STX/END interchanges. are supported.
FCANCEL	The CANCEL function is supported.
FCLASS	User message classes are supported.

Table 179. Common CMCB output fields (continued)

Field name	If this field contains a value of Y:
FACK	Network acknowledgments are supported.
FSYSMSG	System messages are supported.
FRCVBTP	Receiving by trading partner is supported.
FRESTART	Restart is supported.
FNOUSERID	Logging in with account number only is supported.
FACCTSEP	The character used to separate the account number and user ID.

Return codes from communications

For all the Communication API requests, the result of the request is indicated by the return code (ZCCBRC) and extended return code (ZCCBERC) in the CCB. In general, the ZCCBRC field has the following values:

0   The request was processed without error.

4   A warning that the request was processed, but something unusual was noticed. Two extended return codes are:

  1   The network being used does not have a network program defined in the network profile, indicating that no SEND/RECEIVE/CANCEL requests can be processed. **1040**

  1040   No data was returned on a receive request.

8   An error occurred. Another request might not have the same problem.

12   A severe error occurred. This usually indicates a system error. Additional requests probably will not succeed.

Send transactions and restart send transactions API

The communication request to send transactions is used to build all the commands necessary for a network program to send a file of interchanges to the appropriate trading partners. For point-to-point networks, WebSphere Data Interchange assumes that all the interchanges in the file are for the same trading partner. For generalized networks, the file can contain interchanges for multiple trading partners and the network uses information in the interchange header to route each interchange to the correct trading partner. Before the network program is invoked to send the data, the file is scanned by WebSphere Data Interchange and the status of each interchange in the file is set to SEND STARTED.

The communication request to Restart send transactions applies only if your network supports restart and you specified checkpoint-level recovery when initially sending the data. For z/OS, if an error causes network processing to restart during a send operation, you can use the Restart send transactions API to restart and complete the send. The Restart send transactions request is not supported in CICS. For more information on checkpoint recovery, refer to the Expedite Base/MVS Programming Guide.

If you want to use the Restart send transactions API, you must initialize the control blocks with the same values that were set for the initial send, except for NETOP which is not required on restart.

After the network program returns, it updates the status of each interchange to SEND REQUESTED (return code **0**), SENT WITH ERRORS (return codes **1–4**), or SEND REQUEST ERROR (return code not **0–4**), based on the degree of success of the network program. Status is updated through the Update status API (see "Update status services" on page 561). The WebSphere Data Interchange Utility uses the send transactions API whenever any of the following PERFORM commands are requested:

- SEND
- ENVELOPE AND SEND
- REENVELOPE AND SEND
- TRANSLATE AND SEND

The WebSphere Data Interchange Utility uses the Restart send transactions API only for the RESTART SEND command.

The basic format of the API request to send or resend transactions is:

`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`

The unique parameters for the send transactions or restart send transactions API requests are defined in Table 180.

Table 180. Parameters for the send transactions of restart send transactions API

Parameter	Description
SNB	**ZSNBNAME** COMM **ZSNBPC** 5
FCB	**ZFCBFUNC** **211** Send transactions **260** Restart send transactions
CMCB	The communications control block. For more information, see "Communication Control Block (CMCB)" on page 677.
TPPDB	The trading partner profile data block. For more information, see "Trading partner profile data block (TPPDB)" on page 542.

CMCB initialization for sending transaction data

Before issuing an send API request, you must initialize the CMCB fields described in Table 181.

Table 181. CMCB fields required for sending transaction data

Field name	Initialization
BLKLEN	**254** (the length of the CMCB).
BLKNME	Any 8-character name. The recommended value is **EDICMCB**.

Table 181. CMCB fields required for sending transaction data (continued)

Field name	Initialization
TPNICKNM	The trading partner nickname that is used when building the commands for the network program. This field is not necessary for a send transactions request, because the network distributes the data based on the destination in the interchange header. This field is ignored if the TPNICKNM field of the TPPDB is supplied.
NETID	The ID of the network profile (NETPROF) entry that defines the invoked network. The value placed in this field is ignored if the REQID field has a value. When a REQID is provided, the NETID is taken from the mailbox (requestor) profile entry and this field is updated with that NETID value.
NETOP	Do not edit this file unless instructed by Customer Care personnel to do so. Changing the information in this file arbitrarily will disrupt the operation of your communications software. For more information, see "Send network operation" on page 547.
REQID	The mailbox (requestor) profile ID used for this request. The mailbox (requestor) profile identifies the mailbox you want to use to identify yourself. Numerous values are taken from this profile entry while building the commands for the network.
CLRFILE	Indicates whether you want WebSphere Data Interchange to clear the file containing the transaction data at the end of the processing. Valid values are: **Y** Clears the file if the data was sent successfully **U** Always clears the file **N or (other)** Does not clear the file
ACCTYP	Account type. Applies only if the SENDFILE network command is being used. This field must contain a value of **D** to indicate that a trading partner account number and user ID are being supplied.
DATATYP	The type of file name supplied in the FILENAME field. Valid values are: **A** Data set name **D** ddname If FILENAME is not provided on input, both FILENAME and DATATYP are provided on output. **Notes:** 1. This field is ignored in CICS. 2. Special IEBASE considerations: Unless the FILENAME field contains an entire data set name and this field contains a value of **A**, you must place a value of **D** in this field. This is necessary so that WebSphere Data Interchange can build the proper IEBASE INMSG FILEID parameter. If FILENAME is left blank, a **D** is still required in this field.

Table 181. CMCB fields required for sending transaction data (continued)

Field name	Initialization
FILENAME	The name of a file containing the data to be sent. This is either a ddname or data set name based on the value of DATATYP. If this field is blank, WebSphere Data Interchange assigns a value using the following rules: • The Trans data queue field from the network profile member is used as a starting point. If this field is blank, a default value of **QDATA** is used. • If the network command is **SENDEDI**, an **E** is appended to the ddname. For example, the QDATA default name becomes **QDATAE**. If the ddname is already 8 characters long, the last character is overlaid with an **E**. If FILENAME is not provided on input, both FILENAME and DATATYP are provided on output. **Notes:** 1. In CICS, this field must be a TS queue. 2. Special IEBASE considerations: Unless this field contains an entire data set name and the DATATYP field contains a value of **A**, you must place a value of **D** in the DATATYP field. This is necessary so that WebSphere Data Interchange can build the proper IEBASE INMSG FILEID parameter. If this field is left blank, a **D** is still required in the DATATYP field.
DCIND	The delivery class for the data being sent. This value is not interpreted by WebSphere Data Interchange but is handled and understood by the network program. Valid values are: **blank** Normal delivery **P** High priority
ACKIND	The type of acknowledgment wanted. See the definition of this field in "Communication Control Block (CMCB)" on page 677 for a list of values.
ENAME	The message user class to associate with each interchange in the file being sent. See "Default message user class" on page 549 for the default values assigned if a specific value is not supplied.
MSGNAME	The message name to associate with each interchange in the file being sent. See "Default message name" on page 549 for the default values assigned if a specific value is not supplied.

Send network operation

The function you use to invoke communications indicates in general what the application is requesting, but the network commands profile (NETOP) specifies which commands must be built for the network program to process. WebSphere Data Interchange and the WebSphere Data Interchange utilities and facilities set the network operator based on the type of data contained in the file.

Two network commands are available: SENDEDI and SENDSTREAM. Use SENDEDI when you want the network program to parse the interchange headers in the file. Use SENDSTREAM when you want the entire file sent to the trading partner without any interrogation required by the network program.

Send transactions returned information

The return code (ZCCBRC) and extended return code (ZCCBERC) fields of the CCB indicate whether the send transactions request was successful.

Numerous errors are possible.

The CMCB fields described in Table 182 are returned on a request to send transactions.

Table 182. CMCB fields returned on a request to send transactions

Field name	Description
SEQNUM	The network sequential number assigned to this transmission. The number is maintained in the `Network sequence` field of the network profile entry and is incremented on each request to send data. This number might be important later if you want to recall (cancel) this transmission.
ENAME	If a value was not supplied as input, the value from the `Message user class` field in the mailbox (requestor) profile entry is returned.
ACKIND	If a value was not supplied as input, the value from the `Net acknowledgment` field of the mailbox (requestor) profile or the `Net acknowledgment` field of the trading partner profile is returned.
DATATYP	If a value was not supplied as input, **D** is returned for generalized networks and **A** is returned for point-to-point networks, indicating the type of value in the `FILENAME` field. **Notes:** 1. This field is ignored in CICS. 2. When using point-to-point networks, the `DATATYP` and `FILENAME` fields normally indicate that a data set name was used. However, if a trading partner was not provided with the request, the ddname taken from the `Trans data queue` field of the network profile is returned.
FILENAME	If a value was not supplied as input, the ddname is returned for generalized networks and the data set name is returned for point-to-point networks. **Notes:** 1. For CICS, this field must be a TS queue. 2. When using point-to-point networks, the `DATATYP` and `FILENAME` fields normally indicate that a data set name was used. However, if a trading partner was not provided with the request, the ddname taken from the `Trans data queue` field of the network profile is returned.
NPSSCDE	The start session response code returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPESCDE	The end session response code returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPERRCD	The error code of the most severe error returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks. The value in this field is included in the VN1015 error message.

Table 182. CMCB fields returned on a request to send transactions (continued)

Field name	Description
NPSEVER	The severity of the most severe error code returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks. The value in this field is included in the VN1015 error message.

Default message user class

The message user class is a code that trading partners agree to use for identifying classes of information to be sent or received. It allows users to select a single type of information from a mailbox that might hold various kinds of data. You can assign the message user class to an interchange when you send it by supplying a value in the ENAME field of the CMCB.

If you do not supply a value, the message user class from the mailbox (requestor) profile member is used. If a message user class is not present in the mailbox (requestor) profile, a default value is assigned based on the type of interchange, as follows:

ISA, ICS, or GS
> #E2

BG #EC

UNB Taken from the UNB14 (APRF) field. If the UNB14 field is not present or contains blanks, a value of **#EE** is used.

STX Taken from the STX11 (APRF) field. If the STX11 field is not present or contains blanks, a value of **#EU** is used.

Default message name

The message name is an arbitrary value that can be assigned to an interchange when the interchange is sent. It is part of an alternate key that can later be used to update the status of an interchange. It can also be used if you need to recall an interchange. The MSGNAME field of the CMCB contains the message name that must be assigned to each interchange in the file being sent. If a message name is not supplied, a default value is assigned based on the type of interchange, as follows:

ISA or ICS
> The last 8 bytes of the interchange control number

GS The group control number, left-justified and padded with blanks

BG, UNB, or STX
> The interchange control number, left-justified and padded with blanks

Send files API

The communications request to send files is used to build all the commands necessary for a network program to send a file containing non-EDI data to a particular trading partner. There is no specific PERFORM command provided with the WebSphere Data Interchange Utility for sending a file of non-EDI data.

The basic format of the API request to send a file is:

FXXZ*ccc*(SNB,CCB,FCB,CMCB,TPPDB)

The TPPDB is described in Table 178 on page 542. The CMCB settings for this request are described in Table 183. The unique SNB and FCB parameters for the send files API request are:

SNB

> **ZSNBNAME**
>> COMM
>
> **ZSNBPC**
>> 5

FCB

> **ZFCBFUNC**
>> 221

CMCB initialization

Some CMCB fields must be initialized before the API request is made. Table 183 describes the fields and the initialization requirements.

Table 183. CMCB initialization requirements for the send files API

Field name	Initialization
BLKLEN	**254** (the length of the CMCB).
BLKNME	**EDICMCB**.
TPNICKNM	The trading partner nickname to which the file must be sent. This field is ignored if the `TPNICKNM` field of the TPPDB contains a value.
NETID	The network that is invoked. This field is ignored if the `REQID` field has a value. When a `REQID` is provided, the `NETID` is taken from the mailbox (requestor) profile entry and this field is updated with that `NETID` value.
NETOP	**SENDFILE**.
REQID	The mailbox (requestor) profile ID value to use for this request. The mailbox (requestor) profile identifies the mailbox that you want to use to identify yourself. Numerous values are taken from this profile entry while building the commands for the network.
CLRFILE	Indicates whether you want WebSphere Data Interchange to clear the file at the end of the processing. Valid values are: **Y** Clears the file if the data was sent successfully **U** Always clears the file **N or (other)** Does not clear the file
ACCTYP	**D**. Indicates that a trading partner account number and user ID are being supplied.
DATATYP	The type of file name supplied in the `FILENAME` field. Valid values are: **A** Data set name **D** ddname **Note:** In CICS, this field is ignored.
FILENAME	The name of a file containing the data to be sent. This is either a ddname or data set name based on the value of `DATATYP`. **Note:** For CICS, this value must be a TS queue.

Table 183. CMCB initialization requirements for the send files API (continued)

Field name	Initialization
DCIND	The delivery class for the data being sent. The value entered in this field is not interpreted by WebSphere Data Interchange, but is handled and understood by the network program. Valid values are: **blank** **P**　　　　High priority **I**　　　　Express delivery
ACKIND	The type of acknowledgment wanted. See "Communication Control Block (CMCB)" on page 677 for a list of values.
ENAME	The message user class to associate with the file. If a value is not supplied, the value in the `Message user class` field from the mailbox (requestor) profile entry is used.
MSGNAME	The message name to associate with the file.

Send files returned information

The return code (`ZCCBRC`) and extended return code (`ZCCBERC`) fields of the CCB indicate if the request to send a file was successful.

Numerous errors are possible.

The CMCB fields that are returned on a send file API request are described in Table 184.

Table 184. CMCB fields returned on a send file API request

Field name	Description
SEQNUM	The network sequential number assigned to this transmission. The number is maintained in the `Network sequence` field of the network profile and is incremented on each request to send data. This number might be important later if you want to RECALL (CANCEL) this transmission.
ENAME	If a value was not supplied as input, the value in the `Message user class` field of the mailbox (requestor) profile is returned.
ACKIND	If a value is not supplied as input, the value from the `Net acknowledgment` field of the mailbox (requestor) profile or the trading partner profile is returned.
NPSSCDE	The start session response code returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPESCDE	The end session response code returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPERRCD	The error code of the most severe error returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks. The value in this field is included in the VN1015 error message.

Table 184. CMCB fields returned on a send file API request (continued)

Field name	Description
NPSEVER	The severity of the most severe error returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks. The value in this field is included in the VN1015 error message.

Receive and restart receive API

The WebSphere Data Interchange Utility uses the receive API when any of the following PERFORM commands are requested:

* RECEIVE
* RECEIVE AND DEENVELOPE
* RECEIVE AND TRANSLATE

The Restart send transactions request applies only if your network supports restart and you specified checkpoint-level recovery when initially sending the data. For z/OS, if an error causes network processing to restart during a send operation, you can use the Restart send transactions API to restart and complete the send. The Restart send transactions request is not supported in CICS. For more information on checkpoint-level recovery, refer to the Expedite Base/MVS Programming Guide.

The Restart receive transactions request applies only when your network supports restart and you specified checkpoint level recovery when initially receiving the data. For z/OS, if an error causes network processing to restart during a receive operation, you can use the Restart receive transactions API to restart and complete the receive. The Restart receive transactions request is not supported in CICS. For more information on checkpoint-level recovery, refer to the Expedite/MVS Programming Guide.

Note: When a restart receive request is issued, you must initialize the control blocks with the same values as the initial receive, except for `NETOP` which is not required for restart.

The WebSphere Data Interchange Utility uses the Restart receive API when the RESTART RECEIVE command is requested.

The format of the receive API request is:

`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`

The TPPDB is described in Table 178 on page 542. The CMCB settings for this request are described in "CMCB initialization for receive requests" on page 553. The unique SNB and FCB parameters for the receive and Restart receive API request are:

SNB

    ZSNBNAME

        COMM

    ZSNBPC

        5

FCB

    ZFCBFUNC

232 Receive
261 Restart receive

CMCB initialization for receive requests

Some CMCB fields must be initialized before the receive API request is made. Table 185 describes the fields and initialization requirements.

Table 185. Initialization for send transaction data

CMCB	Initialization
BLKLEN	**254** (the length of the CMCB).
BLKNME	**EDICMCB**.
TPNICKNM	The trading partner nickname used when building the commands for the network program. This field is not needed for a receive. If you specify a trading partner nickname in this field, only data from this trading partner is requested. If you do not specify a trading partner nickname, data from all trading partners is requested. This value in this field is also ignored if the TPNICKNM field of the TPPDB contains a value.
NETID	The network that is invoked. The value in this field is ignored if REQID contains a value. When REQID is specified, the value for this field is taken from the NETID field in the mailbox (requestor) profile entry.
NETOP	See "Receive network operation" on page 554.
REQID	The mailbox (requestor) profile ID value that is used for this request. The mailbox (requestor) profile identifies the mailbox from which you want to receive data.
ACCTYP	If a TPNICKNM is provided, this field contains **D**. Otherwise, it must contain a blank.
DATATYP	For z/OS, the type of file name supplied in the FILENAME field (if a FILENAME value is provided). For CICS, this field is ignored. Valid values are: **A** Data set name **D** ddname If FILENAME is not provided on input, both FILENAME and DATATYP are provided on output.
FILENAME	The name of a file for receiving the data. For z/OS, this is either a ddname or data set name based on the value of DATATYP. For CICS, this must be TS queue. If this field is blank, WebSphere Data Interchange uses the value from the Receive file name field in the mailbox (requestor) profile member and forces the DATATYP field to **D**. This field is required when the RECVFILE network command is used.
ENAME	The message user class used to identify the data received. If a value is not specified, the Message user class field from the mailbox (requestor) profile entry is used. The values from this field and the TPNICKNM field combine to form the selection criteria indicating what data is desired from the mailbox.
RECVTYP	Indicates whether only the first file that meets the selection criteria must be returned. **G** Returns only the first file that meets the selection criteria **blank** Returns all files meeting the criteria

Table 185. Initialization for send transaction data (continued)

CMCB	Initialization
RESRECL	Controls whether the network program splits records when the data received has records larger than the logical record length (`LRECL`) of the file to which the data is being written (`FILENAME` field). **S** Records are split into smaller records using the maximum size set in `LRECL` **blank** Stops the network program with an A03 system abend

Receive network operation

The function you use to invoke communications indicates in general what the application is requesting, but the network commands profile (NETOP) indicates which commands must be built for the network program to process. WebSphere Data Interchange and the WebSphere Data Interchange utilities and facilities set the network operator based on the type of data contained in the file.

Two network commands are available: RECVEDI and RECVSTREAM. Use RECVEDI when you want the network program to parse the interchange headers in the file. Use RECVSTREAM when you want the entire file received from the trading partner without any interrogation required by the network program.

Receive returned information

The return code (ZCCBRC) and extended return code (ZCCBERC) fields of the CCB indicate if the receive request was successful.

Numerous errors are possible.

The CMCB fields described that are returned on receive data API request are described in Table 186.

Table 186. CMCB fields returned on the receive data API

Field name	Description
DATATYP	If a value was not supplied as input, **D** is returned for generalized networks and **A** is returned for point-to-point networks indicating the type of file in the `FILENAME` field. **Note:** When using point-to-point networks, the `DATATYP` and `FILENAME` fields normally indicate that a data set name was used. If a trading partner was not specified on the request, the ddname taken from the `Trans data queue` field in the network profile entry is returned.

Table 186. CMCB fields returned on the receive data API (continued)

Field name	Description
FILENAME	If a value was not supplied as input, the ddname is returned for generalized networks and the data set name is returned for point-to-point networks. The ddname returned is the `Receive file name` field from the mailbox (requestor) profile entry if the RECVEDI network command is being used. For other network commands, the `FILENAME` is a required input parameter. **Note:** When using point-to-point networks, the `DATATYP` and `FILENAME` fields normally indicate that a data set name was used. If a trading partner was not specified on the request, the ddname taken from the `Trans data queue` field in the network profile entry is returned.
TPNICKNM	If a value is not supplied on input indicating a request to receive data from any trading partner, on output this field contains the trading partner nickname for the first data received (if the network response data provides enough information for this to be determined). This might not apply to all networks.
FILERCVD	Indicates whether data was received. The message handler sets this flag if processing the network responses indicates that data was received. This might not apply to all networks.
ENAME	The message user class associated with the first file received is returned if the network program provides this information. This field is returned by the message handler while processing the network responses. This might not apply to all networks.
MSGNAME	The message name associated with the first file received is returned if the network program provides this information. This field is returned by the message handler while processing the network responses generated by the network program. This might not apply to all networks.
NPSSCDE	The start session response code returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPESCDE	The end session response code returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.
NPERRCD	The error code of the most severe error returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks. The value in this field is included in the VN1015 error message.
NPSEVER	The severity of the most severe error returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks. The value in this field is included in the VN1015 error message.

Cancel API

The Cancel request is used to build all the commands necessary for a network program to cancel (or recall) data previously sent. If a file is sent to a trading partner by mistake, it can be canceled until the trading partner receives the data. Not all networks support the Cancel request. The following description is directed at the AT&T Global Network. No specific PERFORM command provided with the WebSphere Data Interchange Utility supports the Cancel API.

The basic format of the Cancel API request is:

`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`

The TPPDB is described in Table 178 on page 542. The CMCB settings for this request are described in "CMCB initialization for Cancel requests." The unique SNB and FCB parameters for the receive and Restart receive API request are:

SNB

> **ZSNBNAME**
>> COMM
>
> **ZSNBPC**
>> 5

FCB

> **ZFCBFUNC**
>> 233

CMCB initialization for Cancel requests

Some CMCB fields must be initialized before the Cancel API request is made. Table 187 describes the fields and initialization requirements.

Table 187. Initialization for cancel

Field name	Initialization
BLKLEN	**254** (the length of the CMCB).
BLKNME	**EDICMCB**.
TPNICKNM	The trading partner nickname from which files are being recalled. This field is ignored if the `TPNICKNM` field of the TPPDB has a value.
NETID	The network that is invoked. If the `REQID` field is specified, this field is ignored and the value in the mailbox (requestor) profile entry is used.
NETOP	**CANCEL**.
REQID	The mailbox (requestor) profile ID value used for this request. The mailbox (requestor) profile identifies the mailbox that you want to use to identify yourself. Numerous values are taken from this profile entry while building the commands for the network.
ACCTYP	**D**. A trading partner account number and user ID is being supplied.
ACKIND	The type of acknowledgment wanted regarding the cancellation. See "Communication Control Block (CMCB)" on page 677 for the field definition.
DCIND	The delivery class used when the file was originally sent.
ENAME	The message user class associated with the file when it was originally sent. If a value is not supplied, the `Message user class` field from the mailbox (requestor) profile entry is used.
MSGNAME	The message name associated with the file when it was originally sent.
SEQNUM	The sequential number assigned to the file when it was originally sent.
CANSD	The cancellation start date in `YYMMDD` format. Initialize with blanks if not used.
CANST	The cancellation start time in `HHMMSS` format. Initialize with blanks if not used.
CANED	The cancellation end date in `YYMMDD` format. Initialize with blanks if not used.
CANET	The cancellation end time in `HHMMSS` format. Initialize with blanks if not used.

Table 187. Initialization for cancel (continued)

Field name	Initialization
TMZONE	The time zone used for CANSD, CANST, CANED, and CANET. **L** indicates local time. **G** indicates Greenwich mean time.

Cancel returned information

The return code (ZCCBRC) and extended return code (ZCCBERC) fields of the CCB indicate whether the request to cancel was successful.

Numerous errors are possible.

The CMCB fields described in Table 188 are returned from a Cancel request. The values are returned by the message handler associated with the network. The values are extracted from the network program response records. This might not apply to all networks.

Table 188. CMCB fields returned from a Cancel request

CMCB	Description
NPSSCDE	The start session response code.
NPESCDE	The end session response code.
NPERRCD	The most severe error code. The value in this field is included in the VN1015 error message.
NPSEVER	The severity of the most severe error. The value in this field is included in the VN1015 error message.

Return filename API

The Return filename request is issued to determine the name of the file associated with the network for writing transaction data. This API request is issued internally during translation or enveloping operations. The network program (such as IEBASE or DSXMIT2) is not involved in processing this API request; the request is processed directly through the communication routine.

The basic format of the Return filename API request is:

FXXZ*ccc*(SNB,CCB,FCB,CMCB,TPPDB)

The TPPDB is described in Table 178 on page 542. The CMCB settings for this request are described in "CMCB initialization for Return filename request" on page 558. The unique SNB and FCB parameters for the Return filename API request are:

SNB

> **ZSNBNAME**
>> COMM
>
> **ZSNBPC**
>> 5

FCB

> **ZFCBFUNC**
>> 300

CMCB initialization for Return filename request

Some CMCB fields must be initialized before the API request is made. Table 189 describes the fields and initialization requirements.

Table 189. Initialization for query filename

Field name	Initialization
BLKLEN	**254** (the length of the CMCB).
BLKNME	**EDICMCB**.
TPNICKNM	The trading partner nickname for which the current interchange is being created. This is important for point-to-point networks because the data set name used for transaction data is constructed using this field.
NETID	The network for which information is wanted. If REQID contains a value, the value in this field is ignored and this value is taken from the mailbox (requestor) profile.
REQID	The mailbox (requestor) profile ID value used for this request.

Return filename returned information

The return code (ZCCBRC) and extended return code (ZCCBERC) fields of the CCB indicate whether the request was successful. Numerous errors are possible.

The CMCB fields that are returned on a Return filename request are described in Table 190.

Table 190. CMCB fields returned by the Return filename API

Field name	Description
DATATYP	The type of file identified in the FILENAME field. **A** Point-to-point network **D** Generalized network
FILENAME	For generalized networks, the ddname associated with the network from the Trans data queue field in the network profile entry. For point-to-point networks, the data set name associated with the trading partner.

Internal calls

The following API calls are used internally by the WebSphere Data Interchange Utility.

Queue standard data API

The Queue standard data request is an internal API function issued by the translator or enveloper when an interchange is complete and ready to be written to a file associated with the network. The network program is not called to process this request. The network program is only called when a request to send transaction data is received. For issues to consider before sending the data, see "Sending transaction data" on page 514

514. For information about how to provide the name of the file where the transaction data must be written, see "Translate-file-to-application API" on page 486 and "Envelope API" on page 503.

Process network acknowledgments API

The Process network acknowledgments request is used to build all the commands needed to instruct the network program to return network acknowledgment data. The WebSphere Data Interchange Utility uses this API when the UPDATE STATUS command is issued.

The basic format of the Process network acknowledgments API request is:

`FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)`

The TPPDB is described in Table 178 on page 542. The CMCB settings for this request are described in "CMCB initialization for Process network acknowledgments requests." The unique SNB and FCB parameters for the Process network acknowledgments API request are:

SNB

> **ZSNBNAME**
>> COMM
>
> **ZSNBPC**
>> 5

FCB

> **ZFCBFUNC**
>> 252

CMCB initialization for Process network acknowledgments requests

Some CMCB fields must be initialized before the Process network acknowledgments API request is made, as described in Table 191.

Table 191. CMCB field initialization requirements for Process network acknowledgements API

Field name	Initialization
BLKLEN	**254** (the length of the CMCB).
BLKNME	**EDICMCB**.
NETID	The network that is invoked. This field is ignored if `REQID` has a value.
	Note: This function must be invoked for each network individually. The UPDATE STATUS command issues a request for every network defined in the network profile unless you use the `NETID` keyword to request a specific network.
NETOP	If a file of network acknowledgments has already been received and you want to process it, use a `NETOP` value of **NO-NETOP**. Otherwise, leave the field blank and the communication routine will invoke the network program to return the acknowledgment data.

Table 191. CMCB field initialization requirements for Process network acknowledgements API (continued)

Field name	Initialization
REQID	The mailbox (requestor) profile ID used for this request. If you specify this field, network acknowledgments are requested and processed for the specified requestor. If you do not specify this field, acknowledgments are requested and processed for every requestor defined for the network identified by `NETID`.

There are no output CMCB fields for this request. The return code (`ZCCBRC`) and extended return code (`ZCCBERC`) indicate whether the request was successful.

Update status services

Update status services allow the status of an interchange to be updated as it progresses from enveloping, to sending, to receipt by the trading partner. The update status service is used by the communications routine and message handler when an interchange is sent, and also during the processing of network acknowledgments. Certain characteristics of an interchange become known only when the interchange is sent. Therefore, the update status service also allows other fields in the interchange to be set when status is updated. These are established using the update status data block (USDB), which is described in Appendix A.

The USDB is *NOT* included in Appendix A. The only information about its contents that I can find is in "Update status data block" on page 566 which is later in this section.

Update status service overview

The logical name for the update status service is TRANSSRV.

The basic format of the update status function request call is:

```
FXXZC(SNB,CCB,FCB,USKB###,status[,USDB[,REQID]])
```

where ### is one of the following function codes:

210 The envelope full key with trading partner identified by the account number and user ID.

211 The transaction handle value for one of the transactions in the interchange.

212 An alternate key with the trading partner identified by the account number and user ID.

213 The full envelope key with the trading partner identified by the interchange qualifier and interchange ID.

214 An alternate key with the trading partner identified by the interchange qualifier and interchange ID.

215 The full envelope key with the trading partner identified by the trading partner nickname.

The six functions supported by the update status service all update the status (and possibly other fields provided in the USDB) of an interchange. The different function codes are used to indicate how to identify the interchange whose status has been updated. Four of the functions make up the key for an interchange, as follows:

* Trading partner nickname - 16 characters, left-justified with trailing blanks
* Direction of the interchange (send or receive) - 1 character
* Interchange receiver ID (from the interchange header) - 35 characters, left-justified with trailing blanks
* Interchange control number (from the interchange header) - 14 characters, right-justified with leading zeros

Update status services

The two remaining functions provide two alternate keys for an interchange (established at the time an interchange is sent) that also can be used when processing network acknowledgments. They are:

- Unique 8-character value associated with the interchange by the network program
- Another alternate key that might not be a unique value and consists of the following fields:
  - Trading partner nickname - 16 characters, left-justified with trailing blanks
  - Message name - 8 characters, left-justified with trailing blanks
  - Message user class - 8 characters, left-justified with trailing blanks
  - Message sequence number - 5 characters, right-justified with leading zeros

Which key values are supplied to the update status service depends on whether you use the primary key or one of the alternate keys and on how the trading partner is identified. The format of the different key values is provided in the sections that follow.

Update status API

The basic format of the Update status API request is:

`FXXZccc(SNB,CCB,FCB,USKB,status[,USDB[, 'reqid']])`

The unique parameters for the Update status API request are defined in Table 192..

Table 192. Parameters for the Update status API

Parameter	Description

SNB	**ZSNBNAME**
	      TRANSSRV
	ZSNBPC
	      5 if `USDB` is not specified **6** if `USDB` is specified **7** if `REQID` is specified

FCB	The function control block with a `ZFCBFUNC` value that identifies the type of key provided. Valid values are: **210** Full envelope key with the trading partner identified by account number and user ID **211** Transaction handle value of one of the transactions in the interchange **212** Alternate key provided with the trading partner identified by account number and user ID **213** Full envelope key with the trading partner identified by interchange qualifier and interchange ID **214** Alternate key provided with the trading partner identified by interchange qualifier and interchange ID **215** Full envelope key with the trading partner identified by the trading partner nickname
USKB	The key block with a format based on the value of `ZFCBFUNC`. For more information, see "Full envelope key" on page 563.

Table 192. Parameters for the Update status API (continued)

Parameter	Description
STATUS	The new status for the interchange. Valid values are: **41** Send error **42** Send request error **43** Not sent, network error **46** Send started **48** Send requested **49** Sent to network **50** Accepted by the network **51** Delivered by the network **52** Purged by the network **53** Recall requested from network **54** Recall request error **55** Recalled from network
USDB	The optional update status data block (see "Update status data block" on page 566).
REQID	If you specify this parameter, the management reporting component of WebSphere Data Interchange is called to update the statistics on the number of bytes sent from the specified requestor ID. Applies only if STATUS is **48** or going from **48** to some other status.

Update status return codes

The results of a update status request are posted in the return code and extended return code fields of the CCB. Valid values are defined in Table 193.

Table 193. Update status API return codes

0	Status update was successful.
8	Status update failed. Extended return codes are: **210** No interchange was found. **211** Internal program error. **212** An error occurred when attempting to update the database.

Full envelope key

The three ways to identify a trading partner when using the full envelope key are:

- Trading partner nickname, which identifies the trading partner profile member
- Interchange qualifier and interchange ID fields, which are used to search the trading partner profile members for a matching entry
- Account number and user ID fields, which are used to search the trading partner profile members for a matching entry

The formats for these values are shown in the following tables.

Trading partner nickname

Table 194 on page 564 describes the format for a full key using a trading partner nickname. Use this update status key block when ZFCBFUNC contains **215**.

Table 194. Full key using trading partner nicknames

Name	Offset	Length	Format	Description
TPNICKNM	0	16	Char	The trading partner nickname for an entry in the trading partner profile. Left-justified with trailing blanks.
FILLER	16	48	Char	Blanks.
DIR	64	1	Char	The direction of the interchange. must have value of **S**.
INTCTLNO	65	14	Char	The interchange control number. Right-justified with leading zeros.
RECEIVER	79	35	Char	The interchange receiver ID value sent in the interchange. Left-justified with trailing blanks.

Interchange qualifier and ID

Table 195 describes the format for a full key using an interchange qualifier and ID. Use this update status key block when `ZFCBFUNC` contains **213**.

Table 195. Full key format using an interchange qualifier and ID

Name	Offset	Length	Format	Description
QUALIFIER	0	4	Char	The interchange qualifier sent in the interchange. Left-justified with trailing blanks.
RECEIVER	4	35	Char	The interchange receiver ID value sent in the interchange. Left-justified with trailing blanks.
INTCTLNO	39	14	Char	The interchange control number. Right-justified with leading zeros.

Account number and user ID

Table 196 describes the format for a full key using an account number and user ID. Use this update status key block when `ZFCBFUNC` contains **210**.

Table 196. Full key format using an account number and user ID

Name	Offset	Length	Format	Description
ACCTNUM	0	32	Char	The trading partner account number. Left-justified with trailing blanks.
USERID	32	32	Char	The trading partner user ID. Left-justified with trailing blanks.
DIR	64	1	Char	**S**. The direction of the interchange.
INTCTLNO	65	14	Char	The interchange control number. Right-justified with leading zeros.
RECEIVER	79	35	Char	The interchange receiver ID value as sent in the interchange. Left-justified with trailing blanks.

Transaction handle

The status of an entire interchange can be updated by using the transaction handle of any transaction in the interchange. Use this update status key block when `ZFCBFUNC` contains **211**.

Table 197. Transaction handle

Name	Offset	Length	Format	Description
THANDLE	0	10	Char	The transaction handle value of one of the transactions in the interchange. The transaction handle has a format of `YYYYMMDDHHMMSSxxnnnn` stored as a 10-byte packed value. All of the transactions in the interchange have the same status, and the handle value for any transaction in the interchange can be used on this request.

Alternate keys

You can identify a trading partner using one of the two alternate keys below:

- Account number and user ID - the value in the `UNIQUEID` field
- Interchange qualifier and interchange ID - a combination of the `MSGCLASS`, `MSGNAME` and `SEQNUM` fields

The values for these fields are not known until an interchange has been sent. To establish the alternate key values, use the update status data block (see "Update status data block" on page 566) along with one of the other keys. Once this is done, subsequent updates to status can be made with any of the key values. The formats for the alternate keys are shown in the following tables.

Alternate key 1 using account number and user ID

The alternate key format described in Table 198 uses the account number and user ID. Use this update status key block when `ZFCBFUNC` contains **212**.

Table 198. Alternate key format using account number and user ID

Name	Offset	Size	Format	Description
ACCTNUM	0	32	Char	The trading partner nickname account number. Left-justified with trailing blanks.
USERID	32	32	Char	The trading partner user ID. Left-justified with trailing blanks.
MSGCLASS	64	8	Char	The message user class assigned to the interchange. Left-justified with trailing blanks.
MSGNAME	72	8	Char	The message name assigned to the interchange. Left-justified with trailing blanks.

Table 198. Alternate key format using account number and user ID (continued)

Name	Offset	Size	Format	Description
SEQNUM	80	5	Char	The sequence number assigned to the interchange. Right-justified with leading zeros.
UNIQUEID	85	8	Char	The unique ID value assigned to the interchange by the network. This value is used instead of the MSGCLASS, MSGNAME, or SEQNUM fields unless it contains blanks. If this field is specified, ACCTNUM and USERID are not required.

Alternate key 2 using interchange qualifier and ID

The alternate key format described in Table 199 uses the interchange qualifier and ID. Use this update status key block when ZFCBFUNC contains **214**.

Table 199. Alternate key format using interchange qualifier and user ID

Name	Offset	Length	Format	Description
QUALIFIER	0	4	Char	The interchange qualifier as sent in the interchange. Left justified with trailing blanks.
RECEIVER	4	35	Char	The interchange receiver ID as sent in the interchange. Left-justified with trailing blanks.
FILLER	39	25	Char	Blanks.
MSGCLASS	64	8	Char	The message user class assigned to the interchange. Left-justified with trailing blanks.
MSGNAME	72	8	Char	The message name assigned to the interchange. Left-justified with trailing blanks.
SEQNUM	80	5	Char	The sequence number assigned to the interchange. Right-justified with leading zeros.
UNIQUEID	85	8	Char	The unique ID value assigned to the interchange by the network. This value is used instead of the MSGCLASS, MSGNAME, or SEQNUM fields unless it contains blanks. If this field is specified, QUALIFIER and RECEIVER are not required.

Update status data block

Using the update status data block is always optional. This data block contains information about an interchange that is generally determined at the time an interchange is sent.

Note: If you specify REQID in the call, a update status data block must be available. If you do not have any data for the block, initialize all the fields to blanks to indicate they do not apply.

Table 200 describes the fields used to build an alternate key that can be used later to retrieve and update the status of this interchange.

Table 200. Interchange fields established during status update

Name	Offset	Size	Format	Description
MSGCLASS	0	8	Char	The message user class assigned to the interchange.
MSGNAME	8	8	Char	The message name assigned to the interchange.
SEQNUM	16	5	Char	The message sequence number assigned to the interchange.
UNIQUEID	21	8	Char	The unique message ID assigned to the interchange by the network. If this field is not blank, it must contain a value that is unique for all interchanges currently in the Transaction Store.
SENDDATE	29	8	CHAR	The date the interchange was sent (YYYYMMDD).
SENDTIME	37	6	CHAR	The time the interchange was sent (HHMMSS).
NETACK	43	1	CHAR	The type of network acknowledgment expected for this interchange. See "Trading Partner Profile Block (TPPDB)" on page 688 for valid values.

SYNCPOINT services

The descriptions of translation and enveloping services frequently refer to the recovery scope for the session (see "Send Recovery Scope" on page 469). SYNCPOINT services provide the interface for controlling the recovery scope, and provide an environment-independent interface for requesting either that changes to resources are permanent (COMMIT work) or that changes must be removed (ROLLBACK work).

If a system or application fails, or if a specific ROLLBACK work request is made, all changes made to resources since the last COMMIT point are backed out of the system. It will be as if the changes were never made.

In an z/OS/DB2 environment, a request to commit work is made directly to DB2 using a COMMIT request. A request to roll back work is made directly to DB2 with a ROLLBACK request.

In a CICS/DB2 environment, a request to commit work is made to CICS with an EXEC CICS SYNCPOINT request. CICS passes this request on to DB2 and controls the committing of CICS resources with DB2 resources. A request to roll back work is made to CICS with an EXEC CICS SYNCPOINT ROLLBACK request. Again, CICS passes this request to DB2 and controls the removal of CICS resources with DB2 resources.

SYNCPOINT services issues the proper request based on the execution environment. When you use this API, recovery scope is established using the SCOPE field in the TRCB. When you use the WebSphere Data Interchange Utility, recovery scope is established by using the RECOVERY keyword on the PERFORM command.

You can use SYNCPOINT services to lengthen the recovery scope by setting the SYNCPOINT interval to a positive integer value greater than **1**. You can effectively turn off SYNCPOINT service by setting the SYNCPOINT interval to **-1**. In this case, the application program controls the SYNCPOINT interval and must ensure that not to shorten the recovery scope.

For example, if an interchange recovery scope is in effect and the application issues a COMMIT (or EXEC CICS SYNCPOINT) request in the middle of the interchange, the chances of deadlock increase, and the consistency between the application databases and the WebSphere Data Interchange Transaction Store is compromised.

Table 201 lists the functions provided by the SYNCPOINT service. The logical name for the SYNCPOINT service is SYNCSERV.

Table 201. Syncpoint services functions

Function	Code	Sample Call Statement for Function
Initialize SYNCPOINT services	1	FXXZccc(SNB,CCB,FCB,SYNCVAL)
COMMIT work	2	FXXZccc(SNB,CCB,FCB)
ROLLBACK work	3	FXXZccc(SNB,CCB,FCB)

Note: During API translation, setting the NOCOMMIT field in the TRCB to **Y** prevents the translator from issuing commits. However, setting NOCOMMIT to **Y** does not prevent the WebSphere Data Interchange termination process from issuing a commit. To prevent this termination commit, issue a SYNCPOINT service call with the SYNCPOINT interval set to **-1** prior to the WebSphere Data Interchange termination call. For more information, see "Send Recovery Scope" on page 469.

DB2 TIMEOUT/DEADLOCK processing

DB2 TIMEOUT / DEADLOCK processing occurs in a DB2 environment when multiple translations are being run concurrently and the trading partners are not in the same order for all the translations. The DB/2 terminates one of these transactions and issues one of the following return codes:

–911   The DB/2 parameter is ROLBE=YES.
–913   The DB/2 parameter is ROLBE=NO.

The translator attempts to recognize that a rollback occurred, issues a return code of **–911**, and allows the translation to continue, even though another translation might have acquired the next sequential control number. The DB/2 termination is flagged with an RS0000 message. The translator will attempt a retry after a return code of **–911** or **–913** when appropriate. If DB2 issues a rollback, the translation is terminated with a TR0810 message.

During retry, WebSphere Data Interchange has the options to issue either:
* A ROLLBACK command
* A FETCH command for the trading partner control numbers again (TA)

Table 202 describes the DB2 recovery conditions and actions.

Table 202. DB2 recovery conditions and actions

	Conditions True (T) or False (F)				Actions Yes (Y) or No (N)	
Row	DI Control	Deadlock Not Timeout	Updates Made	DB2 Rollback	DI Rollback	Fetch Control #s
1	F	F	F	F	N	Y
2	F	F	F	T	N	N
3	F	F	T	F	N	Y
4	F	F	T	T	N	N
5	F	T	F	F	N	N
6	F	T	F	T	N	N
7	F	T	T	F	N	N
8	F	T	T	T	N	N
9	T	F	F	F	N	Y
10	T	F	F	T	N	Y
11	T	F	T	F	N	Y
12	T	F	T	T	N	N
13	T	T	F	F	Y	Y

Table 202. DB2 recovery conditions and actions (continued)

	Conditions True (T) or False (F)				Actions Yes (Y) or No (N)	
Row	DI Control	Deadlock Not Timeout	Updates Made	DB2 Rollback	DI Rollback	Fetch Control #s
14	T	T	F	T	N	Y
15	T	T	T	F	Y	N
16	T	T	T	T	N	N

Initialize SYNC function

The syntax of the Initialize SYNC function request is:

`FXXZccc(SNB,CCB,FCB,interval)`

The unique parameters for this function request are defined in Table 203.

Table 203. Parameters for Initialize SYNC request

Parameter	Description
SNB	**ZSNBNAME** SYNCSERV **ZSNBPC** 4
FCB	**ZFCBFUNC** 1 **INTERVAL** Indicates the length of time that elapses between synchronization points. A 4-byte binary value. Valid values are: **0** General SYNCPOINT processing based on the recovery scope in effect. **n** A positive number indicating how often a SYNCPOINT must be taken. For example, if *n* had a value of **5** and transaction recovery scope is specified, a COMMIT is issued after every 5 transactions. Actually, a COMMIT is requested after each transaction, but only every 5th request is honored by the SYNCPOINT service. **−1**

Initialize SYNC function return codes

The results of the Initialize SYNC request are posted in the return code and extended return code fields of the CCB. Valid values are:

0   Initialization was successful.

12   Initialization failed. The extended return code is **12** (insufficient virtual storage).

COMMIT work function

The syntax of the COMMIT work function request is:

`FXXZ`*ccc*`(SNB,CCB,FCB)`

The unique parameters for the COMMIT work function request are defined in Table 204.

Table 204. Parameters for the COMMIT work function request

Parameter	Description
SNB	**ZSNBNAME** SYNCSERV **ZSNBPC** 3
FCB	**ZFCBFUNC** 2

The results of the COMMIT work request are posted in the return code and extended return code fields of the CCB. Valid values are:

0   COMMIT work was successful.

4   COMMIT ignored. The extended return code is:

> **1**   The COMMIT request was ignored for one of the following two reasons:
> - An interval of **-1** was established on the SYNCPOINT initialization function.
> - The interval has not been reached yet.

12   COMMIT work failed. The extended return code is:

> **N**   The `SQLCODE` from DB2 indicating why the COMMIT was not honored.

ROLLBACK work

The syntax of the ROLLBACK work function request is:

`FXXZ`*ccc*`(SNB,CCB,FCB)`

The unique parameters for the ROLLBACK work function request are defined in Table 205.

Table 205. Parameters for the ROLLBACK work function request

Parameter	Description
SNB	**ZSNBNAME** SYNCSERV **ZSNBPC** 3

Table 205. Parameters for the ROLLBACK work function request (continued)

Parameter	Description
FCB	**ZFCBFUNC** 3

The results of the ROLLBACK work request are posted in the return code and extended return code fields of the CCB. Valid values are:

0   ROLLBACK work was successful.

12   ROLLBACK work failed. The extended return code is:

> **N**   The SQLCODE from DB2 indicating the reason for the ROLLBACK failure.

Note: A ROLLBACK request is always honored and issued even when the SYNCPOINT interval has a value of **-1** or the value has not been reached.

Get envelope service

During outbound processing, you can use the Get envelope service to get data directly from the translator, rather than from the file the translator writes to. Normally, after WebSphere Data Interchange creates an interchange, the interchange is routed automatically to a file. (In CICS, this file is always a TS queue.) By writing an exit to circumvent this action, the envelope can be retrieved directly from the translator and then processed. In CICS, for example, the interchange could be routed to a TD queue. WebSphere Data Interchange recognizes this type of user exit when the following keywords are used on enveloping commands: IEXIT(*exitname*), IACCESS(**M**), and ITYPE(**UE**). For more information, see "Get Envelope service example" on page 748 and "Get/Put envelope exit and service" on page 324.

The syntax of the Get envelope API request is:

FXXZASM(GPCB,CCB,FCB,BUFFER,LEN)

The parameters for this function request are defined in Table 206.

Table 206. Parameters for the Get envelope API request

Parameter	Description
GPCB	The Get/Put envelope control block. The fourth parameter passed to IEXIT user exits.
CCB	The common control block used to initialize WebSphere Data Interchange. The second parameter passed to IEXIT user exits.
FCB	The function control block. The third parameter passed to IEXIT user exits.
BUFFER	The working storage into which the envelope will be read. You must specify the length of this area in the LEN parameter.
LEN	The full-word length of the working storage (BUFFER) that will contain the envelope.

Put envelope service

During inbound processing, you can use the Put envelope service to put data directly into the translator, rather than having the translator read a file. Normally, in order to deenvelope an interchange, WebSphere Data Interchange reads a file containing the interchange. In CICS, this file is always a TS queue. By writing an exit to circumvent this action, you can pass an envelope directly into the translator. In CICS, for example, an interchange could be read from a TD queue and passed directly into the translator. WebSphere Data Interchange recognizes this type of user exit when the following keywords are used on deenveloping commands: IEXIT(*exitname*), IACCESS(**M**), and ITYPE(**UE**). For more information, see "Put Envelope service example" on page 748 and "Get/Put envelope exit and service" on page 324.

The syntax of the Put envelope API request is:

`FXXZASM(GPCB,CCB,FCB,BUFFER,LEN)`

The parameters for the Put envelope function request are defined in Table 207.

Table 207. Parameters for the Put envelope API request

Parameter	Description
GPCB	The Get/Put envelope control block. The fourth parameter passed to IEXIT user exits.
CCB	The common control block used to initialize WebSphere Data Interchange. The second parameter passed to IEXIT user exits.
FCB	The function control block. The third parameter passed to IEXIT user exits.
BUFFER	The working storage into which the envelope will be read. You must specify the length of this area in the `LEN` parameter.
LEN	The full-word length of the working storage (`BUFFER`) that will contain the envelope.

Put envelope service

Chapter 7. Using WebSphere Data Interchange in the AIX and Windows environment

The chapter provides instructions for running WebSphere Data Interchange Server.

Running from the command line

WebSphere Data Interchange Server always reads commands from STDIN and writes the results to STDOUT: these are treated as STREAMs. When invoked from the command line, the command line processor automatically opens STDIN and STDOUT, piping them wherever the user requests. You typically prepare a file of PERFORM commands for input and redirect the input from that file. You would probably redirect the STDOUT to a file.

An example of this usage is shown below:

```
ediservr < sample.cmd > results.txt
```

where `sample.cmd` is the input file and has PERFORM commands, and `results.txt` contains the output from ediservr.exe.

The commands files consist of a set of WebSphere Data Interchange commands separated by semicolons. Every command is terminated with a semicolon.

The first command is always a SET, and the second command is always INIT. These are followed by a series of Set file commands that specify the input and output files for the following PERFORM command(s). The SET FILE PERFORM sequence can be repeated as many times as required.

A typical command file is as follows:

```
set plan(ediec32e);
init;
set file(PRTFILE,prtfile);
set file(TRKFILE,trkfile);
set file(EXPFILE,expfile);
set file(XMLFILE,poxml5sr.dat);
set file(OUTFILE,outfile);
PERFORM TRANSFORM WHERE INFILE(XMLFILE) OUTFILE(OUTFILE)
        SYNTAX(X) CLEARFILE(Y) XMLEBCDIC(N)
        TRACELEVEL(A2);
term;
```

The contents of `sample.cmd` are:

SET command

Sets up the environment to point to the WebSphere Data Interchange database

Running from the command line

INIT command

Loads the startup information and connect to the database using parameters defined above.

SET FILE (LogicalFileName, RealFileName)

Defines various INPUT and OUTPUT files needed for translation. The LogicalFileName of each file is assigned with a RealFileName that includes the complete path. Depending on the type of PERFORM command used, some files are mandatory, while others are optional.

PERFORM

Issues standard perform commands. This command uses LogicalFileNames for various files used.

TERM   Disconnects from the database and frees all allocated memory.

If translation is successful, the contents of `results.txt` are:

```
DI Translator Started, build date: Feb 19 2002
DI Translator processed your request.
DI Translator shutdown
```

If translation fails, the contents of `results.txt` are:

```
DI Translator Started, build date: Feb 19 2002
DI Translator Error. RC= "errorcode" , ERC="extended return code"
DI Translator shutdown
```

Triggering from an MQSeries queue

The WebSphere Data Interchange adapter program is installed as part of WebSphere Data Interchange for Multiplatforms Version 3.2. The configuration scripts provided set up the necessary queues and definition objects. The adapter uses MQSeries Triggering to know when messages need processing.

When a message is put to an application queue, a trigger message is created. The MQSeries trigger monitor receives the message and executes the adapter. The adapter then passes the information needed to process the application message to the WebSphere Data Interchange server/translator. Application messages are committed, rollbacked, or moved to a failure queue depending on the return codes from the WebSphere Data Interchange Server.

The adapter will wait the user-configured time interval for any successive messages, and then terminate. The trigger monitor then restarts WebSphere Data Interchange adapter upon receipt of another trigger message.

Base MQSeries support architecture uses six MQSeries queues, three input and three output.

Input queues:
- EDI_IN
- DF_IN
- XML_IN

Output queues:
- EDI_OUT
- DF_OUT
- XML_OUT

The `wdi.mqcommands` file in the samples directory contains all MQ Service Command (MQSC) instructions for creating the needed queues.

The necessary queues and definition objects are created in the default queue manager.

To create these six queues and configure the input queues for triggering, run the wdicommand script. The specific configuration scripts for integrating with other products, such as WebSphere MQ Integrator, are available at:

`http://www.ibm.com/websphere/datainterchange`

Each step of the trigger program is coordinated with a message exits dll or shared library. On Windows this is called `msgExits.dll` On AIX it is called `msgExits.so`. If this dll is found in the binaries path at run time it will change the behavior of the trigger program. If it is not found then the trigger program uses its default settings.

The message exits dll can instruct the trigger program to skip messages, terminate, take or skip a syncpoint, and so on. you can use this to customize the behavior of the adapter to route failed messages to a special queue, or to notify someone if a failure in translation occurs. The interface to the message exits dll is documented in Adapter user exits on page "Adapter user exits" on page 587.

When triggered, the adapter:
1. Reads the wdi.properties file for runtime directories.
2. Calls the trigger startup exit msgTrigger() if present and proceeds based on the return code from the exit.
3. Initializes WebSphere Data Interchange. If WebSphere Data Interchange cannot be initialized, the adapter turns triggering off for the queue and terminates.
4. Sets the name of the file that the message will be received into, which is `datadirectory`(from property files)/`rcvdirectory`(from property files)/`MQSeries message ID`(from MQMD).rcv.
5. For each message on the queue:
   a. Browses the data queue to get the information on the next available message.
   b. Calls the message tracking exit if present, and passes it the browse data. The message exit can return the batch ID to be used and an indicator of whether to proceed or to skip this message.
   c. If OK to proceed, calls WebSphere Data Interchange with a PERFORM RECEIVE AND PROCESS ONEMESG(Y) WHERE REQID(mq_queue_name[1-16]) BATCHSET(batchid).
   d. Upon returning from WebSphere Data Interchange, calls the msgTransform() exit with the return codes. If the return code from the exit instructs the trigger

program not to proceed normally, do what the return code is documented to mean in the adapter user exits on page "Adapter user exits" on page 587, otherwise do the following:

- If translation is acceptable (rc = 0), execute a syncpoint.
- If translation is not acceptable (r <> 0), the adapter posts the message to the dead letter queue defined within MQSeries. Then execute a syncpoint.

   e. Moves on to the next message (restarts the process at step a).

6. When no more messages arrive within the specified interval (see Adapter user exits below), call the msgTerminate user exit (if one exists). If it indicates so, proceed with termination, terminate WebSphere Data Interchange, and then the adapter itself.

Advanced Adapter

The WebSphere Data Interchange advanced adapter is designed to provide an MQSeries interface to WebSphere Data Interchange which is capable of handling multiple requests in parallel. The original WDIAdapter program was designed to handle requests in a serial fashion. This meant that when a trigger message was generated for a queue a single translator instance was assigned to that queue to process all the data from that queue. Using the advanced adapter one or more translators can be assigned to a single queue to provide parallel processing of data on that input queue.

Overview of the Advanced Adapter

WebSphere Data Interchange is designed to interface with MQSeries through the use of an adapter program which can listen for trigger message generated by MQSeries when data arrives on a given queue. The original adapter that came with WebSphere Data Interchange was designed to use a default trigger monitor which was delivered with MQSeries. It allowed for a single translator process to be started and assigned to an input queue. This did not allow for parallel processing of messages on a single queue.

The new advanced adapter is designed to allow the user to create a pool of translators when the WDIServer program is started. When data arrives on an MQSeries queue one or more of these translators can be assigned to the queue to process data. This means that each queue can be independently configured to request any number of translators

(up to the number of translators in the initial pool).
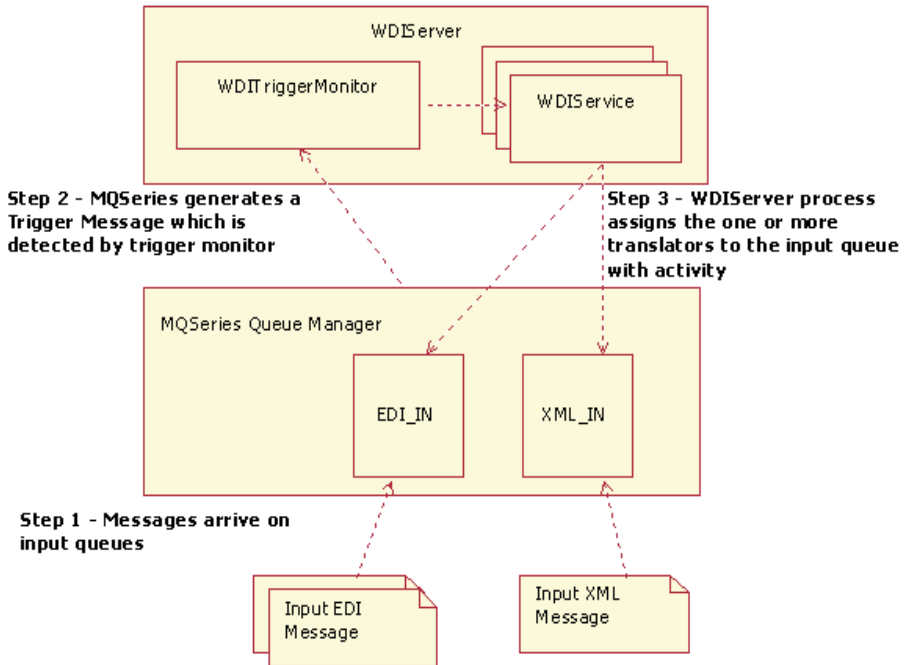
Figure 25. Overview of advanced adapter

The diagram in Figure 25 provides a high level overview of how the advanced adapter works. In that diagram two input queues are defined called XML_IN and EDI_IN. With the advanced adapter each of these queues can be configured to use a different number of translators. For instance, if it is determined that the EDI_IN queue will be receiving more traffic it could be configured to request two translators while the XML_IN queue could be configured to request only one translator.

Installation and Setup

The advanced adapter was first provided in CSD 5 of WebSphere Data Interchange. Several important enhancements were made to the adapter in CSD 7 and CSD10. Therefore it is important that you install the latest CSD which at the time of this writing is CSD 10. The advanced adapter is bundled into a group of four executables, which can all be found in the bin directory under the WDI standard home directory. These four executables are:

1. WDIServer - Main program which starts a trigger monitor and a pool of translators.
2. WDIShutdown - Used to issue a shutdown command to the server
3. WDITrigger - Trigger monitor which is initiated by WDIServer.
4. WDIService - Internal program which is started by WDIServer and controls a single translator instance.

Triggering from an MQSeries queue

Once the advanced adapter has been installed (by applying the latest CSD level) these four programs will be added to your bin directory and the new adapter will be available for use. The next step in the process is to configure the wdi.properties file for your system.

Adapter Configuration

You must first decide where you wish to have the WDIServer program operate. Once this runtime directory has been selected you will probably want to move the wdi.properties file which is in the samples directory over to this runtime directory. If you do not wish to keep the wdi.properties file within the same directory as your runtime environment (or you wish to use some other name other than wdi.properties) the environment variable WDISERVER_PROPERTIES can be used to specify a different file name. To do this simply provide a fully qualified file name as the value to that environment variable. WDIServer will look for a file named wdi.properties in the current working directory. If no file is found by that name it will then check the WDISERVER_PROPERTIES environment variable. If no file is specified it will return an error and exit. This environment variable can be very useful even if plan to keep the wdi.properties file in the same directory where you run from. This is because the WDIShutdown command also needs to find the properties file. With the environment variable set you don't have to be in the runtime directory when you run the WDIShutdown command.

When the WDIServer program is initiated and it has found a properties file a pool of translators will be started. Once these translators are up and ready to begin processing data a trigger monitor will be started which will connect to the initiation queue specified in the properties file. At this time one or more directories with names like WDITransCmdQ_xxxx will be created. The value indicated by the xxxx will be a group of numbers which indicate an internal task id assigned to that instance of the translator. There will be one of these directories for each of the translators started in the pool. The idea behind these directories is to provide each translator with an independent environment.

Since the goal of the advanced adapter is to allow multiple translators to operate independently, it is a good idea to make sure that output files specified in the WDI Service profiles do not use fully qualified paths or relative paths that move the output file out side the current working director (i.e ../MyFile.txt). If you do specify a file outside the current working directory the result will be the same file will be used by all the translators in the pool. This can lead to performance issues as each translator will lock the file while it is in use. Finally, you should ensure that the service profile is not using a directory which will not exist within the WDITransCmdQ_xxxx directory. For instance, specifying a value of ADirectory/MyFile.txt would cause an error because there will be no directory named ADirectory within the WDITransCmdQ_xxxx directory.

The same is true for directories defined in the wdi.properties file. If you look at the wdi.properties file in the samples directory you should see the following information:
dtddirectory=dtds
prtdirectory=prt
appdirectory=adf
edidirectory=edi
eexdirectory=eex

xmldirectory=xml
aexdirectory=aex
rptdirectory=rpt
fakdirectory=fak
qrydirectory=qry
xexdirectory=xex
wrkdirectory=wrk
rcvdirectory=rcv
trkdirectory=trk

These values should all be changed to point to the current working directory. To do this simply change the above lines in the properties file to look like this: dtddirectory=.
prtdirectory=.
appdirectory=.
edidirectory=.
eexdirectory=.
xmldirectory=.
aexdirectory=.
rptdirectory=.
fakdirectory=.
qrydirectory=.
xexdirectory=.
wrkdirectory=.
rcvdirectory=.
trkdirectory=.

Properties File Options

Below is a list of all the keywords along with a description for each. Some options present in the wdi.properties file are used only by the old adapter. Likewise new options have been added to the properties file that the old adapter will simply ignore.

qmgrname
>
> Indicates the queue manager name that the adapter will use. This value is case sensitive and must match a queue manager configured on your system. If it is not provided an attempt will be made to connect to the default queue manager. This option was added to support the new adapter. The old adapter did not need to know the queue manager because it depended upon being triggered using the default trigger monitor provided by MQSeries. It simply took the queue manager name that it was triggered with. The new adapter runs as a service and must connect to a queue manager since the service will be starting its own trigger monitor.

initq   This is the initiation queue that will be used by the trigger monitor. Normally this will be WDI.INIT.Q. If in your environment you have chosen to create your own initiation queue you can provide its name here. This value is case sensitive and must match an existing initiation queue or an error will be reported. This is a new property added to support the new adapter.

runtimedirectory
>
> This indicates the directory where the wdi executables are located. This property is ignored by the advanced adapter.

datadirectory

This indicates the directory where the original WDIAdapter program was going to operate from. This property is ignored by the advanced adapter.

dtddirectory

Indicates the directory where dtds can be found. This is one case where you may wish to provide a fully qualified path. If you are using DTDs in any of your translations you can create a separate directory to store all your DTDs on the server and use this property to find the value.

ptrdirectory

Allows you to specify a directory for the print file. For the advanced adapter this value should point to the current working directory.

appdirectory

Allows you to specify a directory for the application data format files. For the advanced adapter this value should point to the current working directory.

edidirectory

Allows you to specify a directory for the EDI input/output. For the advanced adapter this value should point to the current working directory.

eexdirectory

Allows you to specify a directory for the exception file. For the advanced adapter this value should point to the current working directory.

xmldirectory

Allows you to specify a directory for the XML input/ouput. For the advanced adapter this value should point to the current working directory.

aexdirectory

Allows you to specify a directory for the Application Exception File. For the advanced adapter this value should point to the current working directory.

rptdirectory

Allows you to specify a directory for the Report File. For the advanced adapter this value should point to the current working directory.

fakdirectory

Allows you to specify a directory for the Functional Acknowledgment. For the advanced adapter this value should point to the current working directory.

qrydirectory

Allows you to specify a directory for the Query File. For the advanced adapter this value should point to the current working directory.

xexdirectory

Allows you to specify a directory for the XML Exception File. For the advanced adapter this value should point to the current working directory.

wrkdirectory

Allows you to specify a directory for the work file. For the advanced adapter this value should point to the current working directory.

rcvdirectory

Allows you to specify a directory for the Receive File. For the advanced adapter this value should point to the current working directory.

trkdirectory

Allows you to specify a directory for the Tracking File. For the advanced adapter this value should point to the current working directory.

plan   Specifies the database plan which will be used in the initialization of the translator. In most cases this should be defined as EDIEC32E which is the default plan name created when WDI is installed. Only change this value if you have created an alias to the database or are using more than one WDI database on the same system.

Languagecode

This specifies the language code that will be used. This should be set to the default which is ENU.

waitinterval

This specifies the number of milliseconds each translator should wait for input before the become idle. When data arrives on an input queue one or more translators will be assigned to the queue. They will continue to consume data arriving on that queue until it is empty. The waitinterval property determines how long they will monitor that queue before they become idle and can be reassigned to another queue.

numtranslators

This value indicates how many translators will be started up when WDIServer is invoked. This is a new parameter that was added to support the new adapter. The old adapter did not start multiple translators so it is ignored by the old adapter.

cmdqname

Indicates the name of the command queue. By default this should be set to WDIAdapterCmd.

userexitname

Specifies the name of the user exit module. When WDIServer is started it will attempt to load the module specified by this property. If it is unable to do so (i.e it could not be found) a warning message will be written to the screen and no exits will be called.

genprtfile

This property tells the adapter under what circumstances a print file should be placed onto the designated print file queue. Valid values are: always, onerror, and never. If the keyword is omitted a print file will be generated only on error conditions. This is a new option that will be ignored by the old adapter.

keeparchive

This property tells the adapter that it is ok to delete work files generated on the disk. During initial setup and testing it can be helpful if these files are not cleaned up after each translation. In production you would not want to keep these files. Valid values are: always, and onerror. If during initial testing and setup it is recommended that you set this to always. Once the configuration is

complete normally this would be changed to onerror so that an archive is created only when an error condition is detected.

MQ Configuration

Once the WDI software has been installed and configured the next step would be to setup the input queues which will be feeding data to WDI. When the WDI software is installed a script called AdvAdapterMQ.txt which contains MQSeries commands to create several new MQSeries objects necessary for the operation of the new adapter. These new objects are:

1. WDI.TRANSLATOR.PROC - Process definition used when triggering the new adapter.
2. WDIAdapterCmd - Command queue used to send commands to the new translator service.
3. WDI.PRTFILE.Q - Queue used to store print files generated during the translation process.

Queue Configuration - Triggering

To use the new adapter one or more input queues needs to be created. These queues need to be setup to generate a trigger message. When setting up the triggering for these queues there are three important values that must be changed. The first is the initiation queue name. You will need to use the same value that was setup in the wdi.properties file for the initq property. The second value that must be setup is the process definition. The process definition for the old adapter was WDI.PROC. To use the new adapter you will need to use the WDI.TRANSLATOR.PROC process definition. The third item that needs to be configured is the Trigger Data. Four options that can be passed to the translator throug h the Trigger Data field. These four options are:

1. NumTranslators() - Specifies the number of translators that will be assigned to this queue when a trigger message is generated.
2. Timeout() - Specifies the amount of time these translators will wait for new data to arrive before they return to the pool to be reassigned to a new queue.
3. ReqId() - allows the user to specify the requester id that will be used when processing data from this queue. If this value is not provided the name of the queue will be used as the requester id.
4. FileId() - allows the user to specify a the file id that will be used when processing data from this queue. The file id identifies the service profile that will be executed when wdi is triggered.

In order for the adapter to work you must at least specify the NumTranslators() and Timeout() options. If these two are not specified the trigger will be generated but no translators will be assigned to the input queue. ReqId() and FileId() are optional. The next section will describe why you would want to specify a value for these parameters.

Using the FileId and ReqId to Allow Large Queue Names

The original WDIAdapter.exe was limited to working with queues which were 8 characters or less in length. This was because the values for the REQID and FILEID keywords were taken from the name of the triggered queue. With the advanced adapter a change was made to allow the values for these two keywords to be specified on the

Trigger Data field. This eliminates the 8 character limit on the queue names. If no value is provided on the Trigger Data field the queue name will be used for the for these keyword values.

Considerations on AIX

The advanced adapter uses shared memory to pass state information between the individual translators and the system monitor process. This means that the WDIServer process will be connecting to one shared memory segment for each translator process and one segment for the trigger monitor. In addition to the segments used by WDI, MQSeries also uses shared memory segments. By default AIX does not allow a 32-bit process to connect to more than 11 shared memory segments (each being 256MB). The AIX operating system provides the EXTSHM environment variable which can be used to allow the 256MB segments to be sub-divided into much smaller segments which conform to the actual size that the process requested when the shared memory attachment is made. If this environment variable is not set and too many translators are started in the pool an error message will be displayed and the adapter will shutdown. To set the EXTSHM environment variable simply add the following line to the profile for the user id that will be running the advanced adapter:

```
export EXTSHM=ON
```

Adapter user exits

To modify or monitor the behavior of the adapter, you can implement the adapter user exits. The WebSphere Data Interchange adapter loads the library, if found in the bin directory, and calls the exit functions. The shared library must be named msgExits.dll (msgExits.so on UNIX®) and should be compiled using the native compiler for the target platform (for example, Microsoft Visual C++ for Windows).

msgExits.dll interface:

- bool msgTrigger(const char* pszTriggerMessage , void * pvExitContext);

  Called when the trigger program is started. Passes the trigger message TQTMC2. Accepts a context that will be passed into all subsequent calls. The return value indicates whether to continue or terminate.

- bool bSkip msgArrival(void* pvExitContext, char*pszSessionID)

  Message tracking exit that will be called just before attempting to get the next message. It can browse the queue for any information required and then pass back a session ID for WebSphere Data Interchange to use as the Batch ID. The return value indicates whether to process the message or skip it.

- bool bProceed msgTransform(void* pvExitContext, long rc, long ccbrc, long ccberc)

  Results of the transformation. Return values are:

  - SYNC_CONTINUE - syncpoint and then continue processing.
  - SYNC_TERM - syncpoint and then terminate.
  - CONTINUE - roll back the current message and continue processing from the input queue.
  - TERMINATE - roll back the current message and terminate the adapter.

- bool bOK msgTerminate(void* pvExitContext)

Triggering from an MQSeries queue

OK to terminate? Return values from msgTerminate are:
- #define SYNC_CONTINUE X'0000'
- #define SYNC_TERM X'0001'
- #define CONTINUE X'0002'
- #define TERMINATE X'0003'

A configuration file, `wdi.properties`, is installed in the wdi/bin directory:

The WebSphere Data Interchange Server runtime information is stored in a properties file located in the WDI bin directory. The installation default values for Windows are listed below. The default values for AIX are similar.

```
runtimedirectory=C:\WDIServer32\bin
datadirectory=C:\WDIServer32\runtime
dtddirectory=dtds
prtdirectory=prt
appdirectory=adf
edidirectory=edi
eexdirectory=eex
xmldirectory=xml
aexdirectory=aex
rptdirectory=rpt
fakdirectory=fak
qrydirectory=qry
xexdirectory=xex
wrkdirectory=wrk
rcvdirectory=rcv
trkdirectory=trk
plan=EDIEC32E
userid=user
userpassword=password
Languagecode=ENU
waitinterval=30000
```

The `userid` and `userpassword` fields are only required if you want to connect to the database using a different authorization ID than the one under which the adapter process is running.

All files created and used during run time are created under the data directory. If you want WebSphere Data Interchange to create files in a different directory, change the data directory value in wdi.properties to be the new directory that you created. This directory can be another drive on Windows or file system on AIX.

The waitinterval value specifies the number of milliseconds that the adapter waits for messages on the Application queue before terminating. When the adapter terminates, another trigger message restarts the adapter.

Calling from a C++ program

This section provides an example of how to use the WebSphere Data Interchange C++ API. This example includes all the source code necessary to build a C++ program, using the WebSphere Data Interchange C++ API, to send several PERFORM commands to the WebSphere Data Interchange product.

Elements of the C++ API

The C++ API is made up of several classes that are all defined in the diapi.h header file shipped with the WebSphere Data Interchange product. The classes that make up the API are:

- CSyncTranslator
- CASyncTranslator
- CRemoteTranslator
- CDIEnvironment
- CDIRequest

When a program includes the diapi.h header file, it can use these objects to interact with the WebSphere Data Interchange translator by passing in PERFORM commands to either a CSyncTranslator, CASyncTranslator, or CRemoteTranslator object. The three different types of translator objects that can be created are as follows:

1. CSyncTranslator

   The CSyncTranslator provides access to the translator in a synchronous manner. The process waits for each command to complete before allowing the next command to be performed.

   Its methods are:

   - **CSyncTranslator(void)** (Constructor)

     Instantiates a new CSyncTranslator object. This method takes no arguments.

   - **enum eResult Initialize(CDIEnvironment& env)**

     Causes the translator to be initialized. This method must be called before any transactions can be processed. This method takes a CDIEnvironment object that contains information about the system, such as database information (plan, user ID, password) and system information (language). This method returns an enumerated type that contains success or failure information about the method invocation.

   - **enum eResult Terminate()**

     Terminates the translator and causes it to free any memory that was allocated during the translation process. This method takes no arguments and returns an enumerated type with information about the success or failure of the method invocation.

   - **virtual enum eResult ProcessRequest(CDIRequest& req)**

     Initializes a PERFORM command to be processed by the translator. This method takes a CDIRequest object that has been initialized with a perform command. The enumerated type returned by the function can be used to determine the success or failure of the PERFORM command.

   - **virtual long GetRetCode(void)**

Gets the return code from the last translator action performed. This method takes no arguments.

- **virtual long GetExtRetCode(void)**

Accesses the extended return code from the last translator action performed. This method takes no arguments.

2. CASyncTranslator

The CASyncTranslator provides asynchronous access to the translator. This method allows your program to begin processing on several transactions at once without waiting for the previous PERFORM command to complete.

Its methods are:

- **CAsyncTranslator(short sMaxReqs=10, short nNice=0)**

Takes two arguments: sMaxReqs and nNice. sMaxReqs specifies the maximum number of requests that can be made. nNice specifies a nice value any process created by the translator during this session.

- **enum eResult Initialize(CDIEnvironment& env)**

Initializes the translator using the CDIEnvironment argument. The CDIEnvironment object contains the system environment information, such as the database plan, user ID, and password, as well as the system language setting.

- **enum eResult Terminate()**

Initializes the translator using the CDIEnvironment argument. The CDIEnvironment object contains the system environment information, such as the database plan, user ID, and password, as well as the system language setting.

- **enum eResult ProcessRequest(CDIRequest& req)**

Passes the perform command contained in the CDIRequest object to the translator to be executed.

- **short GetMaxRequests()**

Returns the maximum number of requests that can be executed at one time. This value limits the number of translators that can be started by this object.

- **short GetCurrentRequests()**

Returns the number of requests being processed.

- **enum eResult UpdateCurReqCnt(void)**

Causes the current request count to be updated.

3. CRemoteTranslator

The CRemoteTranslator provides access to a WebSphere Data Interchange translator running on a remote system. The CRemoteTranslator is like the CAsyncTranslator, except its constructor takes the hostname of the remote system as an additional argument to its constructor.

Its methods are:

- **CRemoteTranslator(char* pszHost,short sMaxReqs=10)**

Creates a new instance of the CRemoteTranslator class that can communicate with a remote server using TCP/IP sockets.

- **enum eResult Initialize(CDIEnvironment& env)**

Initializes the translator using the CDIEnvironment argument. The CDIEnvironment object contains the system environment information, such as the database plan, user ID, and password, as well as the system language setting.

- **enum eResult Terminate()**

    Initializes the translator using the CDIEnvironment argument. The CDIEnvironment object contains the system environment information, such as the database plan, user ID, and password, as well as the system language setting.

- **enum eResult ProcessRequest(CDIRequest& req)**

    Passes the PERFORM command contained in the CDIRequest object to the translator to be executed.

- **short GetMaxRequests()**

    Returns the maximum number of requests that can be executed at one time. This value limits the number of translators that can be started by this object.

- **short GetCurrentRequests()**

    Returns the number of requests being processed.

- **enum eResult UpdateCurReqCnt(void)**

    Causes the current request count to be updated.

The following classes are also provided:

- CDIEnvironment

    The CDIEnvironment class encapsulates all the system settings needed by the CSyncTranslator, CAsyncTranslator, and CRemoteTranslator during their initialization. The CDIEnvironment class must be instansiated and then passed to the initialize method of one of the translator objects.

    Its methods are:

    - **void SetSys(char* pszVal)**

        Identifies the installation-defined WebSphere Data Interchange systems used to run the EDIUTILV utility. The default is DIENU.

    - **void SetAppl(char* pszVal)**

        Identifies the Application ID to run the DataInterchange utility. This keyword also identifies the logfile specified by the Activity Log (ACTLOGS) profile. If you specify this parameter, the activity log profile must contain a matching entry to define which log file is used for recording errors and events pertaining to the application. The two APPLID values shipped with WebSphere Data Interchange are:

        - EDIFFS (default)
            - Associated with the LOGFFS ddname
            - The default APPLID and log when using the utilities
        - EDIMP
            - Associated with the LOGEDI ddname
            - The APPLID and log used during online DataInterchange processing

    - **void SetLang(char* pszVal)**

Identifies the language profile to use as specified in the Language (LANGPROF) profile. The value you specify with the SetLang method must match one of the values in the LANGPROF profile. The LANGPROF that ships with WebSphere Data Interchange is ENU.

– **void SetPlan(char* pszVal)**

Identifies the DB2 plan that WebSphere Data Interchange is to use to access its database tables.

– **void SetEdiDataQueueName(char* pszVal)**

Identifies the routing queue for completion message from the translator. When the CAsyncTranslator completes the processing of its EDI data it will send a completion message to this queue.

– **void SetAppDataQueueName(char* pszVal)**

Identifies the routing queue for completion message for DF data coming from the translator. When the CAsyncTranslator completes the processing of a DF, it will send a completion message to this queue.

– **void SetEdiErrorQueueName(char* pszVal)**

Identifies a queue where errors identified during the processing of EDI data can be sent. When the CAsyncTranslator encounters errors, an EDI data message will be sent to this queue.

– **void SetAppErrorQueueName(char* pszVal)**

Identifies a queue where errors identified during the processing of DF data can be sent. When the CAsyncTranslator encounters errors, a DF data message will be sent to this queue.

– **void SetHostName(char* pszVal)**

Sets the hostname of a remote WebSphere Data Interchange translator. The value set in this method is only used with the CRemoteTranslator.

– **void SetHostPort(int nVal)**

Identifies the port number used by a remote WebSphere Data Interchange translator for network communication. The value set by this method is only used by the CRemoteTranslator class.

– **void SetUser(char* pszVal)**

Sets the database user ID needed to access the DB2 database. This only needs to be set if the user ID of the person running the program does not have the necessary authority to access the database.

– **void SetPassword(char* pszVal)**

Sets the password to be used by the translator to access the WebSphere Data Interchange database. This is only required if the user ID of the person running the program does not have the authority to access the database.

– **void SetRouterType(enum CDIMsgQueue::qtype enVal)**

Sets the type of routers (queue) for the completion messages. This method can accept the following values:

```
file      //File
pipe      //Named pipe
socket    //TCP/IP socket
email     //Email address
```

– **void SetUnitOfWork(enum eUnitOfWork enVal)**

Defines a unit of work to the translator. This method allows the application programmer to define the point at which Commits should be done. Possible values for this function are:

eTransaction
> Commits should be done after every transaction

eEnvelope
> Commits should be done after every envelope is encountered

eNoCommit
> No commits are performed

– **void SetInterfaceType(enum eInterfaceType enVal)**

Possible values for the function are:

```
eITCmdLine    //Invoked by command line
eITApi    //Invoked by API
eITWeb    //Invoked by Web server (not supported)
```

- CDIRequest

  The CDIRequest represents a request for translation that can be submitted to a translator to be processed. The CDIRequest object names the files necessary to perform a translation as well as the perform statement to be executed. This is a list of the files that can be associated with a CDIRequest object:
  – Application File
  – EDI File
  – Tracking File
  – Exception File
  – EDI Except File
  – Print File
  – Report File
  – Query File
  – Work File
  – Functional Acknowledgment File

  Its methods are:

  – **CDIRequest(void)**

    The constructor for CDIRequest builds an instance of the CDIRequest object. This method does not take any arguments.

  – **void ClearOutput(void)**

    Clears all the output fields of the request object.

  – **void SetAppFile(char* pszFile)**

    Sets the name of the application file for this request.

  – **void SetEdiFile(char* pszFile)**

    Sets the name of the EDI file for this request.

  – **void SetTrackingFile(char* pszFile)**

    Sets the name of the tracking file for this request.

  – **void SetExceptionFile(char* pszFile)**

Sets the name of the exception file where DF data can be written if a fatal error is encountered during the processing of the PERFORM command.

– **void SetEdiExceptFile(char* pszFile)**

Sets the name of the exception file where EDI data can be written if a fatal error is encountered during the processing of the PERFORM command.

– **void SetPrintFile(char* pszFile)**

Sets the name of the print file where status information about a completed translation can be written.

– **void SetReportFile(char* pszFile)**

Sets the name of the report file where reports and printouts that you have requested can be stored.

– **void SetQueryFile(char* pszFile);**

Sets the name of the file where results from a QUERY or DATA EXTRACT can be stored.

– **void SetWorkFile(char* pszFile)**

Sets the name of a file that can be used as a temporary workspace during the translation.

– **void SetFunAckFile(char* pszFile)**

Sets the name of the file that you want to use for returning functional acknowledgments for the deenveloped transactions.

WebSphere Data Interchange API example

This section outlines an example program that uses the C++ API for WebSphere Data Interchange. This example is made up of one source file that initializes a CDIRequest object with a PERFORM TRANSFORM statement. The "Sample script" on page 595 contains a listing of the main items for this example.

This program begins by creating CSyncTranslator, CDIEnvironment, and CDIRequest object (a TransformRequest) to handle the request. Next, the request is initialized with the filenames needed to process the request, and the setPerformCommand method is used to set the desired PERFORM command to execute.

When the PERFORM command is created and the translator initialized, the ProcessRequest method is called to execute the PERFORM command. When that is complete, the return codes are checked and the translator terminated and the program is exited.

Building the Example

This example can be built on the different platforms supported by WebSphere Data Interchange.

- AIX
    1. Copy the files apiexamp.cpp and apiexamp.mk from `/usr/wdi/DIv32/samples` to your home directory or any other work directory where you have permissions to write files. Also copy the sample input file poxml5sr.dat to the same directory.

2. Issue the command make -f apiexamp.mk. This will build an executable named apiexamp.

3. Execute the newly built apiexamp executable.

- Windows

  1. Using Microsoft Visual C++, open the ediexamp.dsw workspace in the samples directory.

  2. Build the apiexamp project. The executable will be created in the samples directory.

  3. Execute the newly built apiexamp executable.

Sample script

```cpp
#include <iostream.h>
#include <string.h>
#include "diapi.h"

/*-------------------------------------------------------------------*/
/* Main program                                                      */
/*-------------------------------------------------------------------*/
void pause(void);

int main ()
{
   CDIEnvironment   aCDIEnvironment;
   CDIRequest       aTransformRequest;
   CSyncTranslator  aCSyncTranslator;
   enum eResult     rc;
   char*            pszPhysicalName = NULL;
   long             lFileLen = 0;

   // Let the user know what we're doing:
   cout << endl
      << "XML to EDI sample transformation using the C++ API" << endl;

   //Define the Data Interchange Environment
   aCDIEnvironment.SetPlan("EDIEC32E");
   aCDIEnvironment.SetLang("ENU     ");

   // Initialize the translator:
   cout << endl << "Initialize the translator by calling "
      "CSyncTranslator::Initialize()" << endl;
   rc = aCSyncTranslator.Initialize(aCDIEnvironment);
   cout << "Initialize() returns:  rc=" << rc << ", zccbrc="
      << aCSyncTranslator.GetRetCode() << ", zccberc="
      << aCSyncTranslator.GetExtRetCode() << endl;

   //pause();

   // Let the user know we are setting up the files and command:
   cout << endl << "Set the input/output filenames and the "
      "command by calling " << endl
      << "CSyncTranslator::SetFileName and "
      "CDIRequest::SetPerformCmd." << endl;
```

Calling from a C++ program

```cpp
// Name the input and output files:
aCSyncTranslator.SetFileName("XMLFILE", "poxml5sr.dat");
aCSyncTranslator.SetFileName("OUTFILE", "sample.out");
aCSyncTranslator.SetFileName("FFSEXCP", "sample.aex");
aCSyncTranslator.SetFileName("PRTFILE", "sample.prt");

// Set the perform commands to be executed:
//  XML-TO-EDI TEST CASE:  *******************************************
aTransformRequest.SetPerformCmd
("PERFORM TRANSFORM WHERE INFILE(XMLFILE) OUTFILE(OUTFILE) "
   "SYNTAX(X) CLEARFILE(Y) XMLEBCDIC(N) TRACELEVEL(A2)");

// Let the user know we are going to do the translations
cout << endl << "Translate the document by calling "
   "CSyncTranslator::ProcessRequest." << endl;

// Ask the synchronous translator to process the EDI to ADF Request:
rc = aCSyncTranslator.ProcessRequest(aTransformRequest);

// Let the user know what happened:
cout << "CSyncTranslator::ProcessRequest() returns:  rc=" << rc
     << ", zccbrc=" << aCSyncTranslator.GetRetCode()
     << ", zccberc=" << aCSyncTranslator.GetExtRetCode() << endl;

// Confirm the input and output names and print the return codes:
rc = aCSyncTranslator.GetFileName(&pszPhysicalName, "XMLFILE ", &IFileLen);
cout <<"Input file was    : " << pszPhysicalName << ", "
     << lFileLen << " bytes written" << endl;

rc = aCSyncTranslator.GetFileName(&pszPhysicalName, "OUTFILE ", &IFileLen);
cout <<"Output file was    : " << pszPhysicalName << ", "
     << lFileLen << " bytes written" << endl;

rc = aCSyncTranslator.GetFileName(&pszPhysicalName, "FFSEXCP ", &IFileLen);
cout <<"Exception file was: " << pszPhysicalName << ", "
     << lFileLen << " bytes written" << endl;

rc = aCSyncTranslator.GetFileName(&pszPhysicalName, "PRTFILE ", &IFileLen);
cout << "Print file was    : " << pszPhysicalName << ", "
     << lFileLen << " bytes written" << endl;
//pause();

// Terminate the translator:
cout << endl << "Now terminate the translator to free up "
   "any resources" << endl;
rc = aCSyncTranslator.Terminate();
cout << "CSyncTranslator::Terminate() returns:  rc=" << rc << endl;
cout << endl << "Note:  If rc=0, then DO NOT check RetCode and "
   "ExtRetCode because " << endl
   << "the CSyncTranslator is now uninitialized." << endl;

// Let the user know were done:
cout << endl << "XML to EDI sample transformation complete." << endl;
```

```
    // Terminate and go home:
    return(0);
}

void pause()
{
   char achar;
   cout << "Hit enter to continue" << endl;
   cin.get(achar);
}
```

Calling from a C++ program

Chapter 8. Interfacing to other networks and applications

This chapter describes the interfaces between your application, WebSphere Data Interchange, and the network. You can use this chapter to develop programs that communicate with applications and networks.

All communication requests by WebSphere Data Interchange or by applications using WebSphere Data Interchange are made through the Communications API. For more information, see "Communication services" on page 540. WebSphere Data Interchange includes support for multiple networks and allows you to write programs to add support for any network that is not supported. WebSphere Data Interchange uses profiles that define the network (Network Profile (NETPROF)) and define the operations supported by that network (NETOP). The communications API requests listed below represent the minimum support required from a network for WebSphere Data Interchange to function properly on that network.

- Send transactions (function code **211**) with network commands of SENDEDI and SENDFILE.
- Send files (function code **221**) with network command of SENDFILE.
- Receive files (function code **232**) with network commands of RECVEDI and RECVFILE.
- Cancel (function code **233**) with network command of CANCEL.
- Return file name (function code **300**).
- Retrieve status (function code **252**) with network command of RECVMSG.

For more information about API requests, see Chapter 6, "Using WebSphere Data Interchange in the z/OS environment," on page 407.

WebSphere Data Interchange supports two types of networks:

- A generalized network with:
  - A network program provided by the network provider, such as IEBASE from IBM Global Services and DSXMIT2 from the General Electric Information Services Company
  - A file interface to the network program where commands are placed in an input file by the requestor and responses are placed in the output file by the network program. WebSphere Data Interchange has no control over this interface since it is defined by the network provider
  - The ability to process multiple interchanges in a single file and use information from the interchanges to determine the destination
  - A connection made to the network itself rather than any particular trading partner, holding data in a mailbox until requested by the trading partner
- A point-to-point network with:
  - A generalized communications package rather than a specific program designed for a particular network.
  - A direct connection to the trading partner.
  - Files that contain data for a single trading partner.

Some details of each interface are provided in the following sections.

Generalized networks

The application-to-network flow diagram Figure 26 on page 603 illustrates how an API request from an application flows into the WebSphere Data Interchange communications module. The communications module reads all the necessary profiles and invokes the communications routine defined to handle the network. The name of the communications routine is provided in the `Communication rtn` field of the network profile (NETPROF). The communications routine passes information between the application and the network program and must:

• Process the API request
• Build the appropriate interface to the network program
• Interpret the results from the network program

If you are providing an interface to your own network, you must:

1. Write a message handler to process the responses generated by the network program. The logical name of the message handler is specified in the `Message handler` field of the Network profile. The physical load module name and the implementation language for the message handler are specified in the user program information (User Exits (ADAMCTL)) profile. For more information, see "Message handler" on page 612.

2. Design and enter the network commands profile (NETOP) entries to build the commands required by the network program.

3. Determine which of the communications routines provided by WebSphere Data Interchange that you want to use.

The seven logical names you can use in the `Communications routine` field of the network profile are:

• VANINFC
• VANEXPV4
• VANIINB1
• VANIMQ
• VANICICS
• GEISVAN
• PTTOPT

There are slight differences in the operation of these programs, and you should choose one that meets your needs. The differences are:

• VANEXPV4 is for CICS only and is an alternative to VANINFC. For more information, see "Sent to Network status" on page 380.

• VANICICS is for CICS only, fills in several pre-defined data blocks, and then passes control to the user-supplied network program specified in the `Network program` field of the network profile. For more information, see "Special communications routine for CICS" on page 613.

• PTTOPT identifies the point-to-point network. For more information, see "Point-to-point networks" on page 606.

• VANIMQ is for the exclusive use of MQSeries send and receive.

- VANIINB1 changes a value of **G** in the `RECVTYP` field of the CMCB to a value of **N**. For more information, see "Communication Control Block (CMCB)" on page 677.
- If the return code from the network program is zero, VANIINB1 does not update the status of the interchanges but assumes that the message handler updates the status of all interchanges. The other programs update the status based on the return code and assume that the message handler changes the status if necessary.
- VANINFC updates the network sequence value once for each interchange in a file. The other programs update the sequence number once for the entire file.
- VANIINB1 and GEISVAN request status for a single requestor at a time. The other programs request status for 10 requestors at a time.
- GEISVAN uses a network command of STATUS to request status from the network. The other programs use a network command of RECVMSG.
- The message handler for GEISVAN only processes the status update responses from DSXMIT2 during a STATUS request. All other responses from DSXMIT2 are processed directly by GEISVAN. For the other programs, the message handler processes all responses.
- GEISVAN asks for status to be placed into a file with a ddname of GEISTAT. VANIINB1 asks for status to be placed into a file with a ddname of INB1STAT. The other programs assume the status is written in the network output file.
- The default TD queue for GEISVAN is GEISQ. The other programs have a default value of QDATA.
- The default network input file for GEISVAN is DSXMIPT. The other programs have a default value of INFILE.
- The default network output file for GEISVAN is SYSOUT. The other programs have a default value of OUTFILE.
- GEISVAN does not pass network parameters to the network program.
- GEISVAN does not expect a return code from the network program. The other programs expect a return code with the following values and meanings:
  - **<0**   Serious error. Status is send request error (42).
  - **0**   Request processed without error. Status is send requested (48).
  - **1 – 4**   Request processed with warnings. Status is sent with errors (41).
  - **>4**   Request processed with errors. Status is send request error (42).
- GEISVAN does not assume RESTART is an option. The other programs check the `FRESTART` field of the FSUPPORT network command and invoke the network program with a RESTART option if restart is supported by the network.

The communications routine calls the network program with an z/OS ATTACH macro after the commands dictated by the network commands profile (such as SENDEDI, RECVFILE) have been written to the input file. The name of the input file is contained in the `Network input file` field of the network profile entry and the commands written to this file are created from directions stored in the Network Commands profile (NETOP) entries. The Network Commands entries contain instructions for pulling data from various sources along with literals to create commands. For more information, see "Building network commands" on page 608. The network program be written in any language with any attributes because it is accessed using an z/OS ATTACH macro. The parameters passed to the network program are specified in the `Network parameters` field in the network profile entry.

Generalized networks

A communications routine in combination with NETOP should be able to create the commands necessary for the network. Each network produces responses that have a unique format and are written to the output file identified by the `Network output file` field in the network profile entry. Therefore, if you are providing your own network you must provide a program (known as a message handler) that processes those responses to detect errors and perhaps to update the status of interchanges in WebSphere Data Interchange. The name of the message handler program is entered in the `Message handler` field of the network profile entry, which must be defined in the user program information profile (User Exits). For more information, see "Message handler" on page 612.

Application-to-network flow diagram
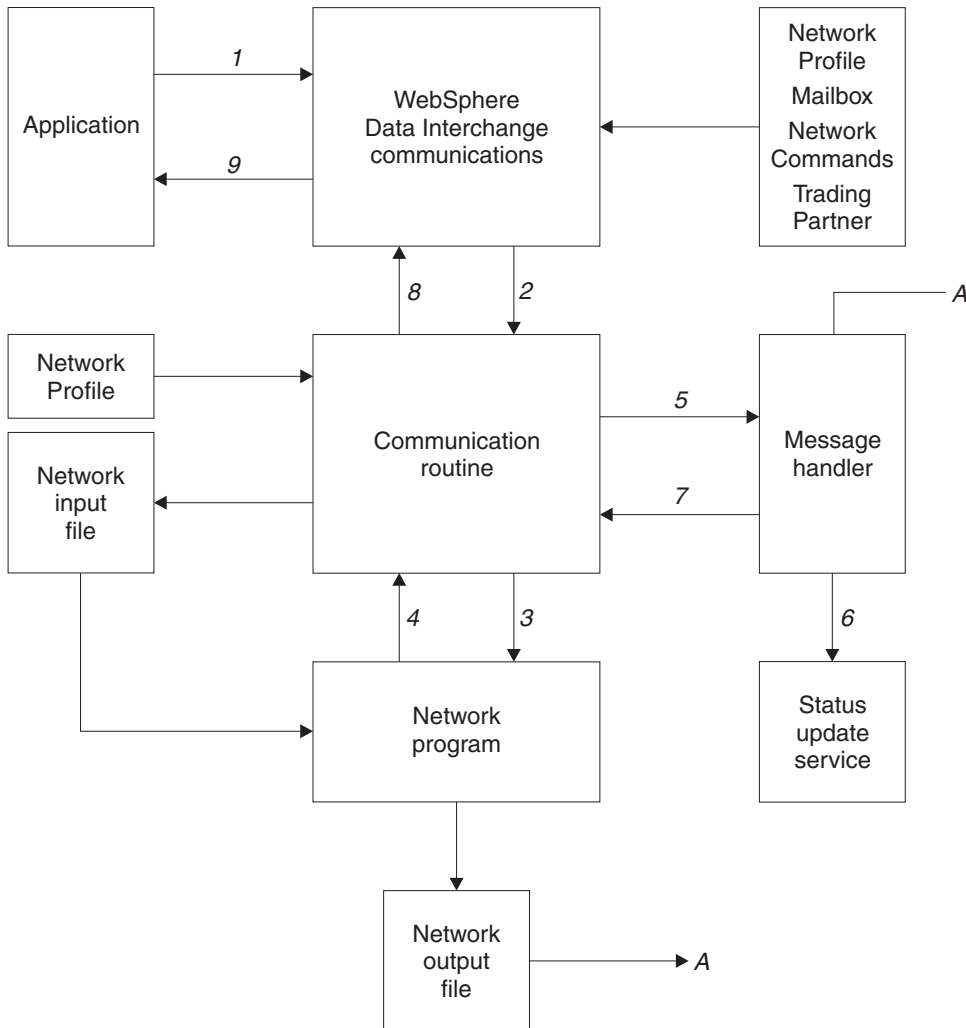
Figure 26. Application-to-network flow diagram

The numbers in the diagram refer to the following sequence of events:

1. An application program requests WebSphere Data Interchange communication services through the API defined in "Communication services" on page 540. The request retrieves:

   - The mailbox (requestor) profile member (REQPROF) identified by the REQID field of the communications control block.
   - The trading partner profile member (TPPROF) identified by the TPNICKNM field of the communications control block. This retrieval occurs only if the trading partner data block passed to communications contains blanks.

- The network profile member (NETPROF) identified by the requestor member or by the `NETID` field in the CCB.
- The functions supported by the network identified in the FSUPPORT member in the network commands profile (NETOP).

2. Using the program identified in the `Communication rtn` field of the network profile member, communications invokes this program as a user-exit routine through the language interface routine (FXXZ*ccc*). The communications routine name is a logical name and must be one of the following:
   - VANINFC
   - VANIINB1
   - VANEXPV4
   - PTTOPT
   - GEISVAN

   VANICICS and VANIMQ are (not applicable for this flow.

3. The communication routines has control. Communications routines are driven by the function requested (send, receive, or cancel) and by the network command specified in the `NETOP` field of the communications control block. The 8-byte network ID and the 8-byte network command are concatenated and used as a partial key to retrieve all matching entries from the Network Commands profile (NETOP). Network Commands members contain instructions for building commands, which are literal values combined with data from the other control blocks and profile members. The commands are written to a file as specified by the `Network input file` field from the network profile.

   If transaction data is being sent (function code **211**), the file of transactions is scanned and each interchange in the file is updated to have a status of **Send started** (**46**). The following fields are also associated with the interchange:
   - Message name (MSGNAME)
   - Message user class (ENAME)
   - Message sequence number (SEQNUM)
   - Network acknowledgment expected (ACKIND)
   - Send date
   - Send time

   When commands have been built and written to the network input file and status has been updated, the program identified in the `Network program` field is invoked using the z/OS ATTACH facility. The parameters passed to the network program are specified in the `Network parameters` field of the network profile. Network parameters are not passed to the network program when the communications routine is GEISVAN.

4. This is a synchronous operation where the communication routine waits until the network program is finished. When that program returns control, it is assumed that the operation is completed or that the network program is responsible for making sure the operation completes.

   A return code of **0** from the network program indicates success. Return codes **1-4** indicate no serious errors. A return code of less than **0** or greater than **5** indicates the function failed completely.

   Note: A return code of **0** is assumed when the communication routine is GEISVAN.

5. If sending transaction data is requested, the communication routine updates the status in the Transaction Store of each interchange in the file based on the return code from the network program. A return code of **0** results in a status of **Send requested** (**48**). Return codes **1-4** result in a status of Sent with errors (**41**). Any other return code results in a status of **Send request error** (**42**).

    Note: If the communication routine is VANIINB1 and the return code from the network program is zero (**0**), the communication routine does not update the status. In this case, the message handler must update the status of each interchange in the file. This is because VANIINB1 is targeted for the Expedite network program and when Expedite has a return code of **0**, it has written an entry for each interchange to the network output file (OUTMSG). The message handler for Expedite can read these records and update the status of each interchange.

    However, if the return code from the network program is not **0**, VANIINB1 sets the status of each interchange to Send request error (**42**), and the message handler must update the status for those interchanges that were successfully sent. With a nonzero return code, the assumption is that the network program not have been able to (or chose not to) complete its processing. Therefore, the network output file (OUTMSG) not contain an entries for all interchanges, but only entries for those that were successfully processed.

    For example, if you are sending a file that contains four X12 interchanges but the second interchange in the file has an error that prevents the network program from processing it (such as an unknown destination), VANIINB1 will initialize the status of all four interchanges to Send request error (**42**), because the return code from the network program should not be zero now (identifying the error).

    Assuming the network program continued processing after the error, the message handler updates the status of the first, third, and fourth interchanges to Send requested (**48**). If the error in the second interchange was severe enough to cause the network to stop processing (such as an I/O error reading the file), only the status of the first transaction is updated by the message handler to Send requested (**48**), and the status of the other three interchanges remains Send request error (**42)**.

Because a message handler is driven by the responses in the network output file created by the network program, it only updates the status of interchanges it has been told about through the responses of the network program. The VANIINB1 communication routine assumes that if the return code from the network program is **0**, a response record of some type is written to the network output file for every interchange in the file so that the message handler can update the status.

The communication routine uses the `Message handler` field from the network profile member and invokes this program as a user-exit routine through the language interface routine (FXXZ*ccc*). For more information about message handler programs, see "Message handler" on page 612.

6. If the network output file contains status information about previously sent data, the message handler should use the status update service to update the status information in the Transaction Store. For more information, see "Update status services" on page 561.

7. If the network output file indicates that data has been received, the message handler should set the `FILERCVD` field in the communications control block to **Y** and put the account number and user ID of the trading partner sending the data into the trading partner data block.

8. The communication routine uses a reverse lookup on the trading partner profile members to locate the entry belonging to the returned account number and user ID. The trading partner nickname is returned to the application program so that it knows data was received and from what trading partner. If any errors were encountered, the communication routine writes an entry to the event log indicating the type of error that occurred. The contents of the `NPSEVER` and `NPERRCD` fields from the communications control block are included in the VN1015 message logged by the communications routine.

   The communication routine acts on the value in the `CLRFILE` field from the communication interface control block, clearing a file just sent if requested to do so.

9. The application regains control with the results of the operation indicated by the `ZCCBRC` and `ZCCBERC` fields of the CCB.

Point-to-point networks

WebSphere Data Interchange for z/OS provides a communications routine called PTTOPT for sending and receiving documents through a point-to-point connection. The point-to-point connection allows you to direct transactions to a file not associated with a network. To use a point-to-point network, you specify the `Communication rtn` field of the network profile (NETPROF).

When the PTTOPT communications routine receives a request to queue a transaction, the transaction is written to a system-generated file created specifically for the trading partner. The file name is a concatenation of the current user ID and the trading partner nickname, with a period inserted every eight characters. For example, if the user ID is MKRUEGER and the trading partner nickname is JWILLIAMS, the file name is MKRUEGER.JWILLIAM.S. The file must already exist, and because the file is allocated dynamically, there are no JCL or CLIST requirements for the file. Transactions are appended to the end of the file; you must clear the file of transactions that are successfully sent to the trading partner.

PTTOPT forwards all other requests to your send/receive program, passing the same parameters that were passed to it. Your program sets the return codes and control block information that are expected by the requesting application. For more information, see "Parameters passed to the communications routine" on page 607

Activating point-to-point connections

To activate point-to-point communications, follow these steps:

1. Add a member to the user program information profile (User Exits) for the point-to-point program.

2. Add a point-to-point member to the network profile by specifying the following:
   - **PTTOPT** in the `Communication routine` field
   - The logical name of your send/receive program in the `Network program` field
3. In the `Network ID` fields of the trading partner profile members for the point-to-point trading partners, enter the name of the point-to-point member that you added to the network profile in step 2 above.
4. Enter the name of the network profile member in the `Network ID` field of the mailbox (requestor) profile members for the point-to-point requestors.
5. Add a member called *netid* FSUPPORT to the network commands profile, where *netid* is the network ID in your network profile. The literal entered in the command field () of this member identifies the functions that are supported by the point-to-point connection.

   You can create the point-to-point member by copying the member FSUPPORT and changing the network ID and LITERAL VALUE values. For details about the LITERAL VALUE literal, see "FSUPPORT member" on page 611

Point-to-point network processing flow

The flow for a point-to-point network is similar to the flow for the generalized network (described in "Application-to-network flow diagram" on page 603) with the following exceptions:

1. The network program is called as a WebSphere Data Interchange exit rather than using an z/OS ATTACH. You must define the program in the User Exits (ADAMCTL) profile and write the program in a language supported by WebSphere Data Interchange. See Chapter 4, "Exit routines," on page 311 for a description of user exits. The parameters passed to the point-to-point network program are the same as those passed to the communications routine.
2. The network program receives the same parameters as the communications routine (a programming interface rather than a file interface).
3. There is no message handler for a point-to-point network. The network program is responsible for processing the requests and for processing the results expected by the API.

Parameters passed to the communications routine

Communications invokes the communications routine to process each request from an application that requires network activity, such as requests to send or receive transactions. Your application program supplies the required values before calling communication. For more information, see "Communication services" on page 540.

The data supplied by the application is combined with information from the profiles, and the parameters listed below are passed to the communications routine. For point-to-point networks (`Communication rtn` has a value of **PTTOPT**), these same parameters are sent to the network program. For more information about control blocks, see Appendix A, "WebSphere Data Interchange control blocks," on page 631
- Service Name Block (SNB)
- Common Control Block (CCB)
- Function Control Block (FCB)

Parameters passed to communications routine

- Communication Interface Control (CMCB)
- Trading Partner Profile (TPPDB)
- Network Profile Block (NPDB)
- Mailbox (Requestor) Profile Block (REQDB)

Building network commands

A communications routine (such as VANIINB1) uses a network commands profile to build command records that it places in a network input file (such as INMSG). The information needed to build the command records is in the control blocks passed to the communication routine. Entries in the network commands profile tell the communication routine what data from the control blocks should be used in building the commands for the network program.

Network commands profile

Table 208 describes the network commands profile. Each member of the profile provides data for a particular field in a command record. The `command record` field is defined by the `SEQUENCE`, `POSITION`, and `LENGTH` fields of the Network Commands (NETOP) profile member. The data for the command record field can either be a literal value (**LITERAL VALUE**) or it can be a value from one of the WebSphere Data Interchange control blocks (BLOCK NAME or BLOCK POSITION).

The keys to a profile member are the network ID (`NETID`), network commands ID (`NETOP`), and network commands field sequence number (`FLDSEQ`).

Table 208. Definition of the network operation profile

Label	Length	Type	Description
BLOCK NAME	8	Char	Name of block from which data is taken
BLOCK POSITION	4	Char	Block position
LENGTH	4	Char	Command field length
LINE	4	Char	Command line sequence number
LITERAL VALUE	58	Char	Command field value
NETWORK COMMAND	8	Char	Network commands ID
NETWORK ID	8	Char	Network ID
POSITION	4	Char	Command field position
SEQUENCE	8	Char	Network commands field sequence number

Network commands profile field descriptions

BLOCK NAME

The name of the block from which data is to be taken or the name of a network command. Valid values are:

EDICMCB

Communication interface control block

EDINPDB
>   Network profile block
EDITPPDB
>   Trading partner profile block
EDIREQDB
>   Mailbox (Requestor) profile block
(blank) LITERAL VALUE field contents

BLOCK POSITION

The starting position of the source data within the block identified in `BLKNAME`. The position is relative to 1.

LENGTH

The length of data to move from the source location (or `BLKPOS`) to the command record (or `CMDFPOS`). The length of a command record is defined in the `Input record length` field of the network profile.

LINE

The sequence number of the command record to which the source data should be moved, or enter an asterisk to use the current command line.

LITERAL VALUE

A literal used in the command field that applies only when `BLKNAME` is blank. When `BLKNAME` is not blank, this field can contain comments. This field can also contain one of these special literals:

//IMBED//
>   Uses the network command indicated by the `BLKNAME` field

//STRIP//
>   Removes trailing blanks from data moved to the command line

NETWORK COMMAND

The name of a network command, such as SENDX12. These names match those in the NETOP field of the communication interface control block. Must be eight characters, left-justified, and padded with trailing blanks.

NETWORK ID

The name (key) that identifies the network. The network commands profile must contain a member with this name. Must be eight characters, left-justified, and padded with trailing blanks.

POSITION

The starting position in the command record where the source data should be moved, or an asterisk to use the current field position.

SEQUENCE

The sequential number of this entry in the network command. The sequence number has two parts:

Building network commands

- The first 4 bytes contain the command line number.
- The last 4 bytes contain the field sequence number in the line.

Network command example

Table 209 describes the network command entries used to build the commands for sending X12 standard transactions over the network. The NETWORK ID (**IINR4**), NETWORK COMMAND (**SENDX12**), and SEQUENCE (**1-n**) fields are not shown in the table.

Table 209. Network command example

BLOCK NAME	BLOCK POSITION	SEQUENCE	POSITION	LENGTH	LITERAL VALUE	Notes
		1	1	3	CSS	Start session command
EDIREQDB	38	1	4	8		Account
EDIREQDB	70	1	12	8		User ID
EDIREQDB	102	1	20	16		Old and new password
EDINPDB	148	1	36	5		Time zone
EDINPDB	153	1	41	8		System type
EDINPDB	161	1	49	4		System level
		2	1	3	CSP	Send EDI File command (part 1)
EDITPPDB	360	2	4	1		Network message class
EDITPPDB	361	2	5	1		Message charge code
EDICMCB	81	2	6	1		Acknowledgment type
EDICMCB	99	2	7	8		Message name
EDICMCB	61	2	15	5		Message sequence number
EDICMCB	80	2	20	1		Message delivery class
EDICMCB	91	2	21	8		Message user class
		3	1	3	CSP	Send EDI file command (part 2)
EDICMCB	78	3	4	1		Data type
EDICMCB	107	3	5	56		File/ddname
		4	1	3	CSE	Session end command

The network commands service builds the following four commands:

```
----+----1----+----2----+----3----+----4----+----5----+----6
CSSacctnum userid  oldpswd newpswd W05004381    R14
CSP 6B         00164 MSGCLSY   10F
CSPDQDATA
CSE
```

FSUPPORT member

Communications determines the support provided by a network before calling the communications routine for the network. It does this by reading the network commands profile using the key `network id` and network command FSUPPORT. The literal in the command field value (**LITERAL VALUE**) of the member indicates which functions are supported. The first byte corresponds to `FQUEUED`; the second byte corresponds to `FMSGS`, and so on. Values entered in the FSUPPORT member must be in uppercase. Table 210 describes how to use the FSUPPORT bytes.

Table 210. FSUPPORT byte values

Byte name:	A value of Y specifies that:
FQUEUED	Queued functions are supported.
FMSGS	Free-form messages are supported.
FFILE	Free-form files are supported.
FEDIX	ISA/IEA files are supported.
FEDIE	UNB/UNZ files are supported.
FEDIU	BG/EG files are supported.
FEDIG	GS/GE files are supported.
FEDII	ICS/ICE files are supported.
FEDIT	STX/END files are supported.
FCANCEL	CANCEL is supported.
FCLASS	Message class is supported.
FACK	Network acknowledgments are supported.
FSYSMSG	System messages are supported.
FRCVBTP	Receive by trading partner is supported.
FRESTART	Restart is supported.
FNOUSERID	Account number only is entered.
FACCTSEP	The character that is used to separate account number and user ID (blank, period, slash).

For non-BG interchange envelopes: If the value of `FNOUSERID` is **Y**, the full account number is concatenated with the full user ID. If the value is other than **Y**, the following occurs:

- For UNB and STX envelopes, or if more than seven characters were entered in the `Account number` field, all trailing blanks are removed from the account number, the separator defined by the `FACCTSEP` field is concatenated, followed by the user ID. This value is then truncated to the maximum allowed by the standard.

- For ISA and ICS envelopes, where seven or fewer characters were entered in the `Account number` field, seven bytes of the account number will be concatenated with eight bytes of the user ID.

Message handler

The message handler program is identified in the `Message handler` field of the network profile. The message handler is architecturally the same as a user exit and, therefore, has the same language and linkage edit considerations as user exits. For more information, see Chapter 4, "Exit routines," on page 311

The value in the `Message handler` field is the logical name for the exit. The physical load module name and the implementation language for the message handler are defined in the user program information profile (User Exits). An User Exit entry is not needed for the network profile entries that are distributed by WebSphere Data Interchange as these programs are defined in internal WebSphere Data Interchange tables.

The communication routine passes the following parameters to the message handler:

- SNB

  Before calling the message handler, the communication routine moves the name of the message handler from the network profile to the `ZSNBNAME` field in this block.

- CCB

  The message handler returns a value of **12** in the `ZCCBRC` field to indicate that the message handler could not process the network output file. Any other nonzero value returned in `ZCCBRC` indicates the message handler found something wrong.

- FCB

  Before calling the message handler, the communication routine places a value of **1** in the `ZFCBFUNC` field of this block.

- CMCB

  The message handler sets the `FILERCVD` field. The message handler also returns a value in the `NPSERVER` and `NPERRCD` fields.

- CMTPPDB

  The message handler sets the `ACCTNUM` and `USERID` fields.

- NPDB

  The message handler is invoked by the communication routine after the network program has returned control. The message handler processes the responses the network program has written to the network output file as follows:

  – Processes data in the network output file.

  – Calls the status update service to update the status in the Transaction Store, if status update messages are contained in network output file. For more information, see "Update status services" on page 561.

  – Sets the error code in the `NPERRCD` field and the severity code in the `NPSEVER` field of the CMCB if errors are noted in network output file. It also sets the `ZCCBRC` field

in the CCB to indicate that the NPSEVER and NPERRCD fields have meaning. The communication routine uses the NPERRCD and NPSEVER values when logging a VN1015 error message.

Special communications routine for CICS

A special communications routine for the CICS environment (logical name VANICICS) uses a LINK interface to pass control to the network program and the message handler. If you do not plan to use Expedite/CICS to communicate with trading partners, this special communications routine meet your needs. You can provide your own network program and message handler with a CICS interface. VANICICS passes control information to the network program and message handler by way of the CICS COMMAREA. Your program is expected to overwrite the COMMAREA with the resulting information, and VANCICS will log errors if your program determines the execution was unsuccessful. VANICICS also provides the interchange queueing function. Whenever an interchange is generated through an ENVELOPE function, the translator indirectly invokes the associated communication routine to write the interchange to the appropriate TS queue. Management reporting functions are also provided. Whenever an EDI data receive occurs, VANICICS invokes management reporting to update the appropriate statistics.

Network profile definition for CICS

The first task in using VANICICS is to define a new network profile member. For more information on defining a network profile member, refer to the WebSphere Data Interchange User's Guide.

The following Network Profile fields are important and are described below. The other fields are ignored by WebSphere Data Interchange and can be used however you wish. Your programs receive a copy of this profile member. Based on this, your programs can use the information stored in these ignored fields for your own purposes.

NETWORK ID

The name WebSphere Data Interchange knows this network by. Whatever name you pick, it must be used in the associated requestor and trading partner profile members.

NETWORK NAME

A comment field used to briefly describe the network.

COMM ROUTINE

To use VANICICS, it is critical that you enter **VANICICS** in this field to tell WebSphere Data Interchange that whenever a function handled by a communications routine is invoked, VANICICS is given control.

NETWORK PROGRAM

The name of your network program. WebSphere Data Interchange will CICS LINK to this program anytime a communications function is directed to VANICICS. The information sent to your program is described in the next section.

TRANS DATA QUEUE

The default TS queue name where interchanges are written when an

Communications routine for CICS

interchange queueing function is received by VANICICS. Your programs are not invoked during an interchange queueing function, but it is important for your network program to know where the interchanges reside. This name can then be used whenever a send function is directed to your network program.

MSG HANDLER

The name of your message processing program. WebSphere Data Interchange will CICS LINK to this program during the processing of an UPDATE STATUS request. VANICICS does not invoke this program after other network functions, such as send or receive. The usual function of this program is to update the status in the Transaction Store.

Note: For VANICICS, the `MSG HANDLER` field contains the name of a physical load module that WebSphere Data Interchange invokes through CICS LINK. For all other communication routines, this field contains a logical service name. This invocation method was chosen for VANICICS because it is easier to use the CICS LINK interface in CICS.

Network program control information for CICS

The network program you supply receives control information via the CICS COMMAREA. The COMMAREA is made up of a set of 4-byte addresses pointing to control blocks. These control blocks contain all the critical information your program requires to process the request. While most of the communication routines in WebSphere Data Interchange use the network commands profile member, VANICICS does not. The interface to your network program is similar to the z/OS point-to-point communication routine. Control blocks with critical information are passed to the program instead of the command being built in the COMMAREA. This method of passing control blocks gives your program greater flexibility. The format of the COMMAREA your program receives is defined in Table 211.

Table 211. Format of COMMAREA received by the network program

Name:	Offset:	Length:	Type:	Address of control block:
SNBADDR	0	4	Bin	SNB
CCBADDR	4	4	Bin	CCB
FCBADDR	8	4	Bin	FCB
CMCBADDR	12	4	Bin	CMCB
TPPADDR	16	4	Bin	CMTPPDB **Note:** The CMTPPDB block contains data only if the `TPNICKN` keyword specifically refers to a trading partner. Otherwise, the `TPNICKNM` field in this block contains the value **DEFAULTTPNICK**.
NPDBADDR	20	4	Bin	NPDB
REQADDR	24	4	Bin	REQDB

Your network program must handle three main function codes that are passed in the `ZFCBFUNC` field of the FCB. Valid values and the required action are:

211   Send a TS queue containing interchanges. The `FILENAME field of the` CMCB contains the name of the TS queue to send. The REQDB contains information about the mailbox or user ID for which the send is to be done.

232   Receive interchanges into a TS queue. The `FILENAME field of the` CMCB contains the name of the TS queue to receive. The REQDB contains information about the mailbox or user ID for which the receive is to be done.

252   Receive status information for an Update status request. You can use the `CMDOUT` field in the NPDB block as a place to hold the name of a TS or TD queue to write out the status information. The NPDB contains an in-storage copy of the associated network profile member. The REQDB contains information about the mailbox or user ID for which the receive is to be done.

When your network program has completed its processing, it must return control to WebSphere Data Interchange via the CICS RETURN command. WebSphere Data Interchange expects your network program to indicate success or failure back to WebSphere Data Interchange. Your network program does this by overwriting the incoming COMMAREA with meaningful information. The overwritten COMMAREA should be formatted as shown in Table 212.

Table 212. Format of the overwritten COMMAREA

Name:	Offset:	Length:	Type:	Address of control block:
RESPONSE	0	5	Char	Response code
SEVERITY	5	2	Char	Response severity
FILLER	7	21	Char	Unused filler

Successful condition (network)

If your network program has executed successfully, it should set the fields in the COMMAREA to the specified values:

RESPONSE
    HI000
SEVERITY
    00
FILLER
    Blanks

No data received (network)

Your network program issue a receive request but there not be any data available to be received. WebSphere Data Interchange does not consider this an error condition; however, this information should be conveyed back to the original calling program. Your network program should indicate to WebSphere Data Interchange that a receive was issued but no data was returned by setting the fields in the COMMAREA to the specified values>

RESPONSE
    HI000

SEVERITY
> 04

FILLER
> Blanks

Error occurred (network)

Your network program encounter many different errors. You can indicate to WebSphere Data Interchange that an error occurred, get a VN1022 message logged, and get an error extended return code of **1022** returned to the caller by filling in the COMMAREA with these values:

RESPONSE
> Any five character string other than HI000

SEVERITY
> **08** or **12**. This string should indicate a specific error in your network program to aid in problem determination. This response code is included in the VN1022 message.

FILLER
> Blanks

Message handler control information

During the processing of an Update status request, WebSphere Data Interchange CICS LINKs to the message handler. VANICICS does not start this program after other network functions, such as send or receive. The message handler program you supply receives control information through the CICS COMMAREA. The format of the COMMAREA your program receives is defined in Table 213.

Table 213. COMMAREA received by the message handler

Name:	Offset:	Length:	Type:	Address of control block:
SNBADDR	0	4	Bin	SNB
CCBADDR	4	4	Bin	CCB
FCBADDR	8	4	Bin	FCB
CMCBADDR	12	4	Bin	CMCB
NPDBADDR	20	4	Bin	NPDB
TPPADDR	16	4	Bin	CMTPPDB
REQADDR	24	4	Bin	REQDB

The message handler is invoked and will always receive the same function code (**252**). Therefore, the `ZFCBFUNC` field in the FCB is not important. When your message handler gains control, its primary task is to update the status of the Transaction Store. You do this by using the update envelope status API documented in "Update status services" on page 561. To aid you in using this API, the SNB and CCB control blocks passed into your program are already initialized and ready to be used by this API.

When your message handler program has completed its processing, it must using the CICS RETURN command to return control to WebSphere Data Interchange. WebSphere Data Interchange expects your message handler program to indicate

success or failure back to WebSphere Data Interchange. Your message handler program does this by overwriting the incoming COMMAREA with meaningful information. The overwritten COMMAREA should be formatted as defined in Table 214.

Table 214. COMMAREA set by the message handler

Name:	Offset:	Length:	Type:	Description:
RESPONSE	0	5	Char	Response code
SEVERITY	5	2	Char	Response severity
FILLER	7	21	Char	Unused filler

Successful condition (message handler)

If your message handler has executed successfully, it should set the fields in the COMMAREA as follows:

RESPONSE
> HI000

SEVERITY
> 00

FILLER
> Blanks

Error occurred (message handler)

Your message handler encounter many different errors. You can indicate to WebSphere Data Interchange an error occurred, and get a VN1022 message logged, and get an error extended return code of **1022** returned to the caller by filling in the COMMAREA as follows:

RESPONSE
> Any five character string other than HI000. The string that you enter should indicate a specific error in your message handler program to aid in problem determination. This response code is included in the VN1022 message.

SEVERITY
> **08** or **12**. This string should indicate a specific error in your network program to aid in problem determination. This response code is included in the VN1022 message.

FILLER
> Blanks

Continuous receive interface (CICS only)

The continuous receive facility was designed to work with Expedite/CICS and Information Exchange, but it can be used with user-written applications. The intent of this interface is to house WebSphere Data Interchange processing information in the continuous receive profile, away from the receiving application's logic. The user-supplied receive application does not need to be concerned with the WebSphere Data Interchange processing that should be performed. This information is contained in the continuous receive profile member. This interface is available in the CICS environment only.

Continuous receive interface

There are three major differences between continuous receive processing which interacts with Expedite/CICS and processing that interacts with user-written receive applications as follows:

- Only continuous receives associated with Expedite/CICS are started and stopped with CICS transactions EDIR and EDIS, or with PLT programs EDICRTS and EDICRSP. Continuous receives associated with user-written receive applications are not started or stopped through WebSphere Data Interchange. The user-written receive application passes the name of the continuous receive profile member to WebSphere Data Interchange when a continuous receive occurs. Since this is all the control information WebSphere Data Interchange requires to perform the appropriate processing, starting and stopping is not necessary. Any request to start or stop a continuous receive associated with a user-written receive application is ignored by WebSphere Data Interchange.

- The mailbox (requestor) profile ID is optional for continuous receive profile members associated with user-written receive applications. If one is supplied in the profile, WebSphere Data Interchange updates the management reporting receive statistics database. If one is not supplied, management reporting statistics are not updated.

- The mailbox (requestor) profile ID is mandatory for continuous receives associated with Expedite/CICS. The continuous receive profile member ID must be a maximum of 8 characters long. Continuous receive profile member IDs associated with Expedite/CICS can have a maximum length of 16 characters.

Invoking the continuous receive interface

When a user-written receive application receives EDI data from a source (such as a leased line, different VAN, remote TD queue), it can invoke the continuous receive interface to have this EDI data processed. The EDI data must reside in a TS queue for WebSphere Data Interchange to process it. When the data resides in a TS queue, the user application uses the CICS LINK command to give control to program EDICRIN. When the CICS LINK command is issued for program EDICRIN, a COMMAREA must be passed. The COMMAREA must have the format defined in Table 215.

Table 215. COMMAREA format for continuous receive (CICS)

Name:	Offset:	Length:	Type:	Address of control block:
CRMEMBER	0	8	Char	Continuous receive profile member name
RESERVED	8	2	Char	Reserved for future use
ENVFILE1	10	8	Char	TS queue (1st 32-K record)
USRFLD	18	16	Char	User area
RESERVED	34	146	Char	Reserved for future use
ENVFILE2	180	8	Char	TS queue (2nd 32-K record)
ENVFILE3	188	8	Char	TS queue (3rd 32-K record)
ENVFILE4	196	8	Char	TS queue (4th 32-K record)
ENVFILE5	204	8	Char	TS queue (5th 32-K record)
ENVFILE6	212	8	Char	TS queue (6th 32-K record)
RESERVED	220	36	Char	Reserved for future use

When invoking program EDICRIN, the user-written receive application must be concerned with the following fields:

CRMEMBER

The name of the continuous receive profile member that contains the WebSphere Data Interchange processing information. The maximum length of this field is eight characters. The profile member contains information such as whether translation is desired, the name and type of the print file, and other critical information for WebSphere Data Interchange to process the incoming data. For more information on the fields in the profile, refer to the continuous receive profile definition in the WebSphere Data Interchange User's Guide.

ENVFILEx

The name of the TS queues holding the EDI data that WebSphere Data Interchange is to process. The TS queues contain multiple interchanges and processing is based on the value in CRMEMBER. For more information about processing multiple TS queues, see "Processing multiple incoming TS queues" on page 351.

USRFLD

The data in this field is moved to the USRFLD field in the WebSphere Data Interchange Utility control block. This data is then available to response programs started subsequently. For more information, see "WebSphere Data Interchange Utility control information field descriptions" on page 370.

When WebSphere Data Interchange has completed its processing, it returns control to the user-written receive application. WebSphere Data Interchange updates the COMMAREA indicating success or failure. The user-written receive application is informed whether WebSphere Data Interchange accepted responsibility for the data. WebSphere Data Interchange does not return any information as to whether the data was processed successfully. This task is to be performed by the response application supplied in the continuous receive profile. The user-written receive application should be sensitive to the returned COMMAREA. If WebSphere Data Interchange was not able to accept ownership for the data, the user-written response program should be prepared to back up the data. When the problem is fixed, the data can be given back to WebSphere Data Interchange to process. The first seven characters in the returned COMMAREA will contain one of the following values:

HI00000

WebSphere Data Interchange was able to accept ownership of the data. Whether or not WebSphere Data Interchange processing was successful, the continuous receive response application must detect and report any WebSphere Data Interchange processing errors.

HI77712

The continuous receive profile member indicated that translation and deenveloping was not to be performed. The only WebSphere Data Interchange processing performed was the invocation of the continuous receive response program. This program returned a value of **-1** in the severity code (CCBRC) field, indicating that processing was unsuccessful. The data in ENVFILE*x* should be backed up.

Continuous receive interface

HI88812

WebSphere Data Interchange processing abended in the EDIB transaction. The data in ENVFILE*x* should be backed up.

HI99912

The profile member specified in the `CRMEMBER` field could not be located, or the `Active?` flag in the profile is set to **N**. The data in ENVFILE*x* should be backed up.

If one of the errors above is encountered, WebSphere Data Interchange attempts to log an error message to the EXPL TD queue. EXPL can be directed to a destination to which other errors are routed.

Interfacing with SAP

SAP is a client/server application which supports business processes such as sales, materials management, and distribution for mainframes and UNIX.

SAP generates application data in the SAP Intermittent Document (IDOC) layout. The file is sent to the EDI subsystem (or translator) using file transfer products such as FTP or TCP/IP.

You must provide the usage/rules for translating inbound and outbound IDOCs. The mapping required to perform the translation of the inbound and outbound IDOCs is a user responsibility. To assist in mapping the IDOC, a mapping literal keyword &THANDLE for send/receive maps and the new source document property THANDLE is provided to enable mapping of the WDI archive key to the SAP IDOC for inbound processing. The special WDI variable name DISAPSEQ is not supported for data transformation processing. WDI provides the capability to capture the SAP status information during different phases of the EDI process by specifying the SAPUPDT keyword on the utility perform commands.

You can capture SAP status information during different phases of the EDI process by specifying the keyword `SAPUPDT` on PERFORM commands. SAP status tracking is only supported with the WebSphere Data Interchange Utility.

PERFORM commands allow you to extract or remove the SAP status records from the database based on selection criteria, and write them (in SAP EDI_DS record format) to a sequential file for transfer to the SAP system. WebSphere Data Interchange supports the SAP status EDI_DS record at IDOC releases 2, 3, and 4.

Outbound processing and SAP status

You can control the capture of SAP status by setting the `SAPUPDT` keyword to **Y** on Utility PERFORM statements during outbound processing. However, if you specify `SAPUPDT` keyword, the SAP status is stored along with the WebSphere Data Interchange archive key in a new database called EDIVSSTK. The status is then updated at various key points during EDI processing.

To activate SAP status tracking, you must specify the SAPUPDT keyword on the following PERFORM commands:
- ENVELOPE
- ENVELOPE AND SEND
- PERFORM TRANSFORM (using SYNTAX(D))
- REENVELOPE
- REENVELOPE AND SEND
- RESTART SEND
- SEND
- TRANSLATE AND ENVELOPE
- TRANSLATE AND SEND
- TRANSLATE TO STANDARD

Inbound processing and SAP status

You can control the inbound processing of SAP status for functional acknowledgments by setting the SAPUPDT keyword to **Y** on Utility PERFORM statements. When you specify the SAPUPDT keyword and the Transaction Store is active, functional acknowledgment status is collected in the database.

To activate SAP status tracking, you must specify the SAPUPDT keyword on the following PERFORM commands:
- DEENVELOPE
- DEENVELOPE AND TRANSLATE
- PERFORM TRANSFORM (using SYNTAX(E))
- RECEIVE AND DEENVELOPE
- RECEIVE AND TRANSLATE

SAP status codes supported by WebSphere Data Interchange

The SAP status codes supported by WebSphere Data Interchange are:

04	Error within control information of EDI subsystem
05	Error during translation process
06	Translation successful
09	Error during interchange handling
10	Interchange handling successful
11	Error during dispatch
12	Dispatch successful
16	Functional Acknowledgment positive
17	Functional Acknowledgment negative
22	Dispatch successful, acknowledgment still due

You must provide the translation usage/rule required to perform the translation of the inbound and outbound IDOCs. The literal keyword &THANDLE is provided to map the WebSphere Data Interchange archive key to the SAP IDOC for inbound processing, and the WebSphere Data Interchange variable name DISAPSEQ is provided to allow you to save the IDOC record sequence number of the first error during outbound processing. The variable DISAPSEQ is collected in the SAP status record to indicate the first record in error. DISAPSEQ is not available for data transformation processing.

Extracting SAP status records

You can use the SAP STATUS EXTRACT command to extract SAP status records from the WebSphere Data Interchange database, and write them to an output file. The record length for the output file must at least as large as the SAP status EDI_DS record. The error codes returned by this command are:

792   No records match the selection criteria.

796   Records were truncated in the output file.

If the SAPFILE, SAPTYPE keywords are specified on the PERFORM TRANSFORM command, the system will write the SAP status data to the file specified by the SAPFILE/SAPTYPE keywords for CICS or to the file specified by the SAPFILE keyword for other platforms. The system will also write the data to the DB, each record will be marked as extracted. For all other PERFORM commands the OUTFILE and OUTTYPE keywords should be used.

Removing SAP status records

You can use the SAP STATUS REMOVE command to remove SAP status records from the WebSphere Data Interchange database. The only error code returned by this command is:

796   Error occurred during processing.

Interfacing with MQSeries

You can exchange data with your trading partners using MQSeries queues in network profiles. WebSphere Data Interchange prohibits the use of MQSeries queues as the target of the envelope process or as input to the deenvelope process. MQSeries queues can be used as part of a logical network where enveloped data is sent to and received from MQSeries queues. WebSphere Data Interchange provides a sample network profile member named MQSAMP to assist you with the setup of this local network, as described in the following steps:

1. Ask your MQSeries Administrator to define the MQSeries queue you plan on using.

2. Define WebSphere Data Interchange MQSeries Queue profile members for the queues your MQSeries administrator has created. For more information, refer to the WebSphere Data Interchange User's Guide.

3. On the Update Profile Member Panel, complete the fields listed below as follows:

   Communication rtn

   VANIMQ. The name of the communication routine that builds network commands and invokes the network's send and receive program to process the commands.

   Envelope File

   The logical name of the staging file that will hold the enveloped transactions waiting to be sent to your trading partner when you create output messages in WebSphere Data Interchange, but do not instruct WebSphere Data Interchange to send them yet. For example, if you do a **PERFORM TRANSFORM**, but do not specify an output logical name on the rule or **PERFORM** command, then output messages will be written to

the envelope file associated with the network used by the destination trading partner. If you leave this field blank, the default is **QDATA**.

Input rec length

Not used by the WebSphere MQ interface.

Message handler

Not used by the WebSphere MQ interface.

Msg text header

Not used by the WebSphere MQ interface.

Net acks file

Not used by the WebSphere MQ interface.

Net output file

Not used by the WebSphere MQ interface.

Network ID

A unique name to identify the network, such as MQSAMP. This value is referenced by the mailbox (requestor) profile and the trading partner profile. Use the same ID throughout WebSphere Data Interchange to refer to this network.

Network input file

Not used by the WebSphere MQ interface.

Network name

A descriptive name of the network, such as Sample MQSeries Network. This field is optional.

Network parameters

The logical names of the queues used to send and receive messages to a trading partner when WebSphere MQ queues are used to communicate with the trading partner. WebSphere Data Interchange associates each external trading partner with a network. If WebSphere MQ queues are used to communicate with the trading partner, then the trading partners ″network″ consists of two WebSphere MQ queues; a queue used to send to the trading partner (**SENDMQ**) and a queue that the trading partner uses to send messages to you (**RECEIVEMQ**). This field must match the WebSphere Data Interchange MQSeries queue profile member names created in Step 2 above. The **SENDMQ** keyword should be followed by the WebSphere Data Interchange MQSeries queue profile member name where you send data. The **RECEIVEMQ** keyword should be followed by the WebSphere Data Interchange MQSeries queue profile member name where you receive data. Both parameters must be delimited by blanks. If you are performing one way communication with your trading partner, only the keyword (or value) combination of that direction is required.

Network program

EDIMQSR, **EDIRFH2** or **EDICYCL**. The physical name of the send and receive program. This program is invoked by the communication routine to process requests. If messages are to be MQPUT to a queue with only an MQMD header, then specify the **EDIMQSR** network program. If messages are the be MQPUT to a queue with both the MQMD header and an RFH2

header, then specify **EDIRFH2**. WebSphere Data Interchange will fill out the `<mcd>` folder and add values like the sender and receiver ID's to the `<usr>` folder of the RFH2 for each message MQPUT to the queue. **EDIRFH2** should also be used for queues going to the iSoft EDI over the Internet (EDI-INT) Gateway product. If the queue goes to a product using the JMS API, such as the IBM Trading Partner Interchange product, Cyclone Interchange Server, or the IBM Web Services Gateway, then specify **EDICYCL**. This will cause WebSphere Data Interchange to format the RFH2 correctly for these products.

Network sequence
> Not used by the WebSphere MQ interface..

Script name
> Not used by the WebSphere MQ interface.

System level
> This field is not used by any currently supported network.

System type
> Not used by the WebSphere MQ interface.

Time zone
> Not used by the WebSphere MQ interface.

Trans data queue
> The ddname of the file that will hold the enveloped transactions waiting to be sent to your trading partner. This file is also used when queuing or sending transactions. If you leave this field blank, the default is **QDATA**.

Trans rec length
> For distributed platforms and z/OS this field is not used. For the Transaction Server only, **trans rec length**is the length of records in the TD queue. For WebSphere Data Interchange, the maximum usable record length for a TS queue is **28000**. To utilize all 28000 bytes in each record, enter a value of zero or blank in this field. Otherwise, the maximum number that can be entered in this 4 character field is **9999**.

4. Your new network profile member can be used in trading partner profiles and mailbox (requestor) profiles. For more information see Appendix A, "WebSphere Data Interchange control blocks," on page 631.

5. If you are using continuous receive, you also perform the following tasks:
   - When using MQSeries queues with an application file, you must identify the associated MQSeries queue profile member in the data format fields (specifically, the `Application file name` and `Application file type` fields).
   - When executing a translation, you can read data to or write data from MQSeries queues (for print, report, exception, and so on).
   - When using an MQSeries queue for your application data file, you must enter the MQSeries queue profile member name and the MQ type in at least one of the following places:
     – Application file name and file type in the data format
     – Application file name and file type in the Receive Usage panel
     – Any PERFORM command that uses the `APPFILE` and `APPTYPE` keywords

Additional information added to MQRFH2

Additional information has been added to MQRFH2 from the EDIRFH2 network program. The EDIRFH2 network program has been modified to propagate routing information in the USR folder of the MQRFH2 header. For a translation from EDI to another format (for example, XML or DF), the following information would be added to the USR folder:

<ReceiverID>
>   EDI receiver ID from WebSphere Data Interchange

<ReceiverQual>
>   EDI receiver qualifier from WebSphere Data Interchange

<SenderID>
>   EDI sender ID from WebSphere Data Interchange

<SenderQual>
>   EDI sender qualifier from WebSphere Data Interchange

In a DF to DF, the trading partner nickname would be used and the <SenderQual> and <ReceiverQual> tags would be left blank. In a DF there is no WebSphere Data Interchange envelope.

XML special considerations

When processing XML data, there are some additional things that you must consider that are not relevant for other types of source and target documents. These include understanding how WebSphere Data Interchange resolves the DTDs that it uses to validate the XML data, and how it handles different encoding types such as EBCDIC or UTF-8. This section describes these special considerations.

Note: The **XMLEBCDIC** keyword is used only for z/OS systems and is ignored for Windows and AIX.

XML DTD resolution

External DTDs can be parsed along with the XML data. The DTD can be used to validate that the XML data conforms to the DTD, and can also be used to resolve things such as default attribute values and parameter entity references. If you want the parser to process an external DTD along with the XML data, you must copy the DTD file to the server system. On z/OS, the file can be stored as either a PDS member or an HFS file.

If your sever is a z/OS system, there are some special encoding considerations to keep in mind when uploading the DTD files. When you use the DTD to validate a source (input) XML document, upload the DTD as text if you specify `XMLEBCDIC` as **Y** (default). If you specify `XMLEBCDIC` as **N**, the DTD file must contain an XML declaration that includes the appropriate encoding type for the file. For more information, see the section on XML encoding considerations for z/OS. For data transformation maps, WebSphere Data Interchange does not validate output (target) XML documents, and so

does not use DTDs for this case. If the DTD will be used to validate the outbound XML data for send maps, upload the DTD as text.

If DTD processing is to be done as part of the XML parsing (based on the selected validation level), the following processing takes place whenever the parser finds a reference to an external DTD file:

1. When an external DTD reference is found in the XML data, all path information (such as a URL path or file path information) is removed from the DTD name, leaving only a base DTD name.
2. Then the base DTD name is combined with the XMLDTDS value to determine the filename of the DTD file. This DTD file is used by the XML parser in place of the URL specified on the DOCTYPE declaration in the XML data. For z/OS, if the XMLDTDS value does not start with a slash (HFS file), the file is assumed to be a PDS and the first 8 characters of the base name (minus the extension) are assumed to be the member name. If the XMLDTDS value starts with a slash, the value is assumed to be an HFS path and the base name is appended to the path information. For CICS, the XMLDTDS value must be an HFS path, and the base name is appended to the path information. For Windows and AIX, the XMLDTDS value specifies a path, and the base name is appended to the path information.

For example, if the XML data contains the following DOCTYPE declaration:

```
<!DOCTYPE PurchaseOrder SYSTEM "http://xyz.org/xml/dtds/MyPO.dtd">
```

The DTD name is resolved as follows:

1. The URL path information is removed, leaving **MyPO.dtd** as the base name.
2. If the server system is AIX or Windows, or if the XMLDTDS value starts with a slash such as **/u/ediuser/mydtds**, then the base name is appended to the XMLDTDS value making the DTD file name **/u/ediuser/mydtds/MyPO.dtd**.
3. If the server system is z/OS and the XMLDTDS value does not start with a slash such as EDIUSER.MYDTDS, then the base name (without the extension) is assumed to be a PDS member making the DTD file name **EDIUSER.MYDTDS(MYPO)**.

Additional DTD resolution for z/OS

On z/OS this process might cause conflicts where long DTD names, or DTD names that differ only in their extensions are used. To resolve conflicts caused by long DTD names you can use a DTD alias file. The DTD alias file (**DD:DTDALIAS**) allows you to specify an alias for a DTD name. For example, if you are keeping your DTD files in a PDS, but you have two DTD files, **LongDTDName1.dtd** and **LongDTDName2.dtd**, using the first 8 characters would yield the same member name for both (LONGDTDN). The DTDALIAS file allows you to specify different member names for each of these. The format is the DTD (base) name followed by one or more blanks, followed by the alias name. Remember that PDS member names are not case sensitive, but HFS file names are case sensitive. For example:

```
longdtdname1.dtd LONGN1
longdtdname2.dtd LONGN2
```

If you allocate a DTD alias file (**DD:DTDALIAS**), the alias table table is used as follows:

1. The base name is first looked up in the alias file. The search for the base name in the alias file is not case sensitive.
2. If the base name is found in the alias file, the alias name is combined with the XMLDTDS value to determine the filename of the DTD file.
3. If no alias is found or the DTDALIAS file is not allocated, then the base name is used as above.

As an example. if you used the above DTDALIAS file, specified the parameter XMLDTDS(EDISUER.DTDS) on your PERFORM command, and your XML data contained the following DOCTYPE declaration:

```
<!DOCTYPE po SYSTEM "http://xyz.org/xml/dtds/LongDTDName1.dtd">
```

The parser would resolve the DTD as follows:

1. Remove the URL path information from the DTD name, resulting in a base name of LongDTDName1.dtd.
2. Search the DTDALIAS file for this base name. Since this search is not case sensitive, it would find the first entry, resulting in an alias name of LONGN1.
3. Combine the alias name with the XMLDTDS value, instead of using the base name. Since EDIUSER.DTDS does not start with a slash, it is assumed to be a PDS. Therefore, the member EDIUSER.DTDS(LONGN1) will be processed as the DTD file for this XML document.

XML encoding considerations for z/OS

The XML processor uses the XML Toolkit for z/OS and OS/390 to parse the XML data. This parser supports many different character encodings, and follows the XML standards for auto-detection of the character encoding. However, this can sometimes cause problems when dealing with EBCDIC data.

According to the XML standard, if the XML document is not in UTF-8 (similar to ASCII for most commonly used characters) or UTF-16 format, it must begin with an XML encoding declaration (<?xml...>). For EBCDIC data, the **encoding=** attribute must be present and indicate which encoding type is in use. If the XML data was generated as ASCII data and then converted to EBCDIC format, it is likely that the **encoding**= attribute would not be added or updated to reflect the EBCDIC conversion.

Additionally, the new line character (EBCDIC x'15') is not recognized as a valid white space character by the XML standard. However, in z/OS this is often inserted into files as a record separator by editors, upload applications and other z/OS applications.

To resolve these problems, the XML processor in WebSphere Data Interchange can instruct the parser to override the default (auto-detected) encoding type for the XML document with a special encoding type: **ebcdic-xml-us**. This causes the parser to ignore the **encoding=** attribute in the XML declaration and use this special type instead. This encoding type is based on the IBM1140 codepage and will treat any newline characters as a carriage-return character (x'0A'), which is a valid XML white space character.

XML special considerations

When receiving XML data, specifying the XMLEBCDIC keyword as **Y** (which is also the default setting) instructs the XML processor to override the default encoding type with the special ebcdic-xml-us encoding type. This applies to any external DTDs processed, as well as to the XML data itself. This allows the XML processor to handle XML data and DTDs in EBCDIC format that contain new line characters but not contain the correct encoding type in the XML declaration.

When you specify XMLEBCDIC as **N**, the XML processor determines the encoding type for both the data and external DTDs based on the normal XML auto-detection rules. Use this setting if your input XML data is in a format other than EBCDIC, such as ASCII or UTF-16. Since the value of XMLEBCDIC is also used to control the interpretation of the DTD files, your DTDs must contain an XML declaration (<?xml...>) with the proper encoding type.

When generating XML data, the XML data is always created as EBCDIC. No encoding type is specified in the default XML prolog. If you want to send the data in another format, such as ASCII or UTF-16, the translation must be done outside of WebSphere Data Interchange. In many cases, translation can be handled by the transport mechanism (such as FTP) that you use to send the data to another system. If you need to specify an **encoding**= attribute in your XML declaration, you override the default prolog. For data transformation maps, you do this by setting the DIPROLOG special property. For send maps, you use the DIPROLOG mapping variable.

Chapter 9. Data Transformation Transaction Store Implementation

Data Management and reporting for Data Transformation processing uses the existing Transaction Store interfaces. The Transaction Store Facility is supported on the z/OS platform. For data transformation, the facility functions supported includes reporting functions and status update. The WDI Utility interface is supported for all platforms and includes the primary transaction store functions.

The primary Transaction Store functions for EDI messages that are translated using the PERFORM TRANSFORM processing include:

- Deferred or Delayed translation - Deenvelope an inbound EDI transaction with one command, then translate it later. This is similar to the existing receive processing commands PERFORM DEENVELOPE with a separate PERFORM TRANSLATE TO APPLICATION.
- Deferred or Delayed enveloping - Translate a document to create an EDI transaction with one command, then envelope it at a later time. This is similar to the existing send processing commands PERFORM TRANSLATE TO STANDARD with a separate PERFORM ENVELOPE.
- Status Tracking/reporting capability - The existing transaction store interface (WDI Client, z/OS Administrator's Menu and Utility PERFORM commands) will be used to report the status of transactions in the transaction store. Documentation on WDI Transaction Store may be found in WebSphere Data Interchange for z/OS Administration Guide v3.2 and WebSphere Data Interchange Programmer's Reference.
- Duplicate envelope detection - Optional checking to see if inbound EDI envelopes have already been received. If a duplicate exists, appropriate Functional Acknowledgments will be generated. Duplicate envelopes may be reported with the CONTRL Functional Acknowledgment.

Activating Transaction Store

The WDI Client Application Defaults profile identifies your business applications to WebSphere Data Interchange and controls certain processing features. An application defaults profile mainly determines whether copies of transactions are stored in WebSphere Data Interchange's Transaction Store database and the log files in which processing messages are stored.

When an application invokes WebSphere Data Interchange, it can provide an application ID (APPLID). WebSphere Data Interchange then searches for an application defaults profile to match the application ID. The default application defaults profile is EDIFFS. For WDI z/OS, application ID can be provided as an invocation parameter APPLID = value. For AIX/WINDOWS, application ID can be provided with a SET APPLID(value) in the STDIN command file or using the SetAppl(char* pszVal) class with the WDI C++ API.

There are three flags in the application defaults profile that control transaction store activity. These flags may contain the Values: Y(es), N(o), E(rror), S(uccessful). Y(es)

specifies transaction store is active, N(o) specifies transaction store is inactive, E(rror) specifies that only translations with errors are written to the store, and S(uccessful) specifies that only successful translation are written to the store. The following application defaults profile flags control transaction store activity:

- Transaction
- Transaction Image
- Func. Ack Image

The Transaction Store flag must be a value other than N for transaction store to be active. If the Transaction Store flag value is N, the store is inactive and the image flag values are ignored.

The Transaction Image flag specifies if inbound or outbound transaction images are to be written to the store. This includes inbound functional acknowledgment images.

The Functional Ack. Image flag specifies if outbound functional acknowledgment images are to be written to the store. A value of E(rror) specifies the outbound functional acknowledgment images are written to the store only for inbound transactions that fail standard compliance checks. A value of S(uccessful) specifies the outbound functional acknowledgment images are written to the store only for inbound transactions that passed standard compliance validation successfully. This flag does not control transaction store activity for inbound functional acknowledgment images. Inbound functional acknowledgment images are controlled by the Transaction Image flag.

Only transactions that are translated to or from EDI data may be written to the Transaction Store. The EDI images may be written to the store for these transactions. Data Format(DF) or XML images will not be written to the store for these transactions. Other translations that do not involve EDI, such as XML-DF, DF-DF, and XML-XML, will not be written to the Transaction Store.

Appendix A. WebSphere Data Interchange control blocks

This appendix describes WebSphere Data Interchange control blocks, including block layouts and field descriptions. The control blocks aare provided in softcopy format.

For Windows and AIX, only the C header files (.h) are included. These files are in the **include** directory under the main installation directory.

For z/OS and CICS, the control blocks are in the following distribution load libraries.

EDI.V3R2M0.SEDIASM1
>   Assembler and COPY

EDI.V3R2M0.SEDICBL1
>   COBOL and CBLCPY

EDI.V3R2M0.SEDICCC1
>   C and H

EDI.V3R2M0.SEDIPLI1
>   PL/I and INCLUDE

The names and descriptions of the files are listed in Table 216. Use these files as a starting point for copy books that are tailored for use by your installation.

Table 216. Distribution libraries member descriptions

File Name	Description
DIAPI	Defines the C++ API for AIX and Windows
EDICCB	Common Control Block (CCB)
EDICMCB	Communications Control Block (CMCB)
EDIDBLK	Translator Data Block (TRIDB, TRODB, DATABLK)
EDIDEA	Data Extract Application Record
EDIDEE	Data Extract Interchange Record
EDIDEG	Data Extract Group Record
EDIDENTA	Network Activity Record
EDIDER	Data Extract Image Record
EDIDET	Data Extract Transaction Record
EDIDETPA	Transaction Activity Record
EDIDETPC	Trading Partner Capability Record
EDIDETPI	Trading Partner Information Record
EDIFCB	Function Control Block (FCB)
EDIFFC	WebSphere Data Interchange Utility C Record
EDIFFD	WebSphere Data Interchange Utility D Record
EDIFFDU	WebSphere Data Interchange Utility Control Information Block
EDIFFE	WebSphere Data Interchange Utility E Record

Table 216. Distribution libraries member descriptions (continued)

File Name	Description
EDIFFG	WebSphere Data Interchange Utility G Record
EDIFFI	WebSphere Data Interchange Utility I Record
EDIFFQ	WebSphere Data Interchange Utility Q Record
EDIFFT	WebSphere Data Interchange Utility T Record
EDIFFZ	WebSphere Data Interchange Utility Z Record
EDIHBLK	Translator Huge Block (TRIDB, TRODB)
EDINPDB	Network Profile Data Block (NPDB)
EDIRQDB	Mailbox (Requestor) Profile Data Block (REQDB)
EDISNB	Service Name Block (SNB)
EDISPDB	Security Profile Data Block (SPDB)
EDISUDB	Status Update Data Block
EDISUK0	Status Update 210 Key Block
EDISUK1	Status Update 211 Key Block
EDISUK2	Status Update 212 Key Block
EDISUK3	Status Update 213 Key Block
EDISUK4	Status Update 214 Key Block
EDISUK5	Status Update 215 Key Block
EDITPDB	Trading Partner Profile Data Block (TPPDB)
EDITRCB	Translator Control Block (TRCB)
EDIVNMH	VANICICS commarea to message handler (z/OS only)
EDIVNNP	VANICICS commarea to network program (z/OS only)

Service Name Block (SNB)

The SNB allows you to access WebSphere Data Interchange services by associating a logical name with a physical load module name at the time of execution. These logical names and load modules are associated in a static WebSphere Data Interchange table.

The same interface used to access WebSphere Data Interchange API services is used when WebSphere Data Interchange invokes a user-written exit program. User exits are defined and given logical names during the customization process. The User Exits (ADAMCTL) profile is then updated so that the physical load module and the implementation language can be associated with the logical name. User Exit profile entries are automatically added to the service table during execution as user exits are requested.

Using the same SNB each time you request a service improves performance. Once a SNB has been used, the logical and physical association is saved in the ZSNBNDX field, eliminating the table search on the next request. During processing, the SNB is

modified by WebSphere Data Interchange. If you want a reentrant program, do not assign storage for the SNB from static storage.

Table 217 on page 634 describes the layout of the SNB.

Table 217. SNB definition. Layout of the SNB and descriptions of fields

Name	Offset	Length	Type	Description
ZSNBLL	0	2	Bin	SNB length
ZSNBID	2	2	Bin	Reserved
ZSNBEYE	4	8	Char	Dump eye catcher
ZSNBNAME	12	8	Char	Service name
ZSNBNDX	20	4	Bin	Service entry index
ZSNBPC	24	2	Bin	Parameter count
ZSNBFLG0	26	1	Char	Reserved for WebSphere Data Interchange
ZSNBFLG1	27	1	Char	Reserved for WebSphere Data Interchange
ZSNBFANC	28	4	Bin	Reserved for WebSphere Data Interchange

SNB field descriptions

ZSNBLL
A 2 byte binary field that contains the length of the SNB control block. An SNB is 32 bytes.

ZSNBID
Reserved. This field is not currently used.

ZSNBEYE
The first time an SNB is used, WebSphere Data Interchange initializes this field with a value of ****ZSNB****. When you look at virtual storage dumps, this field helps identify the sections containing an SNB.

ZSNBNAME
Indicates the logical name of the WebSphere Data Interchange service being requested. Service names must be left-justified and padded with blanks. WebSphere Data Interchange maintains an internal table that associates a logical name with the physical load module that processes the request. User exits are given logical names at various points during the customization process. The association of a user-defined logical name with a physical load module name is done through the User Exits (ADAMCTL) profile. Valid values are:

ENVSERV
　　　Environmental services

TRANPROC
　　　Translation services

TRANPROC
　　　Enveloping services

TRANPROC
Data extraction services

COMM
Communications services

TRANSSRV
Update status services

SYNCSERV
SYNCPOINT services

ZSNBNDX
Used internally by WebSphere Data Interchange to record the offset into the internal table that defines the services.

ZSNBPC
Indicates the number of parameters that are being provided in the FXXZC, FXXZCBL, FXXZPLI or FXXZASM call. Not all functions in a service require the same number of parameters. Setting the parameter count incorrectly can yield unpredictable results.

ZSNBFLG0
Reserved for WebSphere Data Interchange. First flag byte.

ZSNBFLG1
Reserved for WebSphere Data Interchange. First flag byte.

ZSNBFANC
Reserved for WebSphere Data Interchange. First anchor position.

Common Control Block (CCB)

The CCB is used by WebSphere Data Interchange to maintain status information about the current WebSphere Data Interchange session. A WebSphere Data Interchange session includes anything that happens between an initialize service request and a terminate service request. The CCB used on the initialize request must be used for all requests made during a session. The CCB is provided to all WebSphere Data Interchange programs and all user-defined exit programs that are invoked during the session.

Most of the fields in this control block are for use by WebSphere Data Interchange. However, the return code (ZCCBRC) and extended return code (ZCCBERC) are used to communicate the results of the request to the calling program. The CCB must be at least 608 bytes long for WebSphere Data Interchange to maintain status. However, the length of the CCB can exceed 608 bytes if your application program wants to provide data to a user-defined exit program invoked to process an application request. Anything defined beyond the ZCCBRSV field is for application use only and is not altered by WebSphere Data Interchange. During processing, the CCB is modified by WebSphere Data Interchange. If you want a reentrant program, the storage for the CCB must not come from a static storage area.

Table 218 describes the layout of the CCB.

Table 218. CCB definition. Layout of the CCB and descriptions of fields

Name	Offset	Length	Type	Description
ZCCBLL	0	2	Bin	CCB length
ZCCBID	2	2	Bin	Reserved
ZCCBEYE	4	8	Char	Dump eyecatcher
ZCCBRC	12	4		Return code
ZCCBERC	16	4		Extended return code
ZCCBSID	20	8		System ID
ZCCBUID	28	8		User ID
ZCCBAID	26	8		Application ID
ZCCBCID	44	8		Error module ID
ZCCBXFID	52	2		Function ID
ZCCBLPID	54	6		Language profile ID
ZCCBCPID	68	4		Code page ID
ZCCBRSV	72	4		Reserved
ZCCBCCXP	76	4		Pointer to CCB extension
ZCCBCABP	80	4		Pointer to common area block
ZCCBDBID	84	4		z/OS DB2 subsystem ID
ZCCBDBPL	88	8		z/OS DB2 plan/AIX DB2 alias
ZCCBDBUI	96	8		AIX DB2 user ID
ZCCBDBPW	104	18		AIX DB2 password

Table 218. CCB definition. Layout of the CCB and descriptions of fields (continued)

Name	Offset	Length	Type	Description
ZCCBDRSV1	122	26		Reserved for WebSphere Data Interchange
ZCCBRSV2	148	460		Reserved for WebSphere Data Interchange

CCB field descriptions

ZCCBLL
A 2-byte binary field that contains the length of the CCB control block. A CCB is 608 bytes and that amount of storage must be allocated for this block.

ZCCBID
This field is not currently used.

ZCCBEYE
A label that is useful for identifying this control block when looking at a storage dump. During the WebSphere Data Interchange initialization call, WebSphere Data Interchange initializes this field with the text ****ZCCB****. The calling program is not affected by this field.

ZCCBRC
A signed integer that indicates the success or failure of your last request. The severity of the error is identified in this field and the exact error is identified in the `ZCCBERC` field. Valid values are:

0　　　Successful completion

–nnnn　An error getting to the service you have requested

+nnnn　An error returned from the requested service

ZCCBERC
A signed integer that identifies the error that occurred.

ZCCBSID
WebSphere Data Interchange provides the system ID (a static value of **EDIV00)** during the initialize service API request.

ZCCBUID
The user ID field activated by WebSphere Data Interchange during the initialize service API request. For a TSO user, this is the TSO user ID. For an z/OS batch job, this is the user ID provided in the //JOB card. For a CICS system, this is the CICS signon user ID, the terminal ID, or the application ID of the CICS region.

ZCCBAID

Identifies the Application Default (APPDEFS) profile entry to use for this WebSphere Data Interchange session.The value in this field is moved into the CCB when a service API request is initialized and is a required parameter.

ZCCBCID

Used internally by WebSphere Data Interchange for error logging, but the information it contains can be used in problem determination.

ZCCBXFID

Used internally by WebSphere Data Interchange for error logging, but the information it contains can be used in problem determination.

ZCCBLPID

The language profile ID. During initialization, WebSphere Data Interchange verifies that a Language (LANGPROF) profile entry exists that matches this field. If a LANGPROF entry does not exist or this field is not specified, initialization fails.

ZCCBCPID

This field is not currently used.

ZCCBRSV

Reserved for WebSphere Data Interchange. Do not alter after initialization.

ZCCBCCXP

The pointer to the CCB extension. Do not alter after initialization.

ZCCBCABP

The pointer to the common area block. Do not alter after initialization.

ZCCBDBID

The z/OS DB2 subsystem ID. Do not alter after initialization.

ZCCBDBPL

The z/OS DB2 plan/AIX DB2 alias. Do not alter after initialization.

ZCCBDBUI

The AIX DB2 user ID.

ZCCBDBPW

The AIX DB2 password.

ZCCBRSV1

Reserved for WebSphere Data Interchange. Do not alter after initialization.

ZCCBRSV2

Reserved for WebSphere Data Interchange. Do not alter after initialization.

Function control block (FCB)

The function control block (FCB) identifies the function requested from the service identified in the SNB. Table 219 describes the layout of the FCB.

Table 219. FCB definition. Layout of the FCB and descriptions of fields

Name	Offset	Length	Type	Description
ZFCBLL	0	2	Binary	FCB length
ZFCBFUNC	2	2	Binary	Function code

FCB field descriptions

ZFCBLL

A 2-byte binary field that contains the length of the FCB. An FCB is 4 bytes.

ZFCBFUNC

The function requested for the service defined by the SNB control block. Valid values are:

- **Environmental services (ENVSERV)**
  - **1**   Initialize WebSphere Data Interchange
  - **2**   Terminate WebSphere Data Interchange
  - **5**   Switch APPLID
- **Translation services (TRANPROC)**
  - **131**   Production translate-to-standard
  - **111**   Test translate-to-standard
  - **212**   Production deenvelope and translate-to-application
  - **211**   Test deenvelope and translate-to-application
  - **213**   Translate a specific transaction to application
  - **1000**   End translation
- **Enveloping services (TRANPROC)**
  - **1**   Return interchange header
  - **2**   Return group header
  - **3**   Return transaction header
  - **215**   Envelope transactions
  - **214**   Deenvelope transactions
  - **990**   Close and queue current envelope
  - **991**   Issue COMMIT request
- **Data extraction services (TRANPROC)**
  - **216**   Retrieve detailed data
  - **217**   Retrieve transaction image
  - **218**   Retrieve functional acknowledgment image
  - **219**   Retrieve transaction acknowledgment image
- **Communications services (COMM)**
  - **211**   Send transactions
  - **221**   Send files
  - **232**   Receive
  - **233**   Cancel

252	Process network acknowledgment
300	Return file name
110	Queue transaction data

- **Update Status Services (TRANSSRV)**

210	Update using envelope key (account number, user ID)
211	Update with transaction handle
212	Update using alternate key (account number, user ID)
213	Update using envelope key (qualifier, receiver ID)
214	Update using alternate key (qualifier, receiver ID)
215	Update using envelope key (trading partner nickname)

- **SYNCPOINT services (SYNCSERV)**

1	Initialize SYNCPOINT services
2	Request a COMMIT
3	Request a ROLLBACK

Translator Control Block (TRCB)

The translator control block (TRCB) is the primary control block used to convey information between an application and WebSphere Data Interchange when using translation, enveloping, and data extraction services. For more information about these services, see "Translation services" on page 443, "Enveloping services" on page 500, and "Data extraction services" on page 535

Table 220 describes the layout of the TRCB.

Table 220. TRCB definition. Layout of the translator control block (TRCB) and descriptions of fields

Name	Offset	Length	Type	Description
BLKLEN	0	2	Bin	Block length
RSRVD1	2	2	Bin	Reserved
BLKNME	4	8	Char	Block name
REQID	12	16	Char	Requestor ID
APPFILE	28	8	Char	Application file name
ATFID	36	16	Char	Data format ID
ATSID	52	16	Char	Structure ID
EJECT	68	1	Char	Processing flag
INTPID	69	35	Char	Internal trading partner ID
APPCTLNUM	104	32	Char	Application control value
TRNID	136	16	Char	Standard transaction ID
TEST	152	1	Char	Usage indicator
IHCTL	153	9	Char	Interchange control number
GHCTL	162	9	Char	Group control number
THCTL	171	9	Char	Transaction control number
ENVTYPE	180	1	Char	Envelope type
BLKTYPE	181	1	Char	Data block type
DUPTRAN	182	1	Char	Duplicate flag
ITPBREAK	183	1	Char	New interchange on INTPID change
REQSIZE	184	4	Bin	Required size
XPANDED	188	1	Char	Expanded flag
NEWENV	189	1	Char	New interchange flag
NEWGRP	190	1	Char	New group flag
NEWTRN	191	1	Char	New transaction flag
QSIZE	192	4	Bin	Interchange size
ESIZE	196	4	Bin	Number of bytes processed of this interchange
GRPNUM	200	4	Bin	Number of groups processed so far

Table 220. TRCB definition. Layout of the translator control block (TRCB) and descriptions of fields (continued)

Name	Offset	Length	Type	Description
TRNNUM	204	4	Bin	Number of transactions in processed the interchange
SEGNUM	208	4	Bin	Number of segments in processed in the interchange
TRNGRP	212	4	Bin	Number of transactions processed in the group
SEGTRN	216	4	Bin	Segment count for transaction
ERRNUM	220	4	Bin	Error counter
QBT	224	8	Char	Interchange size
IHXCTL	232	14	Char	Interchange control number
ISYNTAXID	246	4	Char	Interchange syntax ID
ISYNTAXVER	250	1	Char	Interchange syntax version
ISIDQUAL	251	4	Char	Interchange sender ID qualifier
ISID	255	35	Char	Interchange sender ID
ISENDNAME	290	14	Char	Interchange sender name or application sender code
IREVROUT	304	14	Char	Interchange reverse routing
IRIDQUAL	318	4	Char	Interchange receiver ID qualifier
IRID	322	35	Char	Interchange receiver ID
IRECVNAME	357	14	Char	Receiver name
IROUTEADDR	371	14	Char	Routing address
IDATE	385	6	Char	Interchange date
ITIME	391	6	Char	Interchange time
IVERREL	397	5	Char	Interchange version/release
IGT	402	6	Char	Interchange group total
ITT	408	6	Char	Interchange transaction total
IST	414	10	Char	Interchange segment total
IBT	424	8	Char	Interchange byte total
ISPW	432	14	Char	Interchange password
IAPREF	446	14	Char	Application reference
ISTDID	460	4	Char	Interchange standard ID
FASPEC	464	1	Char	Indicates that **FUNACKFLE** is being used.
IPRIOR	465	1	Char	Delivery priority
ICOMMAGREE	466	35	Char	Communication agree
GHXCTL	501	14	Char	Group control number
GFGID	515	6	Char	Group functional group ID

Table 220. TRCB definition. Layout of the translator control block (TRCB) and descriptions of fields (continued)

Name	Offset	Length	Type	Description
GSIDQUAL	521	4	Char	Group sender qualifier
GSID	525	35	Char	Group application sender ID
GRIDQUAL	560	4	Char	Group receiver qualifier
GRID	564	35	Char	Group application receiver ID
GDATE	599	6	Char	Group date
GTIME	605	6	Char	Group time
GVER	611	12	Char	Group version
GREL	623	12	Char	Group release
GTT	635	6	Char	Group transaction total
GAPW	641	14	Char	Group password
GRESPAGENCY	655	2	Char	Group responsible agency
RSRVD3	657	12	Char	Reserved
THXCTL	669	14	Char	Transaction control number
TTC	683	6	Char	Transaction ID
TVER	689	6	Char	Transaction version
TREL	695	6	Char	Transaction release
TST	701	10	Char	Transaction segment total
LASTINENV	711	1	Char	Last transaction in interchange
XACFIELD	712	35	Char	Application control number
FABUILT	747	1	Char	Functional acknowledgment built
QGTNUM	748	4	Bin	Total number of groups in the interchange
QTTNUM	752	4	Bin	Total number of transactions in the interchange
QSTNUM	756	4	Bin	Total number of segments in the interchange
QGT	760	6	Char	Total number of groups in the interchange
QTT	766	6	Char	Total number of transactions in the interchange
QST	772	10	Char	Total number of segments in the interchange
FASPM	782	8	Char	Functional acknowledgment standard profile member
ENVCHK	790	1	Char	Envelope status check
TRNSTAT	791	1	Char	RAWDATA transaction status
APTYPE	792	2	Char	Application file type
TSKEY	794	10	Packed	Packed transaction handle

Table 220. TRCB definition. Layout of the translator control block (TRCB) and descriptions of fields (continued)

Name	Offset	Length	Type	Description
TSKEYU	804	20	Char	Unpacked transaction handle
MAPKEY	824	16	Char	Map ID
BATCHID	840	8	Char	Batch ID
ENVLDATE	848	8	Char	Earliest envelope date
TRXLIFE	856	2	Bin	Transaction life span
IMGLIFE	858	2	Bin	Transaction image life span
HOLDFLAG	860	1	Char	Hold flag
BNDLFLAG	861	1	Char	Bundle flag
RAWDATA	862	1	Char	RAWDATA flag
ENVLDELAY	863	1	Char	Delayed enveloping flag
TRXACCEPT	864	1	Char	Transaction acceptable flag
TRABORT	865	1	Char	Translator abort flag
FILEID	866	8	Char	User-defined file ID
DSNAME	874	56	Char	Physical data set name
QNETID	930	8	Char	Network ID
QPTTOPT	938	1	Char	Point-to-point network flag
QSRPGM	939	1	Char	Send/Receive program flag
QDDNAME	940	8	Char	ddname for transaction file
FUNACKFLE	948	8	Char	ddname for functional acknowledgment file
QTPNICK	956	16	Char	Trading partner nickname
QRC	972	2	Bin	Queuing return code
QERC	974	2	Bin	Queuing extended return code
ERRCDES	976	20	Bin	First 10 error codes
INMEMTRANS	996	2	Bin	In-storage transactions
NOCOMMIT	998	1	Char	Commit flag
SCOPE	999	1	Char	Recovery scope
TPNICK	1000	16	Char	Trading partner nickname
CONCATENATE	1016	1	Char	Concatenation flag
ASSERTLVL	1017	1	Char	Assertion level
RAWDATAOUT	1018	1	Char	RAWDATA wanted for output
FIXEDTRX	1019	1	Char	Fixed-to-fixed transaction flag
MRREQID	1020	16	Char	Management reporting requestor ID
ERRFILTER	1036	80	Char	Initial error filter
FFILEID	1116	8	Char	File ID for fixed translations

Table 220. TRCB definition. Layout of the translator control block (TRCB) and descriptions of fields (continued)

Name	Offset	Length	Type	Description
VARTRACE	1124	1	Char	Variable trace wanted flag
EXPTRACE	1125	1	Char	Expression trace wanted flag
SSEGVAL	1126	1	Char	Service segment validation flag
TRXFACODE	1127	1	Char	Functional acknowledgment code generated for the transaction
IUSEREXIT	1128	8	Char	User exit for envelopes
IUSERAREA	1136	4	Bin	User area for IUSEREXIT
IUSERACCESS	1140	1	Char	User exit access type
IUSERTYPE	1141	2	Char	User exit program type
MAPCHAIN	1143	1	Char	Mapchain active flag
FORCETEST	1144	1	Char	Force test receive translation
ENVPRBRK	1145	1	Char	Envelope profile name change at ISA break
SUSBLKF	1146	1	Char	CICS suspend block flag
CLRERRS	1147	1	Char	Clear ERRCDES array flag
FARC	1148	4	Bin	Functional acknowledgment return code
FAERC	1152	4	Bin	Functional acknowledgment extended return code
SAPUPDT	1156	1	Char	SAP updates requested
SAPTRX	1157	1	Char	SAP transaction indicator
SAPCLIENT	1158	3	Char	SAP client
SAPDOCNUM	1161	16	Char	SAP document number
SAPSEQ	1177	6	Char	SAP error record sequence number
ROUTCODE	1183	3	Char	Generic send usage routing code
FAEREQ	1186	1	Char	Functional acknowledgment envelope file required
BOUNDARY	1187	1	Char	Incremental translation flag
SUSBLKP	1188	4	Bin	CICS SUSPEND block pointer
CUSERDATA	1192	256	Char	User data area
APPLTPID	1448	15	Char	Application trading partner ID
EXTENDC	1464	1	Char	Extend the C record flag
VAXFLAG	1465	1	Char	Pageable translation flag
GDATE8	1466	8	Char	Group envelope 8-byte data
RECOVBAD	1474	1	Char	Recover from bad EDI standard data
RSRVD4	1475	61	Char	Reserved for WebSphere Data Interchange

TRCB field descriptions

BLKLEN
A 2-byte binary field that contains the length of the TRCB control block. A TRCB is 1536 bytes.

RSRVD1
Reserved.

BLKNME
EDITRCB. The name of the TRCB.

REQID
The requestor ID for a member in the mailbox (requestor) profile. This field might be required for the translating-to-file and deenveloping requests, because the `Receive file name` field in the mailbox (requestor) profile contains the ddname for the file to be processed.

If a value is specified for `FILEID`, this field is not required.

APPFILE
The name of the file where application records should be written. This field is returned by the translator during operations that translate EDI standard data to an application format. The value is taken from the data format definition unless a value is supplied in the trading partner receive usage record.

In z/OS, this field contains a ddname for a file. In CICS, the storage mechanism represented by this field is indicated in the `Aptype` field.

ATFID
The data format ID. This is a required input field for any operation that translates application data to an EDI standard format, because it is part of the primary key used to determine the translation usages/rules.

This field is an output field for operations that translate EDI standard data to an application format and for the enveloping function.

ATSID
The name of the structure that describes the format of the application data. Unless raw data is being processed, this is a required input field for functions that translate application data to an EDI standard format.

This field is an output field for functions that translate EDI standard data to an application format.

EJECT

Indicates when all data associated with a transaction has been provided, and when an interchange has been written to the file associated with the network. Used by almost all functions, as both an input and an output field. See the descriptions of the different functions for the specific allowable values for this field.

INTPID

The internal trading partner ID associated with a transaction. This field might contain a customer ID, vendor ID or other identifier known to the application.

This field is a required input field for functions that translate application data to an EDI standard format because it is part of the key used to locate the translation usage/rules.

This field is not required for input if raw data is being processed, but becomes the output field. It is also the output field for those functions that translate EDI standard data to an application format and for the enveloping function.

APPCTLNUM

A 32-byte version of XACFIELD.

TRNID

The ID of the EDI standard transaction or message that is being translated, enveloped, and deenveloped. This is always a returned field.

TEST

Indicates whether the transaction requires a test usage/rule. This is a required input field for functions that translate application data to an EDI standard format because it is part of the key used for locating the translation usages/rules. Valid values are:

I   Information transaction. An information usage/rule should be used if one is found. If an information usage/rule is not found, a production usage/rule is used instead. Even when a production usage/rule is used, the transaction is flagged as an information transaction.

P   Production transaction. Only a production usage/rule should be used (default).

T   Test transaction. A test usage/rule should be used if one is found. If a test usage/rule is not found, a production usage/rule is used instead. Even when a production usage/rule is used, the transaction is flagged as a test transaction.

U   The translator should determine if the transaction is test, information, or production based on the usage/rule found. If a test usage/rule is found, the transaction is a test transaction, and a value of **T** is returned. If an information usage/rule is found, the transaction is an information transaction, and a value of **I** is returned. If only a production usage/rule is found, the transaction is a production transaction, and a value of **P** is returned.

This field is an output field when translating EDI standard data to an application format, and for deenveloping or enveloping.

IHCTL
A 9-character version of IHXCTL.

GHCTL
A 9-character version of GHXCTL.

THCTL
A 9-character version of THXCTL.

ENVTYPE
The type of interchange enveloping used. This is a returned value for translating, enveloping, and deenveloping functions. Valid values are:

E   UNB/UNZ
I   ICS/ICE
T   STX/END
U   BG/EG
X   ISA/IEA

BLKTYPE
Indicates whether 2- or 4-byte length data blocks are being used for the TRIDB and TRODB structures. The formats for TRIDB and TRODB are always the same. Valid values are:

H   4-byte length blocks
(other) 2-byte length blocks

DUPTRAN
Indicates whether duplicate interchanges are considered errors. This is usually an output field for those functions that translate data from EDI standard format to application format or deenvelope functions. This is an input field for the translate file and the deenvelope function. Valid values are:

Y or (other)
        Part of a duplicate interchange; duplicates are not errors.
N   Not part of a duplicate interchange; duplicates are errors.

ITPBREAK
Indicates whether a change in value of the internal trading partner (`INTPID` field) should create a new interchange. Applies only to envelope processing. Valid values are:

Y   Starts a new interchange each time the INTPID value changes
(other) Does not cause an interchange break, unless the change in internal trading partner results in a new trading partner nickname

REQSIZE
The size of the structure being returned in the TRODB. Applies only when EDI standard data is being translated to an application format and the TRODB is not large enough to contain the entire structure. This is provided for informational purposes only because all the data can be retrieved as partial structures.

XPANDED

Indicates whether a post-Release 1 API program is executing. Set this field to **Y**.

NEWENV

Indicates whether this transaction is the first transaction in an interchange. This is a returned field for all except the translate-specific functions (function code **213**). Valid values are:

Y   First transaction of an interchange
(other) Not the first transaction of an interchange

NEWGRP

Indicates whether the current transaction is the first transaction of a group. This is a returned field for all except the translate-specific functions (function code **213**). Valid values are:

Y   First transaction of a group
(other) Not the first transaction of a group

NEWTRN

Indicates whether a new transaction was started. This is a returned field for all translating, enveloping, and deenveloping functions. Valid values are:

Y   New transaction
(other) Not a new transaction

QSIZE

When translation to a EDI standard or enveloping, the total number of bytes in the interchange queued (`EJECT` value of **Q**). When received, the total number of bytes in the interchange (`NEWENV` value of **Y** during translating to a EDI standard or deenveloping). See `QBT` for a character representation of this field.

ESIZE

If an interchange is being processed, this field represents the amount of the interchange processed so far. See `IBT` for a character representation of this field.

For send operations (translate-to-standard), this field represents the size of the interchange.

For receive operations (translate-to-application), this field represents the number of bytes from the interchange that have been processed.

GRPNUM

If an interchange is being processed, this field represents the number of groups in the interchange processed so far. See `IGT` for a character representation of this field.

For send operations (translate-to-standard), this field represents the number of groups in the interchange.

For receive operations (translate-to-application), this field represents the number of groups from the interchange that have been processed.

TRNNUM

If an interchange is being processed, this field represents the number of transactions in the interchange processed so far. See `ITT` for a character representation of this field.

For send operations (translate-to-standard), this field represents the number of transactions in the interchange.

For receive operations (translate-to-application), this field represents the number of transactions from the interchange that have been processed.

SEGNUM

If an interchange is being processed, this field represents the number of segments in the interchange processed so far. See `IST` for a character representation of this field.

For send operations (translate-to-standard), this field represents the number of segments in the interchange.

For receive operations (translate-to-application), this field represents the number of segments from the interchange that have been processed.

TRNGRP

If an interchange is being processed, this field represents the number of transactions in the group processed so far. See `GGT` for a character representation of this field.

For send operations (translate-to-standard), this field represents the number of transactions in the group.

For receive operations (translate-to-application), this field represents the number of transactions from the group that have been processed.

SEGTRN

The total number of segments in the current transaction. See `TST` for a character representation of this field.

ERRNUM

The total number of errors in the current transaction.

QBT

The character representation of `QSIZE`.

IHXCTL

If an interchange with a **CN** data type is being processed, this field contains the interchange control number extracted from the interchange header.

ISYNTAXID

If an EDIFACT or UN/TDI interchange is being processed, this field contains the interchange syntax identifier extracted from the interchange header field UNB01 or

STX01. For enveloping functions, this field is also an input field that identifies the syntax to be used when building the interchange header segment.

ISYNTAXVER

If an EDIFACT or UN/TDI interchange is being processed, this field contains the interchange syntax version extracted from the interchange header field UNB01 or STX01. For enveloping functions, this field is also an input field that identifies the syntax identifier to be used when building the interchange header segment.

ISIDQUAL

If an EDIFACT, ICS, or ISA interchange is being processed, this field contains the interchange sender ID qualifier extracted from the interchange header field UNB04, ICS04, or ISA05.

For enveloping functions, this field is also an input field that identifies the sender ID qualifier to be used when building the interchange header segment.

ISID

If an interchange with an **IS** data type is being processed, this field contains the interchange sender ID extracted from the interchange header.

For enveloping functions, this field is also an input field that identifies the interchange sender ID to be used when building the interchange header segment. If this value differs from the current interchange sender ID value, a new interchange is created.

ISENDNAME

If a UN/TDI interchange is being processed, this field contains the interchange sender name extracted from the interchange header field STX04. For enveloping functions, this field is also an input field that identifies the sender name to be used when building the interchange header segment.

If a UCS interchange is being processed, this field contains the application sender code extracted from the interchange header field UCS03 if this field does not contain an **IS** data type. For enveloping functions, this field is also an input field that identifies the application sender code to be used when building the interchange header segment if this field does not contain an **IS** data type.

IREVROUT

If an EDIFACT interchange is being processed, this field contains the interchange reverse routing extracted from the interchange header field UNB05.

For enveloping functions, this field is also an input field that identifies the reverse routing to be used when building the interchange header segment.

IRIDQUAL

If an EDIFACT, ICS, or ISA interchange is being processed, this field contains the interchange receiver ID qualifier extracted from the interchange header field UNB04, ICS04, or ISA05.

For enveloping functions, this field is also an input field that identifies the receiver ID qualifier to be used when building the interchange header segment.

IRID
If an interchange with an **IR** data type is being processed, this field contains the interchange receiver ID extracted from the interchange header.

For enveloping functions, this field is also an input field that identifies the interchange receiver ID to be used when building the interchange header segment. If this value differs from the current interchange receiver ID value, a new interchange is created.

IRECVNAME
If a UN/TDI interchange is being processed, this field contains the interchange receiver name extracted from the interchange header field STX06. For UN/TDI enveloping functions, this field is also an input field that identifies the receiver name to be used when building the interchange header segment.

If a UCS interchange is being processed, this field contains the application receiver code extracted from the interchange header field UCS04 if this field does not contain an **IR** data type. For UCS enveloping functions, this field is also an input field that identifies the application receiver code to be used when building the interchange header segment if this field does not contain an **IS** data type.

IROUTEADDR
If an EDIFACT interchange is being processed, this field contains the interchange routing address extracted from the interchange header field UNB08.

For enveloping functions, this field is also an input field that identifies the routing address to be used when building the interchange header segment.

IDATE
If an interchange with a **DT** data type is being processed, this field contains the interchange date extracted from an interchange header.

ITIME
If an interchange with a **TM** data type is being processed, this field contains the interchange time extracted from an interchange header.

IVERREL
If an interchange with a **VR** or **LV** data type is being processed, this field contains the interchange version or release extracted from the interchange header.

IGT
The character representation of `GRPNUM`.

ITT
The character representation of `TRNNUM`.

IST
The character representation of SEGNUM.

IBT
The character representation of ESIZE.

ISPW
If an interchange with a **PW** data type is being processed, this field contains the interchange password extracted from the interchange header.

For enveloping functions, this field is also an input field that identifies the interchange password to be used when building the interchange header segment. If this value differs from the current interchange password value, a new interchange is created.

IAPREF
If an interchange with an **AP** data type is being processed, this field contains the interchange application reference extracted from the interchange header.

For enveloping functions, this field is also an input field that identifies the interchange application reference to be used when building the interchange header segment. If this value differs from the current interchange application reference value, a new interchange is created.

ISTDID
If an X12 or ICS interchange is being processed, this field contains the interchange standard ID extracted from the interchange header field ISA11 or ICS02.

For enveloping functions, this field is also an input field that identifies the EDI standard ID to be used when building the interchange header segment.

FASPEC
Indicates that **FUNACKFLE** is being used.

IPRIOR
If an EDIFACT or UN/TDI interchange is being processed, this field contains the interchange processing priority extracted from the interchange header field UNB15 or STX12.

For enveloping functions, this field is also an input field that identifies the processing priority to be used when building the interchange header segment.

ICOMMAGREE
If an EDIFACT interchange is being processed, this field contains the interchange communication agreement extracted from the interchange header field UNB17.

For enveloping functions, this field is also an input field that identifies the processing priority to be used when building the interchange header segment.

GHXCTL

If a group with a **CN** data type is being processed, this field contains the group control number extracted from the group header.

GFGID

If a group is being processed, this field contains the functional group ID value associated with the group.

GSIDQUAL

If an EDIFACT group is being processed, this field contains the group sender ID qualifier extracted from the group header field UNG03.

For enveloping functions, this field is also an input field that identifies the sender ID qualifier to be used when building the group header segment.

GSID

If a group with an **AS** data type is being processed, this field contains the group application sender ID extracted from the group header.

For enveloping functions, this field is also an input field that identifies the group application sender ID to be used when building the group header segment. If this value differs from the current group application sender ID value, a new group is created.

GRIDQUAL

If an EDIFACT group is being processed, this field contains the group receiver ID qualifier extracted from the group header field UNG05.

For enveloping functions, this field is also an input field that identifies the receiver ID qualifier to be used when building the group header segment.

GRID

If a group with an **AR** data type is being processed, this field contains the group application receiver ID extracted from the group header.

For enveloping functions, this field is also an input field that identifies the group application receiver ID that should be used when building the group header segment. If this value differs from the current group application receiver ID value, a new group is created.

GDATE

If a group with a **DT** data type is being processed, this field contains the group date extracted from the group header.

GTIME

If a group with a **TM** data type is being processed, this field contains the group time extracted from the group header.

GVER

If a group with a **VR** data type is being processed, this field contains the group version extracted from the group header.

For enveloping functions, this field is also an input field that identifies the group version that should be used when building the group header segment. If this value differs from the current version value, a new group is created.

GREL

If a group with an **LV** data type is being processed, this field contains the group release extracted from the group header.

For enveloping functions, this field is also an input field that identifies the group release that should be used when building the group header segment. If this value differs from the current group release value, a new group is created.

GTT

The character representation of `TRNGRP`.

GAPW

If a group with a **PW** data type is being processed, this field contains the group application password extracted from the group header.

For enveloping functions, this field is also an input field that identifies the group application password that should be used when building the group header segment. If this value differs from the current group application password value, a new group is created.

GRESPAGENCY

If an EDIFACT group is being processed, this field contains the group controlling agency extracted from the group header field UNG09. For enveloping functions, this field is also an input field that identifies the controlling agency to be used when building the group header segment.

If an ICS, UCS, or X12 group is being processed, this field contains the group responsible agency extracted from the group header field GS07. For enveloping functions, this field is also an input field that identifies the responsible agency name to be used when building the group header segment.

RSRVD3

Reserved.

THXCTL

If a transaction with a **CN** data type is being processed, this field contains the transaction control number extracted from the transaction header.

TTC

If a transaction with a **TC** data type is being processed, this field contains the transaction or message ID extracted from the transaction header.

TVER

If a transaction with a **VR** data type is being processed, this field contains the transaction version extracted from the transaction header.

For enveloping functions, this field is also an input field that identifies the transaction version that should be used when building the transaction header segment.

TREL

If a transaction with an **LV** data type is being processed, this field contains the transaction release extracted from the transaction header.

For enveloping functions, this field is also an input field that identifies the transaction release that should be used when building the transaction header segment.

TST

The character representation of SEGNUM.

LASTINENV

Indicates whether this transaction is the last transaction of the interchange. Valid values are:

Y   Last transaction of the interchange
(other) Not the last transaction of the interchange

XACFIELD

The application control number that uniquely identifies the transaction to the application. This is a returned value for all translation and enveloping functions. Which field contains the application control number is indicated in the data format with an **AC** data type. This field can be overridden by providing up to eight fields that can be concatenated to form an application control number in the map ID.

FABUILT

Indicates the envelope type, if any, for the functional acknowledgment. You can control when functional acknowledgments are deenveloped. For more information, see the ENVLDELAY field on 661. Valid values are:

E   UNB/UNZ
G   GS/GE
I   ICS/ICE
S   Stored, but not enveloped
T   STX/END
U   BG/EG
X   ISA/IEA

QGTNUM

When queued, the total number of groups in the interchange (`EJECT` value of **Q** during translating to an EDI standard or during enveloping). When received, the total number of groups in the interchange (`NEWENV` value of **Y** during translating to a EDI standard or during deenveloping). See `QGT` for a character representation of this field.

QTTNUM

When queued, the total number of transactions in the interchange (`EJECT` value of **Q** during translating to an EDI standard or during enveloping). When received, the total number of transactions in the interchange (`NEWENV` value of **Y** during translating to an EDI standard or during deenveloping). See `QTT` for a character representation of this field.

QSTNUM

When queued, the total number of segments in the interchange (`EJECT` value of **Q** during translating to an EDI standard or during enveloping). When received, the total number of segments in the interchange (`NEWENV` value of **Y** during translating to a EDI standard or during deenveloping). See `QST` for a character representation of this field.

QGT

The character representation of `QGTNUM`.

QTT

The character representation of `QTTNUM`.

QST

The character representation of `QSTNUM`.

FASPM

Used only by the translator. The standard profile member used in building the service segments for the functional acknowledgment.

ENVCHK

Indicates that the translator should verify a transaction's status during an envelope function for the intent of the envelope operation. Valid values are:

1   Envelopes transactions, and rejects this transaction if it has been enveloped before.
2   Reenvelopes transactions, and rejects this transaction if it has not been enveloped before.
(other) Does not check the status of the transaction.

This field becomes important if multiple jobs can be enveloped concurrently and you want to ensure that a transaction is not enveloped and sent twice.

TRNSTAT

Indicates the status of the transaction. Your application can use this field in conjunction with the `RAWDATA` field to determine the status of a raw data transaction or to determine the next action expected from the application. Valid values are:

I The data format has defined a structure that starts a transaction but the structure has not been received. Data is ignored until the starting transaction structure is recognized.

S The starting transaction structure has been received. If a starting structure is not defined, this value is set when the first structure is received.

C A transaction is in progress and the structure passed was neither the starting nor the ending structure.

R A transaction is in progress and a structure that defines the start of a transaction was received. The data is ignored. The application must first call the translator with an `EJECT` value of **Y** to process the current transaction, and then call the translator again with the same data to start a new transaction

Y A transaction is in progress and a structure recognized as the ending structure is received. The transaction is ready to be processed and a call to the translator is required with an `EJECT` value of **Y**.

APTYPE

The type of application file identified in the APPFILE field. Applies only to CICS. Valid values are:

PG Program
TD TD queue identified by the first 4 characters of `APPFILE`
TM TS queue (in main storage)
TS TS queue (in auxiliary storage) identified by `APPFILE`
TX Transaction code identified by the first 4 characters of `APPFILE`

TSKEY

The Transaction Store handle value for a transaction in packed format. This is an input field for the translate-specific and envelope functions.

This field indicates the transaction to be translated or enveloped. For all other functions, this field is an output field that indicates the transaction key value just processed.

If handling the key in a packed format is a problem for your program, `TSKEYU` contains the key value in an unpacked format. If TSKEY contains all blanks or binary zeros, the value in the TSKEYU field is used.

TSKEYU

The Transaction Store handle value for a transaction in unpacked format. If the `TSKEY` field contains blanks or binary zeros, this field becomes the input field for the translate-specific and envelope functions.

This field indicates the transaction to be translated or enveloped. This field is a returned value for all translation, enveloping, and deenveloping functions. If this field is used as an input field, its packed value is returned in the `TSKEY field`.

MAPKEY

A returned value that identifies the map used to translate the data.

BATCHID

An input field used for all translation functions. This field is an indexed field in the Transaction Store database and provides an efficient method for retrieving transactions from the store. It can also be used to identify all the transactions that were processed during a particular execution of a job. If you do not specify a value, the system date and time are used to create a default value of `DDHHMMSS`.

ENVLDATE

The earliest date that a transaction is eligible to be enveloped and sent. If you do not specify a date, the transaction is eligible immediately, unless other conditions indicate that it is not eligible (such as a transaction in held status). This is an input field for translating application data to an EDI standard format only.

TRXLIFE

The number of days the transaction should remain in the Transaction Store before it is eligible to be purged. If this field contains zero, a default value of 30 days is used. The default value is returned in this field. This is an input field for all translating and deenveloping functions.

IMGLIFE

The number of days the transaction image should remain in the Transaction Store before it is eligible to be purged. If this field contains zero, a default value of 30 days is used. The default value is returned in the this field. This is an input field for all translating and deenveloping functions.

A transaction's life span (`TRXLIFE`) and the transaction's image life span (this field) are always the same. However, you should still set a value for this field to ensure correct processing.

HOLDFLAG

Indicates whether a transaction should be held. This is an input field for functions that add transactions to the Transaction Store. Valid values are:

Y   Holds the transaction
(other)   Does not hold the transaction

BNDLFLAG

Specifies the bundle status for transaction. This is an input field for translating application data to an EDI standard format. If the current trading partner is not using functional groups (the `Functional group` field in the trading partner profile contains **N**), changes in data that would generally cause a new group to be created are ignored, and the bundles are not terminated. Valid values are:

Y   The start of a bundle. This transaction becomes the controlling transaction for the bundle. All transactions that follow become part of the bundle until:
  • Another transaction ends the current bundle or starts a new bundle.
  • The input data forces a new group or envelope.
  • The translation process ends.
N   The last transaction in the bundle.
(blank)   Continue as before.

RAWDATA
Indicates whether raw data processing is being used.

For translating application data to an EDI standard format, valid values are:
Y   Uses the RAWDATA interface.
(other) Uses the C and D record type interface. Application knows the format of the data and is communicating the type of data identified in the `ATSID` field.

Y   Requests raw data processing. If raw data specifications were provided in the data format ID, the translator automatically moves in the record ID values, and the internal trading partner ID is moved to the specified field. The `RAWDATA` field is returned with a value of **Y**.

        If raw data specifications have not been supplied in the data format, the `RAWDATA` field is set to **N** and raw data processing does not occur. The data format ID is returned in the `ATFID` field.

(other) Does not request raw data processing.

For translating EDI standard data to an application format, valid values are:

Note: If RAWDATA processing has been requested and was possible, WebSphere Data Interchange writes the data to the application file in raw data format. However, if RAWDATA processing was not requested or was not possible, C and D records are written to the application file. Your application program can write the records in any desired format, regardless of the setting of the `RAWDATA` field.

ENVLDELAY
Indicates whether enveloping should be delayed for transactions being translated. In any case, the transaction (or functional acknowledgment) and its image are saved in the Transaction Store. The value set for this field on the first call of a session is used for the entire session.

For translating EDI standard data to an application format, valid values are:
Y   Does not envelope the transaction at the same time it is translated.
(other) Envelopes the transaction at the same time it is translated

For the deenveloping and translating a file, valid values are:
Y   Does not envelope functional acknowledgments at the same time it is translated.
(other) Envelopes the functional acknowledgments at the same time it is translated.

For related information about functional acknowledgments, see the `FUNACKFLE` field description on 662.

TRXACCEPT
Indicates whether the transaction just processed had an acceptable error level. Valid values are:

Y   The transaction translated with an acceptable error level (less than or equal to the error level specified in the trading partner usage/rule).

N   The transaction had an unacceptable error level.

TRABORT

Indicates whether an error found while processing the last request was so severe that the translator does not continue. Valid values are:

Y   The error was so severe that translation was halted.

(blank) No errors occurred.

FILEID

The ddname (TSQ name in CICS) to which an interchange should be written during a TRANSLATE-TO-STANDARD (without delayed enveloping) or ENVELOPE function. The value in this field overrides the default file name in the `Trans data queue` field from the network profile.

This is also the ddname from which transactions should be read during a translate file or deenvelope function. The value in this field overrides the default file name in the `Receive file name` field from the mailbox (requestor) profile.

DSNAME

The name of the data set to which transactions were written (`EJECT` is **Q**), or from which transaction data is being read (`NEWENV` is **Y**). In CICS, this field has the same value as the `QDDNAME` field.

QNETID

The network ID associated with the interchange that was written (`EJECT` is **Q**).

QPTTOPT

Indicates whether the network associated with the interchange that was written (`EJECT` is **Q**) is a point-to-point network. Valid values are:

Y   Point-to-point network

N   Not a point-to-point network

QSRPGM

Indicates whether the network associated with the interchange that was written (`EJECT` is **Q**) has a send/receive program defined. Valid values are:

Y   A send/receive program is defined

N   No send/receive program is defined

QDDNAME

The ddname of the file to which the interchange was written (`EJECT` is **Q**).

FUNACKFLE

The ddname (TSQ name in CICS) to which a functional acknowledgment interchange should be written if enveloping is not being delayed (`ENVLDELAY` is **N**). Applies only during deenveloping or translate-file functions. This field overrides the default file name in the `Trans data queue` field from the network profile.

Note: FASPEC must also be set to **Y**.

QTPNICK
The trading partner nickname associated with an interchange just written (`EJECT` is **Q**), or the trading partner nickname associated with a transaction just received.

QRC
The error return code on the attempt to write an interchange. Applies only when `EJECT` has a value of **E**.

QERC
The extended error return code on the attempt to write an interchange. Applies only when `EJECT` has a value of **E**.

ERRCDES
An array of ten binary values of 2 bytes each indicating the errors found while processing this transaction. You can use the `CLRERRS` field in this control block to reset the array. For more information about error codes, "Translator Error Codes" on page 670.

INMEMTRANS
A 2-byte binary value that indicates the number of transactions that should be maintained in storage before database updates are attempted. Applies only when using interchange level recovery. For related information about interchange level recovery, see the description of the `SCOPE` field on 664. This value affects the degree of concurrency you can achieve if you execute multiple processes at the same time.

Keeping transactions in storage can delay the application of a database lock and reduce the amount of time that the lock is held. The amount of storage used by each transaction depends on the function being performed, as follows:

ENVELOPE
> 1508 bytes per transaction. If encryption is taking place, you must add the size of an average transaction image.

TRANSLATE AND ENVELOPE
> 1820 bytes per transaction. You can subtract 240 from this value, if overrides from the C record are not being used. If encryption is taking place, you must add the size of an average transaction image.

DEENVELOPE, DEENVELOPE AND TRANSLATE
> 1508 bytes per transaction.

If interchange level recovery is being used and you do not specify a value for this field, the default value of **100** is used.

NOCOMMIT
Indicates whether COMMIT requests should be delayed until the application databases have been updated. Once the databases have been updated, an API request to commit should be issued. Valid values are:

Y   Does not issue a COMMIT. Overrides the value in the `SCOPE` field.

N   Issues COMMITs as specified in the `SCOPE` field.

SCOPE

Indicates the recovery scope that should be in place for this session. Valid values are:

T   Transaction recovery scope. The translator issues a COMMIT request at the end of every transaction during translate-to-standard or enveloping functions and at the beginning of every transaction during translate-to-application or deenveloping functions.

E   Interchange recovery scope. The translator issues a COMMIT request only after an interchange has been written during translate-to-standard or enveloping functions and on the next request after the last transaction of an interchange (`LASTINENV` field) during translate-to-application or deenveloping functions.

An interchange recovery scope is not recommended for interchanges that contain a large number of transactions because the number of locks that can be obtained by a process is limited in DB2. If this limit is exceeded, DB2 automatically issues a ROLLBACK.

When setting the value in this field, you must consider the value you set in `INMEMTRANS`. If the number of transactions in an interchange exceeds the value in `INMEMTRANS`, other concurrent processes are blocked from the time the value is exceeded until the interchange has been completely processed.

TPNICK

The trading partner nickname associated with the current transaction.

CONCATENATE

Indicates whether data extraction records should be concatenated in the output data block. Applies only to data extraction requests. Valid values are:

Y   Concatenates data extraction records.

N   Does not concatenate data extraction records.

ASSERTLVL

During mapping, the &ASSERT*n* special literal can be used to establish assertions about the transaction. For example, you can specify that the total amount of a transaction does not exceed one million dollars. The *n* in &ASSERT*n* is the assertion level that should be active for this translation. Only &ASSERT*n* special literals with an *n* value greater than or equal to the value in this field are executed. The default assertion level is **0**, indicating that all assertions apply.

RAWDATAOUT

Indicates whether raw data format should be used for output data resulting from a fixed-to-fixed translation. Valid values are:

Y   Uses raw data format for output from fixed-to-fixed translation.

N   Does not use raw data format for output from fixed-to-fixed translation.

FIXEDTRX

A returned value that indicates whether the current transaction used fixed-to-fixed translation.

MRREQID

During a translate file or deenvelope function, you can provide the name of a requestor ID in the MRREQID field. If this value is specified, the management reporting component of WebSphere Data Interchange is notified of the number of bytes in the interchanges processed in the session. Specify this field only if the interchanges being processed have not already been counted. By default, the data is counted when it was received using WebSphere Data Interchange communications functions and entering a value in this field is not necessary.

ERRFILTER

The initial list of errors that should be filtered during this translation session so that error messages are not created for them. The list of errors provided here is active at the start of each transaction and provides the initial values for the DIERRFILTER named variable. Separate the entries with a blank or comma. You can specify a range of codes by using a dash (-) between the low and high values of the range. For example, to filter all warning error codes (0 through 99) plus error codes 103 and 105, specify **0-99,103,105** in this field. For a list of error codes, see "Translator Error Codes" on page 670.

The following errors cannot be filtered and attempts to do so will be ignored:

* **106-110, 117-118**
* **204, 207-210**
* **301, 303, 305, 308-334**
* **401, 403, 405-411**
* **501, 505-509**
* **601, 602, 604**
* **900-999**

If the following errors are filtered, the error is ignored and a transaction, group or interchange is processed in spite of the inconsistency:
302, 304
    Transaction header/trailer inconsistent
402, 404
    Group header/trailer inconsistent
502, 503
    Interchange header/trailer inconsistent

When the error occurs before or after the usages/rules are processed, error codes **3-6** (messages TR0403-TR0406) cannot be filtered using the DIERRFILTER field in the usage/rule. However, you can filter these errors using the DIERRFILTER keyword on the PERFORM statement.

FFILEID

The ddname (TSQ name in CICS) to which the result of a fixed-to-fixed translation should be written during a translate-to-standard (without delayed enveloping) or envelope functions. This field overrides the default file name that is the concatenation of the `Application file name` field from the target data format and the `File suffix` field from the trading partner profile.

VARTRACE

Indicates whether a variable level translator trace is wanted (for WebSphere Data Interchange use only).

EXPTRACE

Indicates whether an expression level translator trace is wanted (for WebSphere Data Interchange use only).

SSEGVAL

Indicates the level of service segment validation that should take place. The service segments are the segments used when a transaction is enveloped (ISA, GS, ST, UNB, UNH, UNT, and so on). If you do not specify this field, no validation is performed. You can use the `SERVICESEGVAL` keyword on the PERFORM command to set this value. Valid values are:

1   Validates service segments for syntax only. This includes checking for mandatory data that is missing, as well as data elements that are too large or too small.

2   In addition to checking syntax, validates the values in the service segment data elements according to their types (only dates and times are validated), and if a validation table has been specified, checks the value of the data element against the validation table.

TRXFACODE

The type of functional acknowledgment that was generated for this transaction. Valid values are:

A   Accepted
R   Rejected
E   Accepted with errors

IUSEREXIT

During send processing, the name of a user exit that is given to each interchange as it is created by WebSphere Data Interchange. During receive processing, the name of a user exit that will provide the interchange to be processed by WebSphere Data Interchange.

IUSERAREA

The 4-bytes of information that is returned to the program identified by `IUSEREXIT`.

IUSERACCESS

Indicates how the interchange should be presented to the `IUSEREXIT` program. Valid values are:

M   Delivers the interchange to the exit in virtual storage. Applies only when
          `IUSERTYPE` contains **UE** (user exit).
F   Delivers the interchange to the exit in a file. The interchange is first written to
          the TD queue file, and then the IUSEREXIT program is invoked.

IUSERTYPE

The type of program specified in `IUSEREXIT`. Valid values are:
PG   A program that should be linked to (EXEC CICS LINK in CICS).
UE   A WebSphere Data Interchange user exit program defined in the User Exits
          (ADAMCTL) profile.

MAPCHAIN

Indicates whether a transaction will be translated multiple times using different
translation usage/rules. Valid values are:
Y   Translates the current transaction again rather than translating the next
          transaction.
(other) Translates the next transaction.

FORCETEST

Indicates whether the deenvelope and/or translate-to-application process is forced to
select only a test usage/rule regardless of the value of the test indicator in the
envelope. Valid values are:
Y   Forces the process to test mode and select only a test usage/rule (if defined).
          If a test usage/rule is not found, an error is generated and the transaction is
          rejected.

          If this value is used on the deenvelope process, it must also be used on the
          TRANSLATE-TO-APPLICATION or RETRANSLATE-TO-APPLICATION
          commands to select those transactions stored by the deenvelope process.
N   Uses the test indicator from the envelope to determine the usage/rule to select
          (default). An envelope without a test indicator is always considered a
          production envelope.

ENVPRBRK

Specifies whether a change in the EDI standard envelope member name should create
a new interchange envelope or a new group envelope. Valid values are:
Y   Creates a new interchange envelope when the EDI standard envelope profile
          member name changes.
(other) Creates a new group envelope when the EDI standard envelope profile
          member name changes (default).

SUSBLKF

Indicates whether a suspend block has been allocated (CICS API only). If the API
program supplies the suspend block (32 bytes, initialized with binary zeroes), the CICS
SUSPENDs will be in effect over multiple translator calls. By having WebSphere Data
Interchange issue periodic SUSPENDs, AICA abends can be eliminated. Valid values
are:
Y   A suspend block has been allocated and its address is in SUSBLKP.
(other) No suspend block has been allocated.

CLRERRS

Indicates whether the ERRCDES array should be reset before processing continues. Valid values are:

Y   Resets the ERRCDES array before processing.

(other) Does not reset the ERRCDES array before processing.

FARC

The return code from the translator for the functional acknowledgment translation done during deenveloping. For more information about return codes, refer to .

FAERC

The extended return code from the translator for the functional acknowledgment translation done during deenveloping. For more information about extended return codes, refer to .

SAPUPDT

Indicates that SAP updates are requested (for WebSphere Data Interchange use only).

SAPTRX

The SAP transaction indicator (for WebSphere Data Interchange use only).

SAPCLIENT

The SAP client code (for WebSphere Data Interchange use only).

SAPDOCNUM

The SAP document number (for WebSphere Data Interchange use only).

SAPSEQ

The SAP error record sequence number (for WebSphere Data Interchange use only).

ROUTCODE

A 3-character generic routing code provided by the application and used by WebSphere Data Interchange to select a generic send usage/rule. A blank specifies a default generic send usage/rule.

FAEREQ

Indicates whether a functional acknowledgment file is required. Valid values are:

Y   A functional acknowledgment is required. An error is produced if the file is not available.

(other) A functional acknowledgment is not required.

BOUNDARY

Indicates whether incremental translation should be used. During regular (non-incremental) translation, this field should be blank. For more information about incremental translation, see "Outbound incremental translation" on page 472.

SUSBLKP

The suspend block pointer. Applies only to the CICS API. If the API program supplies a suspend block (32 bytes, initialized with binary zeroes), this field should contain the address of the block and the SUSBLKF field should contain **Y**.

CUSERDATA

On send translation of C and D records, the user area provided by the user. On receive translation, this value comes from the WebSphere Data Interchange variable DICUSERDATA. This field can be modified by user exits and is copied to the C record before the C record is output by WebSphere Data Interchange. The default is all blanks.

APPLTPID

The ID of an application trading partner (or internal trading partner within your business organization such as a division or department). The application trading partner ID can be used when no specific EDI trading partner is defined, or in combination with an EDI trading partner. Interchange control numbers are generated using a combination of the application and EDI trading partner IDs. This trading partner ID should be used when multiple application trading partners do business with the same EDI trading partner.

EXTENDC

Indicates whether extended C record format should be used. Valid values are:

Y   Uses extended C record format.
(blank) Does not use extended C record format (default).

VAXFLAG

Indicates whether pageable translation should be enabled. Valid values are:

X   Enables pageable translation.
(blank) Does not enable pageable translation.

For more information, see the PAGE keyword description on 166.

GDATE8

The date of the group envelope in 8-byte date format.

RECOVBAD

Indicates whether the translator should try to recover from bad EDI standard data. The translator checks for EDI standard interchange headers when a segment terminator is not found on a particular standard segment. The translator then attempts to reset the delimiters and check the current segment/record for the segment terminator. Valid values are:

Y   Tries to recover from bad EDI standard data (default).
N   Does not try to recover from bad EDI standard data.

RSRVD4

Reserved for WebSphere Data Interchange.

Translator Error Codes

The following sections identify the error codes that are generated by the translator.

Translator warnings

Table 221. Translator warnings

Code	Msg	Code	Msg	Code	Msg	Code	Msg
1	TR0401	2	TR0841	3	TR0403	4	TR0404
5	TR0405	6	TR0406	7	TR0407	8	TR0408
9	TR0409						

Field-level translator errors

Table 222. Field-level translator errors

Code	Msg	Code	Msg	Code	Msg	Code	Msg
101	TR0001	102	TR0002	103	TR0003	104	TR0004
105	TR0005	106	TR0006	107	TR0007	108	TR0008
109	TR0009	110	TR0014	111	TR0010	112	TR0011
113	TR0012	114	TR0013	115	TR0015	116	TR0016
117	TR0017	118	TR0018	119	TR0023	120	TR0024
199	TR0026						

Segment-level translator errors

Table 223. Segment-level translator errors

Code	Msg	Code	Msg	Code	Msg	Code	Msg
201	TR0050	202 ()	TR0051	203	TR0052	204	TR0053
205	TR0054	206 ()	TR0055	207	TR0019	208	TR0020
209	TR0021	210 ()	TR0022	211	TR0056	212	TR0058
213	TR0059	214 ()	TR0060	215	TR0057		

Transaction-level translator errors

Table 224. Transaction-level translator errors

Code	Msg	Code	Msg	Code	Msg	Code	Msg
302	TR0101	304	TR0103	305	TR0104	306	TR0821
307	TR0818	308	TR0820	309	TR0822	310	TR0825
311	TR0826	312	TR0830	313	TR0834	314	SA0042
315	TR0105	316	TR0106	317	TR0107	318	TR0108
319	TR0109	320	TR0110	321	TR0111	322	TR0112
323	TR0850	324	TR0113	325	TR0114	326	TR0025

Table 224. Transaction-level translator errors (continued)

Code	Msg	Code	Msg	Code	Msg	Code	Msg
327	TR0115	328	TR0116	329	TR0848	330	TR0117
331	TR0118	332	TR0119	333	TR0120	334	TR0121
335	TR1257	336	TR1258	337	TR1259	338	TR1260
339	TR1261	340	TR1262	341	TR0122		

Group-level translator errors

Table 225. Group-level translator errors

Code	Msg	Code	Msg	Code	Msg	Code	Msg
301	TR0100	303	TR0102	402	TR0151	404	TR0153
405	TR0154	406	TR0155	407	TR0156	408	TR0157
409	TR0107	410	TR0108	411	TR0158	412	TR1257
413	TR1258	414	TR1259	415	TR1260	416	TR1261
417	TR1262						

Interchange-level translator errors

Table 226. Interchange-level translator errors

Code	Msg	Code	Msg	Code	Msg	Code	Msg
401	TR0150	403	TR0152	501	TR0201	502	TR0203
503	TR0205	504	TR0206	505	TR0824	901	TR0810
902	TR0811	903	TR0812	904	TR0815	905	TR0816
906	TR0817	907	TR0843	908	TR0827	909	TR0828
910	TR0829	911	TR0832	912	TR0836	913	TR0838
914	TR0839	915	TR0840	916	TR1201	917	TR1202
918	TR1203	919	TR1205	920	TR1206	921	SA0042
922	TR0846	923	TR0847	924	TR0848	925	TR0849
926	TR0851	927	TR1207	928	TR1208	929	TR1209
930	TR1252	931	TR1253	932	TR1254	933	TR1255
934	TR1256	935	TR1257	936	TR1258	937	TR1259
938	TR1260	939	TR1261	940	TR1262	941	TR1263

Translator Input Data Block (TRIDB)

The translator input data block (TRIDB) is required on all service requests for translation, enveloping, and data extraction. For more information about these services, see "Translation services" on page 443, "Enveloping services" on page 500, and "Data extraction services" on page 535

Table 227 describes how the TRIDB is used for all functions and the initialization requirements for each.

Table 227. TRIDB functions and initialization requirements

Function code:	Initialization requirements:
1	A BLKLEN of 16 is sufficient. For more information, see "Retrieve interchange header API" on page 533
2	A BLKLEN of 16 is sufficient. For more information, see "Retrieve group header API" on page 533
3	A BLKLEN of 16 is sufficient. For more information, see "Retrieve transaction header API" on page 534
111 113	The DATA field contains the application data. The DATALEN field identifies the length of the data in the DATA field. The block has no minimum length and it can be as large as necessary to hold the application data. For more information, see "Translate-to-standard API" on page 446
211 212	TRIDB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "Translate-to-application API" on page 474
213	TRIDB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "Translate specific API" on page 478
214	TRIDB is used as a work buffer to hold a segment from the transaction. The minimum BLKLEN is 32000 but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "Deenvelope API" on page 518
215	A BLKLEN of 16 is sufficient. For more information, see "Envelope API" on page 503
216	A BLKLEN of 16 is sufficient. For more information, see "Retrieve detailed data API" on page 536
217	A BLKLEN of 16 is sufficient. For more information, see "Retrieve transaction image API" on page 539
218	A BLKLEN of 16 is sufficient. For more information, see "Retrieve transaction acknowledgment image API" on page 539
219	A BLKLEN of 16 is sufficient. For more information, see "Retrieve functional acknowledgment image API" on page 539
990	A BLKLEN of 16 is sufficient. For more information, see "Close and queue interchange API" on page 516
991	A BLKLEN of 16 is sufficient. For more information, see "Issue commit API" on page 532

Table 227. TRIDB functions and initialization requirements (continued)

Function code:	Initialization requirements:
1000	A BLKLEN of 16 is sufficient. For more information, see "End translation/enveloping API" on page 517

Table 228 describes how to define the TRIDB with 4 byte lengths.

Table 228. Defining the TRIDB with 4 byte lengths

Name	Offset	Length	Type	Description
BLKLEN	0	4	Bin	Block length
RESERVED	4	8	Hex	Reserved
DATALEN	12	4	Bin	Length of the data in the DATA field
DATA	16	Variable	Char	Application data for function codes **111** and **131**

TRIDB field descriptions

BLKLEN
The length of the TRIDB.

RESERVED
Initialize with binary zeros.

DATALEN
For function codes **111** and **131**, the amount of data being provided in the DATA field.

DATA
For function codes **111** and **131**, the application data that should be translated. The format of the data must match the definition in the data format.

Translator Output Data Block (TRODB)

The translator output data block (TRODB) is required on all service requests for translation, enveloping, and data extraction. For more information, see "Translation services" on page 443, "Enveloping services" on page 500, and "Data extraction services" on page 535

The TRODB always has a minimum length of 32000. Entering a value less than 32000 in the `BLKLEN` field results in a TR0829I error message, and the translator terminates. There is no maximum length for the TRODB.

Table 229 describes shows how the TRODB is used for all functions and the initialization requirements for each.

Table 229. TRODB functions and initialization requirements

Function code:	Initialization requirements:
1	The `DATA` field contains the interchange header image. The `DATALEN` field identifies the number of bytes in the header segment. For more information, see "Retrieve interchange header API" on page 533
2	The `DATA` field contains the group header image. The `DATALEN` field identifies the number of bytes in the header segment. For more information, see "Retrieve group header API" on page 533
3	The `DATA` field contains the transaction header image. The `DATALEN` field identifies the number of bytes in the header segment. For more information, see "Retrieve transaction header API" on page 534
111	TRODB is used as a work buffer to hold a segment from the transaction. The minimum `BLKLEN` value is 32000 but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "Translate-to-standard API" on page 452
131	TRODB is used as a work buffer to hold a segment from the transaction. The minimum `BLKLEN` value is 32000 but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "Translate-to-standard API" on page 452
211	The `DATA` field contains the application data. The `DATALEN` field identifies the length of the data in the `DATA` field. For more information, see "Translate-to-application API" on page 474
212	The `DATA` field contains the application data. The `DATALEN` field identifies the length of the data in the `DATA` field. For more information, see "Translate-file-to-application API" on page 486
213	The `DATA` field contains the application data. The `DATALEN` field identifies the length of the data in the `DATA` field. For more information, see "Translate-to-application API" on page 478
214	TRODB is used as a work buffer to hold a segment from the transaction. The minimum `BLKLEN value` is 32000 but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "Deenvelope API" on page 518

Table 229. TRODB functions and initialization requirements (continued)

Function code:	Initialization requirements:
215	TRODB is used as a work buffer to hold a segment from the transaction. The minimum `BLKLEN value` is 32000, but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "Envelope API" on page 503
216	The `DATA` field contains the data extraction detail record. The `DATALEN` field identifies the number of bytes in the record. For more information, see "Retrieve detailed data API" on page 536
217	The `DATA` field contains the image record and the `DATALEN` field identifies the number of bytes in the record. For more information, see "Retrieve transaction image API" on page 539
218	The `DATA` field contains the image record. The `DATALEN` field identifies the number of bytes in the record. For more information, see "Retrieve transaction acknowledgment image API" on page 539
219	The `DATA` field contains the image record. The `DATALEN` field identifies the number of bytes in the record. For more information, see "Retrieve functional acknowledgment image API" on page 539
990	TRODB is used as a work buffer to hold a segment from the transaction. The minimum `BLKLEN` value is 32000, but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "Close and queue interchange API" on page 516
991	TRODB is used as a work buffer to hold a segment from the transaction. The minimum `BLKLEN` value is 32000, but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "Issue commit API" on page 532
1000	TRODB is used as a work buffer to hold a segment from the transaction. The minimum `BLKLEN` value is 32000, but should be large enough to hold the largest segment, excluding the binary segment. For more information, see "End translation/enveloping API" on page 517

Table 230 describes how to define the TRODB with 4 byte lengths.

Table 230. Defining the TRODB with 4 byte lengths

Name	Offset	Length	Type	Description
BLKLEN	0	4	Bin	Block length
RESERVED	4	8	Hex	Reserved
DATALEN	12	4	Bin	Length of data in the `DATA` field
DATA	16	Variable	Char	Application data for function codes **211**, **212**, and **213**

TRODB field descriptions

BLKLEN
The length of the TRODB. Maximum length is 32000.

RESERVED
Initialized with binary zeros.

DATALEN
For function codes **211–213** and **216–219**, the amount of data being provided in the
DATA field.

DATA
For function codes **211–213**, the application data that were produced. The format of the
data must match the definition in the data format.

For function codes **216–219**, the detail extraction records that were produced.

Communication Control Block (CMCB)

Table 231 describes the fields in the communication control block (CMCB). The offset values in this table are relative to **0**. If this control block is used in network commands (NETOP) profile, you should add **1** to the offset value, because the offsets used for the Network Commands (NETOP) profile are relative to **1**.

Table 231. Definition of communication interface control block

Name	Offset	Length	Type	Description
BLKLEN	0	2	Bin	Length of the CMCB
RESERV1	2	2	Bin	Reserved
BLKNME	4	8	Char	Block name
TPNICKNM	12	16	Char	Trading partner nickname
NETID	28	8	Char	Network ID
NETOP	36	8	Char	Network command
REQID	44	16	Char	Requestor ID
SEQNUM	60	5	Char	Message/file/transaction sequence number
CONTRCV	65	1	Char	Continuous receive flag
FTYPE	66	2	Char	File type
FILERCVD	68	1	Char	File received flag
RESERV2	69	5	Char	Reserved
CLRFILE	74	1	Char	Clear file indicator
DATAFMT	75	1	Char	Format of data being sent
ACCTYP	76	1	Char	Account type
DATATYP	77	1	Char	Data type
RECVTYP	78	1	Char	Type of receive
DCIND	79	1	Char	Delivery class
ACKIND	80	1	Char	Acknowledgment code
RESRECL	81	1	Char	Resolution of RECLEN
SCRIPT	82	8	Char	Script name
ENAME	90	8	Char	Envelope name
MSGNAME	98	8	Char	Message name
FILENAME	106	56	Char	File name
CANSD	162	6	Char	Cancel start date
CANST	168	6	Char	Cancel start time
CANED	174	6	Char	Cancel end date
CANET	180	6	Char	Cancel end time
TMZONE	186	1	Char	Time zone
MODEM	187	1	Char	Modem type
NPSSCDE	188	5	Char	Start session response code

Table 231. Definition of communication interface control block (continued)

Name	Offset	Length	Type	Description
NPESCDE	193	5	Char	End session response code
NPERRCD	198	5	Char	Error code
NPSEVER	203	2	Char	Error severity
BLKTYPE	205	1	Char	Block type
FQUEUED	206	1	Char	Queued functions indicator
FMSGS	207	1	Char	Free-form messages indicator
FFILE	208	1	Char	Free-form files indicator
FEDIX	209	1	Char	EDI ISA/IEA file indicator
FEDIE	210	1	Char	EDI UNB/UNZ file indicator
FEDIU	211	1	Char	EDI BG/EG file indicator
FEDIG	212	1	Char	EDI GS/GE file indicator
FEDII	213	1	Char	EDI ICS/ICE file indicator
FEDIT	214	1	Char	EDI STX/END file indicator
FCANCEL	215	1	Char	CANCEL indicator
FCLASS	216	1	Char	Message class indicator
FACK	217	1	Char	Network acknowledgment indicator
FSYSMSG	218	1	Char	System messages indicator
FRCVBTP	219	1	Char	Receive by trading partner indicator
FRESTART	220	1	Char	Restart indicator
FNOUSERID	221	1	Char	Account number indicator
FACCTSEP	222	1	Char	Value used to separate account and user ID
DDCOLON	223	3	Char	DD: string
RESERV3	226	2	Char	Reserved
ADMTYPE	228	2	Char	Administrative response file type
UNIQID	230	8	Char	Unique ID returned by the network program
SAPUPDT	238	1	Char	SAP update flag
FSENTNET	239	1	Char	Skip send requested status flag
RESERV4	240	14	Char	Reserved for WebSphere Data Interchange

CMCB field descriptions

The following descriptions cover only the fields that apply to the API. Your definition, however, must include all fields in the block. The descriptions apply to all functions for which the block is used, unless exceptions are stated.

BLKLEN
A 2-byte binary field that contains the length of the CMCB control block. A CMCB is 254 bytes.

RESERV1
Reserved.

BLKNME
The name of this block which is **EDICMCB**.

TPNICKNM
The key for a member of the trading partner profile. For receiving, Communications supplies the nickname for the sender of the first file received if your application does not supply it.

NETID
The key for a member of the network profile. Your application must supply the network ID if it does not supply a requestor ID. If the application supplies a requestor ID, Communications uses the network ID from the mailbox (requestor) profile and ignores this field. For related information, see `REQID` on 680.

NETOP
Specifies the commands that should be built for the network program to process. Valid values are:

SENDFILE
> Sends non-EDI file for function codes **0121** and **0221**.

SENDX12
> Sends X12 standard transactions for function code **0211**.

SENDEDI
> Sends EDIFACT or UN/TDI standard transactions for function code **0211**.

SENDUCS
> Sends UCS standard transactions for function code **0211**.

RECVFILE
> Receives non-EDI file for function codes **0132** and 0232.

RECVX12
> Receives X12 standard transactions for function codes **0132** and **0232**.

RECVEDI
> Receives EDIFACT or UN/TDI standard transactions for function codes **0132** and **0232**.

RECVUCS
> Receives UCS standard transactions for function codes **132** and **232**.

CANCEL
> Cancels delivery of a file or message for function codes **133** and **233**.

NO-NETOP

> Processes a file of responses for function code **252**, but does not invoke a network program. Used if you have network acknowledgment responses in a file that have not been processed.

REQID

The key for a member of the mailbox (requestor) profile. Communication gets the network ID from the mailbox (requestor) profile member. Your application must supply the network ID if it does not supply the requestor ID. For related information, see `NETID` on 679.

SEQNUM

The sequence number assigned to a transaction, file, or message. For sending or queuing to send, Communication returns the sequence number. For canceling a file or message, your application supplies the sequence number of the file or message to be canceled.

CONTRCV

Indicates whether continuous receive should be active. Applies only to CICS. Valid values are:

(blank) Starts single receive
C   Starts continuous receive
E   Ends continuous receive

FTYPE

Indicates the destination file type for received transactions. Valid values are:

MQ   MQSeries queue
TM   TS queue - main storage
TS   TS queue - auxiliary storage
PG   Program

FILERCVD

Indicates whether a file was received. Valid values are:

Y   File was received.
(other) File was not received.

RESERV2

Reserved.

CLRFILE

Indicates when the Communications service is to clear the file you are sending. Applies only to immediate sending of EDI standard transactions and non-EDI files. Valid values are:

Y   Clears the file after successfully sending the transaction data or non-EDI file.
U   Always clears the file.

DATAFMT

Indicates whether data is in binary format. The first character of the `Envelope name` field defines the receive class used by the Expedite product interface. Expedite will set the data type to A or E, depending on which Expedite product you are using. Valid values are:

B   Binary data
(blank) Not binary data (default)

ACCTYP

For sending or canceling data, your application must supply a value of **D**, which indicates that the trading partner ID is an account number or user ID (Network reference: DESTTYP). For receiving data, your application must supply one of the following values (IN reference: SRCTYP):

(blank) Receives from any source
D   Receives from a trading partner identified by an account number or user ID

DATATYP

The type of file name supplied in the `FILENAME` field. Valid values are:

A   Data set name
D   ddname

For sending, your application must supply the data type. You can use a ddname only for immediate sends.

For receiving, the communications service supplies the data type if:
* The application does not supply the data type.
* `NETOP` contains RECVX12, RECVEDI, or RECVUCS.
* Immediate receive (function code **232)** is requested.

The Expedite parameters for network profiles is FILEID.

RECVTYP

The type of messages to receive. Valid values are:

G   Receives only the first message or file that matches the trading partner nickname, the envelope name, or both. For example, receive only X12 envelopes from partner ABC.
(blank) Receives all messages and files that meet the receive criteria.

The Expedite parameter for network profiles is ALLFILES.

DCIND

The delivery class. Valid values are:

(blank) Normal delivery.
P   High priority delivery.
I   Express delivery. Might require that the receiving partner be signed on to the network when the data is sent.

The Expedite parameter for network profiles is PRIORTY.

ACKIND

The type of acknowledgment requested. For sending, valid values are:

(blank) No acknowledgments
R Receipt only
D Delivery only
B Both receipt and delivery
A Purge only
C Both receipt and purge
E Either receipt or purge
F Receipt and either delivery or purge
(binary zero)
Network acknowledgement code (`NETACK`) from the trading partner profile is returned in this field

For canceling, valid values are:

(blank) No acknowledgements
H Only header information in the acknowledgement
T Both header and text information in the acknowledgement

The Expedite parameter is ACK.

RESRECL

Indicates whether the received data set records and the output records must be the same length. Valid values are:

E Applies only to IINB41. Ends the session with an error if the length of the output record is greater than the length of the data set record.
S Splits the output records to fit the length defined for the data set records.

You do not need to set this field if the storage format (`STGFRMT` from the trading partner profile) contains **C** or **D**. In these cases, the length of the output record is set to the length of the data set record.

The Expedite parameter is RESRECL.

SCRIPT

The name of the script that your communications software should follow when processing requests for service. The script would be part of your communication software package and not part of WebSphere Data Interchange.

ENAME

The envelope name. For sending and canceling, if your application supplies a null value (binary zero), Communications uses the value in the `Message user class` field from the mailbox (requestor) profile and returns it here. If the mailbox (requestor) profile does not supply a value, the default is **#IDI#**.

For receiving, if your application supplies a null value, Communications uses the value in the `Message user class` field from the mailbox (requestor) profile and returns the message user class of the first file received.

The Expedite parameter is ACK.

MSGNAME
The message name. You can define any value that fits in the field length, or use blanks. For sending or canceling, your application supplies the message name. The default is **#IDI#**.

For receiving, communications returns the message name of the first file received.

The Expedite parameter is MSGNAME.

FILENAME
The name of a file containing data to be sent or into which data is to be received from the network. The `Data type` field (DATATYP) indicates whether the name is a data set name or a ddname.

Your application must supply the file name for sending non-EDI files (function codes **121** and **221**). You can use a ddname only for immediate sends. For sending EDI standard transactions (function code **211**), this field is optional. If you do not specify a file name, Communications uses the ddname in the `Transaction data queue` field from the network profile member. When sending EDIFACT or UN/TDI requests, an **E** is appended to the ddname. When sending UCS requests, a **U** is appended to the ddname. The ddname is used as supplied for sending X12 requests. Communications also sets the `Data type` field (DATATYP) to **D**.

For receiving non-EDI files, your application must supply the file name. You can use a ddname only for immediate receives. For receiving EDI standard transactions, this field is optional. If you do not specify a file name, Communications uses the ddname in the `Receive file name` field from the mailbox (requestor) profile member. WebSphere Data Interchange also sets the `Data type` field (DATATYP) to **D**.

The Expedite parameter for network profiles is FILEID.

CANSD
The cancellation start date in `YYMMDD` format. Applies only to canceling files and messages.

The Expedite parameter for network profiles is STARTDATE.

CANST
The cancellation start time in `HHMMSS` format. Applies only to canceling files and messages.

The Expedite parameter for network profiles is STARTTIME.

CANED
The cancellation end date in `YYMMDD` format. Applies only to canceling files and messages.

The Expedite parameter is ENDDATE.

CANET

The cancellation end time in HHMMSS format. Applies only to canceling files and messages.

The Expedite parameter is ENDTIME.

TMZONE

The time zone for cancellation requests. Valid values are:
L   Local time
G   Greenwich Mean Time

The Expedite parameter is TIMEZONE.

MODEM

The type of modem used.

NPSSCDE

The network program start-session response code.

NPESCDE

The network program end-session response code.

NPERRCD

The network program error code.

NPSEVER

The network program error severity code.

BLKTYPE

The type of data blocks passed to communications for sending messages. Applies only to sending messages using function codes **141** and **241**. Valid values are:
H   Data blocks are larger that 32 K bytes
(other) Data blocks are smaller that 32 K bytes

FQUEUED

Indicates whether the network supports queued functions. Valid values are:
Y   Supports queued functions
(other) Does not support queues functions

FMSGS

Indicates whether the network supports free-form messages. Valid values are:
Y   Supports free-form messages
(other) Does not support free-form messages

FFILE

Indicates whether the network supports non-EDI files. Valid values are:

Y   Supports non-EDI files

(other) Does not support non-EDI files

FEDIX

Indicates whether the network supports X12 ISA/IEA envelopes. Valid values are:

Y   Supports X12 ISA/IEA envelopes

(other) Does not support X12 ISA/IEA envelopes

FEDIE

Indicates whether the network supports EDIFACT UNB/UNZ envelopes. Valid values are:

Y   Supports EDIFACT UNB/UNZ envelopes

(other) Does not support EDIFACT UNB/UNZ envelopes

FEDIU

Indicates whether the network supports BG/EG envelopes. Valid values are:

Y   Supports BG/EG envelopes

(other) Does not support BG/EG envelopes

FEDIG

Indicates whether the network supports GS/GE envelopes. Valid values are:

Y   Supports GS/GE envelopes

(other) Does not support GS/GE envelopes

FEDII

Indicates whether the network supports ICS/ICE envelopes. Valid values are:

Y   Supports ICS/ICE envelopes

(other) Does not support ICS/ICE envelopes

FEDIT

Indicates whether the network supports STX/END envelopes. Valid values are:

Y   Supports STX/END envelopes

(other) Does not support STX/END envelopes

FCANCEL

Indicates whether the network supports the CANCEL function. Valid values are:

Y   Supports CANCEL functions

(other) Dos not support CANCEL functions

FCLASS

Indicates whether the network supports user message classes. Valid values are:

Y   Supports queued functions

(other) Does not support queues functions

FACK

Indicates whether the network supports network acknowledgments. Valid values are:

Y   Supports network acknowledgements

(other) Does not support network acknowledgements

FSYSMSG

Indicates whether the network supports system messages. Valid values are:

Y   Supports system messages

(othre) Does not support system messgaes

FRCVBTP

Indicates whether the network supports receiving by trading partner. Valid values are:

Y   Supports receiving by trading partner ID

(other) Does not support receiving by trading partner ID

FRESTART

Indicates whether the network supports restart. Valid values are:

Y   Supports restart

(other) Does not support restart

FNOUSERID

Indicates whether the network supports account numbers only. Valid values are:

Y   Supports account numbers only

(other) Requires other information in addition to account number

FACCTSEP

The character used to separate the account number and user ID.

DDCOLON

The text string "**DD:**".

RESERV3

Reserved.

ADMTYPE

The type of administrative response file for CICS.

UNIQID

A unique ID returned by the network program.

SAPUPDT

Indicates whether VANI is to update SAP (for WebSphere Data Interchange use only).

FSENTNET

Indicates whether the status should be set directly to **Sent to network (**skipping over **Send requested** status). This is the case with network program EDIMQSR (the MQSeries network program). Valid values are:

Y   Skips the **Send requested** status and sets status to **Sent to network**
N   Does not skip the **Send requested** status

RESERV4

Reserved.

Trading Partner Profile Block (TPPDB)

Table 232 describes the trading partner profile block (TPPDB) All fields in this block are optional. When you specify a value, it is used in place of the corresponding value in the trading partner profile. When you specify `TPNICKNM` in the CMCB and leave it blank in this block, Communications returns a value for the fields that apply to the request.

The offset values in this table are relative to **0**. If this control block is used in the Network Commands (NETOP) profile, you should add **1** to the offset value, because the offsets used for the Network Commands profile are relative to **1**.

Table 232. Definition of the Trading Partner Profile Block

Name	Offset	Length	Type	Description
BLKLEN	0	2	Bin	Length of TPPDB
RESERV1	2	2	Bin	Reserved
BLKNME	4	8	Char	Block name
TPNICKNM	12	16	Char	Trading partner nickname
NETID	28	8	Char	Network ID
SYSQUAL	36	1	Char	Intersystem address qualifier
SYSID	37	8	Char	Intersystem ID
ACCTNUM	45	32	Char	Requestor's network account number
USERID	77	32	Char	Requestor's network user ID
ENVLQUAL	109	4	Char	Interchange qualifier
ENVLID	113	35	Char	Interchange sender/receiver ID
CONAME	148	40	Char	Company name
ADDR1	188	40	Char	Company address line 1
ADDR2	228	40	Char	Company address line 2
PHONE	268	25	Char	Contact phone number
CONTACT	293	30	Char	Contact name
PASSWORD	323	14	Char	Interchange password for send
RCVPASS	337	14	Char	Interchange password for receive
SECUID	351	8	Char	Network security profile member
NETCLS	359	1	Char	Network message class
NETCHG	360	1	Char	Network charges code
NETACK	361	1	Char	Network acknowledgment code
NETVCHK	362	1	Char	Destination verification code
NETRETN	363	3	Char	Mailbox retention period
NETEDIO	366	1	Char	Option for storing received data
NETEDIP	367	1	Char	Special processing requested for received data
STGFRMTO	368	1	Char	Storage format override
MACHTYPE	369	1	Char	Machine type

Table 232. Definition of the Trading Partner Profile Block (continued)

Name	Offset	Length	Type	Description
STGFRMT	370	1	Char	Storage format
EOTID	371	1	Char	End of text/message delimiter
LOGENV	372	1	Char	Log envelope data
FNGRPENV	373	1	Char	Send functional group
SEDELIM	374	1	Char	Sub-element delimiter
DEDELIM	375	1	Char	Data Element delimiter
SGDELIM	376	1	Char	Segment delimiter
SGSEP	377	1	Char	Segment ID separator
DECNOT	378	1	Char	Decimal notation
RLSCHAR	379	1	Char	Release character
TPICTLNO	380	9	Char	Interchange mask
TPGCTLNO	389	9	Char	Group mask
TPTCTLNO	398	9	Char	Transaction mask
COMMENT1	407	40	Char	Comment line 1
COMMENT2	447	40	Char	Comment line 2
NETCMDS	487	8	Char	Net commands PDS member
TPDATALINE	495	32	Char	Data line phone number
TIMEOUT	527	4	Char	Communications line timeout value
SEGMENTED	531	1	Char	Segmented output requested
SUFFIX	532	2	Char	File suffix
TPENVSUF	534	2	Char	Envelope profile member suffix
TPGENRCV	536	1	Char	Generic receive usages allowed
TPCMPRES	537	1	Char	Compress flag
TPRSRV1	538	8	Char	Reserved for WebSphere Data Interchange
TPSUPAD3	546	40	Char	Company address line 3
TPSUPCTY	586	30	Char	City name
TPSUPST	616	2	Char	State code
TPSUPPST	618	15	Char	Postal code
TPSUPCON	633	30	Char	Country code
TPSUPFAX	663	25	Char	Fax number
TPSUPU3	688	40	Char	Comment line 3
TPSUPU4	728	40	Char	Comment line 4
TPSUPU5	768	40	Char	Comment line 5
TPSUPU6	808	40	Char	Comment line 6
TPSUPU7	848	40	Char	Comment line 7
TPSUPU8	888	40	Char	Comment line 8

Table 232. Definition of the Trading Partner Profile Block (continued)

Name	Offset	Length	Type	Description
TPSUPU9	928	40	Char	Comment line 9
TPSUPU10	968	40	Char	Comment line 10
PRIORITY	1008	1	Char	Delivery priority
TPRSRV2	1009	3	Char	Reserved for WebSphere Data Interchange
DESCRIPT	1012	30	Char	Profile member description
LOGLOCK	1042	1	Char	Logical lock flag
LASTUID	1043	17	Char	User ID that performed the latest update
LASTUDT	1060	4	Bin	Date and time of the latest update
TPTYPE	1064	1	Char	Trading partner type
DESEP	1065	1	Char	Repeating data element separator
PROCESS	1066	40	Char	Associated process ID
TPRSRV3	1106	426	Char	Reserved for WebSphere Data Interchange

TPPDB field descriptions

BLKLEN
A 2-byte binary field that contains the length of the TPPDB data block. A TPPDB is 1532 bytes.

RESERV1
Reserved.

BLKNME
The name of this block.

TPNICKNM
The name you use to refer to the trading partner. The value in this field must identify a member from the trading partner profile.

NETID
The network ID. The value in this field must match the key field of a member of the network profile.

SYSQUAL
Indicates whether intersystem addressing is required for this trading partner. For the AT&T Global Network, must contain the value **I** if intersystem addressing is required (IN reference: DTBLTYP). Enter the ID of the other system in the SYSID field.

SYSID

For intersystem addressing, the ID of the system responsible for the receiver's account. The ID is limited to 3 characters.

The Expedite parameter for network profiles is SYSID.

ACCTNUM

The account number that the network assigns to the trading partner. Applies only to sending or receiving non-EDI files and messages. The entry must be left-justified. For sending and receiving EDI standard transactions using ISA/IEA envelopes, the last position must be blank. The combined value of this field and the `USERID` field must be a unique value. This field is required if you want to use network acknowledgments.

USERID

The user ID that the network assigns your trading partner. Applies only to sending or receiving non-EDI files and messages. The entry must be left-justified. The combined value of this field and the `ACCTNUM` field must be a unique value. Together, the account number and user ID make up the trading partner ID in the interchange envelope except for UCS (BG/EG) envelopes. For UCS envelopes, the phone number contains the trading partner ID. This field is required if you want to use network acknowledgments.

ENVLQUAL

The type of interchange ID used in the `ENVLID` field. The EDI standard defines these codes. If this field or the `Interchange ID` field (`ENVLID`) is blank, the enveloper takes the qualifier from the envelope profile member.

ENVLID

The ID used to fill in the interchange receiver ID field when you send to this partner, and to identify the interchange sender when you receive from this partner. If you leave this field blank, the enveloper uses the account number and user ID (or phone number for BG/EG interchanges).

CONAME

The name of the trading partner's company. The company name can be used as envelope data by using the **CO** envelope data type.

ADDR1

Line 1 of the trading partner's address.

ADDR2

Line 2 of the trading partner's address.

PHONE

The trading partner's telephone number. When enveloping type **U** (BG/EG) interchanges, if the `Interchange ID` field is blank, the value in this field is used as the

interchange receiver ID. When de-enveloping type **U** interchanges, if the
`Interchange ID` field is blank, the value in this field is used as the interchange
sender ID.

CONTACT
The name of the person you speak with when dealing with this trading partner.

PASSWORD
The password that you and your trading partner agreed upon for sending to this trading
partner. The value in this field corresponds to the **PW** data type in the interchange
envelope.

RCVPASS
The password that you and your trading partner agreed upon for receiving from this
trading partner. If this value matches the interchange password (**PW** data type) that
was received, translation occurs.

SECUID
The name of the network security profile member that specifies the encryption and
authentication processes that apply to EDI data. For sending, the trading partner
usage/rule specifies the network security profile member. If the send usage/rule does
not specify a member, the member specified here is used.

NETCLS
Indicates any special status of the data being sent. Applies only to send requests. Valid
values are:
(blank) Normal status
T Test status

The Expedite parameter for network profiles is MODE.

NETCHG
Indicates how charges are shared between sender and receiver. Applies only to send
requests. Valid values are:
1 Receiver pays all charges.
2 Receiver pays all charges if agreed to, or charges are split between sender
and receiver.
3 Receiver pays all charges if agreed to, or charges are split between sender
and receiver if agreed to. Otherwise, the sender pays all charges (default).
4 Charges are split between sender and receiver if agreed to. Otherwise, the
sender pays all charges.
5 Charges are split between sender and receiver.
6 Sender pays all charges.

The Expedite parameter for network profiles is CHARGE.

NETACK

Indicates which network acknowledgments are requested. Applies to send requests when the acknowledgment indicator in the CMCB contains binary zeros. For related information, see `ACKIND` on 682. The network specifies the acceptable values. Valid values are:

(blank) No acknowledgments
R Receipt only
D Delivery only
B Both receipt and delivery
A Purge only
C Both receipt and purge
E Either receipt or purge
F Receipt and either delivery or purge

The Expedite parameter for network profiles is ACK.

NETVCHK

Indicates whether the destination is verified before sending occurs. Valid values are:
N Does not verify the destination (default)
Y Requires verification
F Requests verification, and sends even if the destination is not verified (useful for intersystem addressing)

If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

The Expedite parameter for network profiles is VERIFY.

NETRETN

The number of days that data is to be kept in the network mailbox before it is purged, if it is not received. Enter blanks or zeroes to use the default number. For Expedite Base/z/OS, the valid range is **001-180**. For Expedite/CICS, the valid range is **001-099**, left-justified.

If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

NETEDIO

Indicates whether you want EDI segments stored in the receiving file as separate records. You provide the file name in the mailbox (requestor) profile. Valid values are:
Y Ends records at the segment delimiter (default)
N Does not end records at the segment delimiter

If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

The Expedite parameter is EDIOPT.

NETEDIP

Indicates whether the EDI data you receive has special EDI processing (breaking records by the segment delimiter). Valid values are:

Y   Performs EDI processing if the common data header indicates that the data is in EDI standard format (default)

N   Omits EDI processing, regardless of the common data header

If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

The Expedite parameter for network profiles is AUTOEDI.

STGFRMTO

Indicates whether you want to use the storage format defined in the common data header. Valid values are:

Y   Uses the storage format as defined in the common data header (default)

N   Ignores the storage format defined in the common data header

If there is no common data header, the format indicated in the `Storage format` field is used. If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

The Expedite parameter for network profiles is DLMOVERRIDE.

MACHTYPE

This field is not currently used.

STGFRMT

Indicates to the network how data is stored for free-form messages and files. Applies only to sending or receiving non-EDI files.

When determining what codes to select, consider the type of data you want to send and how the file is received. Contact a representative of each network you are using for all available codes. For example, if you are using:

For Expedite Base/z/OS (IEBASE), valid values are:

C   Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Select this option to send files containing program source code that is defined with variable length records. Output records do not include the carriage return and line-feed characters.

L   Precedes each record with a 2-byte hexadecimal record length. Select this option when sending data in fixed format or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.

N   Stores data as it is received. Output records are built based on the record length of the data set allocated to receive the data.

For Expedite/CICS, valid values are:

A   Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Select this option to send files containing program source code that is defined with variable length records. Output records do not include the carriage return and line-feed characters.

L   Precedes each record with a 2-byte hexadecimal record length. Select this option when sending data in fixed format or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.

O   Other (free-form).

The Expedite parameter for network profiles is DELIMITED.

EOTID

The character that signifies the end of the message text to the network. EOTID applies only to sending or receiving free-form messages.

The Expedite parameter for network profiles is ENDSTR.

LOGENV

Indicates whether EDI standard data will be logged. Applies only when the `Log standard data` field from the Application Defaults (APPDEFS) profile member does not contain a **Y** or an **N**. Valid values are:

Y   Logs EDI standard data

(other)   Does not log EDI standard data

FNGRPENV

Indicates whether functional groups will be created for transactions with type **E** (UNB/UNZ) envelopes. Functional groups are always created for type **I** (ICS/ICE), **U** (BG/EG), and **X** (ISA/IEA) envelopes, and they are never created for type **T** (STX/END) envelopes. Valid values are:

Y   Creates functional groups for type **E** envelopes

(other)   Does not create functional groups for type **E** envelopes

SEDELIM

The character that separates sub-elements (component data elements) in a transaction set. A value here (other than a low-value or space) overrides the character specified in the EDI standard. This value is only used when interchanges are created (not when they are received).

DEDELIM

The character that separates the data elements in a transaction set. A value here (other than a low-value or space) overrides the character specified in the EDI standard. This value is only used when interchanges are created (not when they are received).

SGDELIM
The character that marks the end of each segment in a transaction set. A value here (other than a low-value or space) overrides the character specified in the EDI standard. This value is only used when interchanges are created (not when they are received).

SGSEP
The character that separates the segment ID and the first data element in a segment for type **E** (UNB/UNZ) envelopes only. A value here (other than a low-value or space) overrides the character specified in the EDI standard. This value is only used when interchanges are created (not when they are received).

DECNOT
The character that represents decimal points in a transaction set. For type **E** (UNB/UNZ) envelopes, a value here (other than a low-value or space) overrides the character specified in the EDI standard. For all other types, a period represents the decimal point. This value is only used when interchanges are created (not when they are received).

RLSCHAR
For type **E** (UNB/UNZ) and **T** (STX/END) envelopes, this character indicates when a delimiter is being used as part of the data. A value here (other than a low-value or space) overrides the value specified in the EDI standard. This value is only used when interchanges are created (not when they are received).

TPICTLNO
The initial reference number that the enveloper places in the **CN** data type of the interchange header and trailer. This value is used as the base value for each trading partner/receiver ID combination. It does not represent the current control number for this trading partner.

TPGCTLNO
The initial reference number or special codes that the enveloper places in the **CN** data type of the functional group header and trailer. This value is used as the base value for each trading partner/receiver ID combination. It does not represent the current control number for this trading partner.

TPTCTLNO
The initial reference number or special codes that the enveloper places in the **CN** data type of the transaction set header and trailer. This value is used as the base value for each trading partner/receiver ID combination. It does not represent the current control number for this trading partner.

COMMENT1
A 40-byte area for free-form notes about the trading partner.

COMMENT2
A 40-byte area for free-form notes about the trading partner.

NETCMDS

The name of a member of a PDS that will be allocated to the ddname of EDINTCMD. This member will contain the commands that you want to pass to a network. WebSphere Data Interchange reads the commands from the PDS member and writes the commands to the network input file specified in the network profile member after all substitutable variable tags have been resolved by WebSphere Data Interchange.

TPDATALINE

The phone number used to connect your computer to talk directly to your trading partner's computer.

TIMEOUT

The maximum allowable time that the data line for communications can be idle without being dropped. If you specify a trading partner when requesting network activity (send or receive), the value for this field is taken from the trading partner profile. Otherwise, the value for this field is taken from the mailbox (requestor) profile.

SEGMENTED

Indicates whether you want EDI segments to be stored in the output file as separate records. Valid values are:

Y   Ends records at the segment delimiter
N   Does not end at the segment delimiter (default)

SUFFIX

A 2-character suffix for the ddname used to store the results of a fixed-to-fixed translation. The basic part of the ddname is taken from the `Application file name` field of the target data format.

TPENVSUF

A 2-character suffix for a generic EDI standard envelope profile member name. The basic part of the name is taken from the `Send or Receive usage override` field.

TPGENRCV

A code to indicate whether generic receive usages/rules are allowed for this trading partner. Valid values are:

Y   Allows generic receive usages/rules
(other)   Does not allow generic receive usages/rules

TPCMPRES

The Expedite Base/z/OS compression code. Applies only during send processing. Valid values are:

Y   Compresses the data.
N   Does not compress the data.
T   Expedite Base/z/OS uses its own table to decide whether to compress the data.

TPRSRV1

Reserved for WebSphere Data Interchange.

TPSUPAD3
Line 3 of the trading partner's address.

TPSUPCTY
The trading partner's city.

TPSUPST
The trading partner's state code.

TPSUPPST
The trading partner's postal code.

TPSUPCON
The trading partner's country.

TPSUPFAX
The trading partner's fax number.

TPSUPU3
A 40-byte area for free-form notes about the trading partner.

TPSUPU4
A 40-byte area for free-form notes about the trading partner.

TPSUPU5
A 40-byte area for free-form notes about the trading partner.

TPSUPU6
A 40-byte area for free-form notes about the trading partner.

TPSUPU7
A 40-byte area for free-form notes about the trading partner.

TPSUPU8
A 40-byte area for free-form notes about the trading partner.

TPSUPU9
A 40-byte area for free-form notes about the trading partner.

TPSUPU10
A 40-byte area for free-form notes about the trading partner.

PRIORITY
Indicates whether to prioritize delivery of messages.
(blank) Normal delivery
P   Priority delivery

TPRSRV2
Reserved for WebSphere Data Interchange.

DESCRIPT
The description of this profile member.

LOGLOCK
The profile member logical lock flag (for WebSphere Data Interchange use only).

LASTUID
The user ID of the last person to update this profile member.

LASTUDT
The date and time that this profile member was last updated.

TPTYPE
The trading partner type. Valid values are:

A　　　Application or internal trading partner
E　　　EDI or external trading partner (default)
B　　　Both an external and internal trading partner

DESEP
The repeating data element separator to be used for all transactions sent to this trading partner.

PROCESS
The ID of the business process associated with this trading partner such as `PRODUCTION_PURCHASING`.

TPRSRV3
Reserved for WebSphere Data Interchange.

Communication Data Block (DATABLK)

Your application assigns values to this block only when sending a message for function codes **141** and **241**. This block can be defined two ways:
- Data blocks that are 32-K bytes or less.
- Data blocks that are more than 32-K bytes.

Use the tables below to define the DATABLK. Use the BLKTYPE field in the CMCB to indicate which definition you are using.

Data blocks up to 32-K bytes

Table 233. Data blocks up to 32–K bytes

Name	Offset	Length	Type	Description
BLKLEN	0	2	Bin	Length of the data block
RESERV1	1	2	Bin	Reserved
BLKNME	2	8	Char	Name of this block
DATALEN	10	2	Bin	Length of the following message
DATA	12	Variable	Char	Text of the message

Data blocks more than 32-K bytes

Table 234. Data blocks greater than 32–K bytes

Name	Offset	Length	Type	Description
BLKLEN	0	4	Bin	Length of the data block
RESERV1	2	8	Char	Reserved
DATALEN	10	4	Bin	Length of the following message
DATA	12	Variable	Char	Text of the message

DATABLK field descriptions

BLKLEN
The length of the data block, including this field. The DATABLK length is either 14 or 16 bytes, plus the length of the message.

BLKNME
The name your application gives to the data block.

DATALEN
The length of the message in the DATA field. The message must not exceed 32-K bytes unless the BLKTYPE field of the CMCB contains a value of **other (**data blocks larger than 32-K bytes).

DATA

The message text. For the AT&T network, your application must arrange the message text in 80-byte segments that begin with the letter **T**.

Network Profile Block (NPDB)

The Network profile block (NPDB) contains the settings for the network you are using. A network profile block can be defined for each network you use.

The offset values in Table 235 are relative to **0**. If this control block is used in Network Commands (NETOP) profile, you should add **1** to the offset value, because the offsets used for the Network Commands profile are relative to **1**

Table 235. Definition of the Network Profile Block

Name	Offset	Length	Type	Description
BLKLEN	0	2	Bin	Length of NPDB
RESERV1	2	2	Bin	Reserved
BLKNME	4	8	Char	NPDB name
NETID	12	8	Char	Network ID
NETNME	20	30	Char	Network name
COMROT	50	8	Char	Communication routine name
NETPGM	58	8	Char	Network program name
PGMPARM	66	57	Char	Network program parameters
CMDIN	123	8	Char	Network command input file
CMDLRECL	131	4	Char	Network command record length
QDATA	135	8	Char	TD queue file
DATLRECL	143	4	Char	Transaction data record length
TMZONE	147	5	Char	Time zone
SYSTYP	152	8	Char	System type
SYSLVL	160	4	Char	System level
TXTHDR	164	1	Char	Message text header character
CMDOUT	165	8	Char	Network command output file
MSGROUT	173	8	Char	Program to process messages
SEQNUM	181	5	Char	Sequence number for network
NETACKFILE	186	8	Char	File where network acknowledgments are written
NETPHONE	194	32	Char	Dial connection phone number
SCRIPT	226	8	Char	Script name
FILLER	234	14	Char	Reserved for WebSphere Data Interchange
DESCRIPT	248	30	Char	Member description
LOGLOCK	278	1	Char	Logical lock flag
LASTUID	279	17	Char	User ID that performed the latest update
LASTUDT	296	4	Bin	Date and time of the latest update

NPDB field descriptions

BLKLEN
A 2-byte binary field that contains the length of the NPDB data block. An NPDB is 300 bytes.

RESERV1
Reserved.

BLKNME
The name of the network profile block. **EDINPDB.**

NETID
The ID of this network such as MYNET.

NETNME
The name of the network such as My EDI Network.

COMROT
The communication routine or the name of the main program that builds network commands and calls the network program to process the commands. WebSphere Data Interchange supplies the VANICICS, VANIINB1, VANIMQ, and PTTOPT programs. Any programs you write must be defined in a member of the User Exits (ADAMCTL) profile.

NETPGM
The name of the network program (such as IEBASE) that sends and receives the transactions, messages, and files.

PGMPARM
The network program parameters. The parameters that are passed to the network program for connecting to IBM services are NOSPIE,NOSTAE/,IBM0DIMR,,,,,,,Y.

CMDIN
The network command input file. The file that contains the commands the network program processes (IN reference: INMSG).

CMDLRECL
The network command record length or the length of records in the network command input file. Maximum length is 80.

QDATA
The file that contains EDI standard transactions that are waiting to be sent to trading partners. If you leave the field blank, the file has one of the following names by default, that represent the specified interchange envelope type:
QDATA
　　　　ISA/IEA. The transaction data is in X12 syntax.

> **QDATAE**
>> STX/END or UNB/UNZ. The transaction data is in EDIFACT or in UN/TDI syntax.
>
> **QDATAU**
>> BG/EG. The transaction data is in UCS syntax.

If you enter a name, an **E** is appended to the ddname for sending EDIFACT or UN/TDI requests, or a **U** is appended to the ddname for sending UCS requests. X12 requests use the ddname as supplied.

The type of send command issued (such as SENDX12) determines which file is used. For example, if you enter the name SENDPO and use the file to send EDIFACT transactions, WebSphere Data Interchange expects to find an allocation for the ddname SENDPOE.

DATLRECL
The length of records in the TD queue. For z/OS, the logical record length that you allocate for the file.

The Expedite parameter for network profiles is VANIINB1.

TMZONE
The time zone for your location. The network specifies the allowable codes.

The Expedite parameter is TIMEZONE.

SYSTYP
This field is not used by any currently supported network.

SYSLVL
This field is not used by any currently supported network.

TXTHDR
This field is not used by any currently supported network.

CMDOUT
The network command output file containing the network's responses to the command input file.

MSGROUT
The name of the program that processes messages from the network.

The Expedite parameter for network profiles is INBIMSG.

SEQNUM
A sequential number assigned to all outbound documents.

NETACKFILE
The name of a file (z/OS ddname) where you would like the network to write network acknowledgments when you request a status update. The network acknowledgments are read and evaluated by the message handler program.

NETPHONE
The phone number to dial to connect to your network.

SCRIPT
The script to be used by your communications software when processing service requests. This script would be part of your communication software package and not part of WebSphere Data Interchange.

FILLER
Reserved.

DESCRIPT
The description of this profile member.

LOGLOCK
The profile member logical lock flag (for WebSphere Data Interchange use only).

LASTUID
The user ID of the last person to update this profile member.

LASTUDT
The date and time that this profile member was last updated.

Mailbox (Requestor) Profile Block (REQDB)

Table 236 describes the mailbox (requestor) profile block (REQDB). The offset values in this table are relative to **0**. If this control block is used in network commands (NETOP) profile, you should add **1** to the offset value, because the offsets used for the Network Commands (NETOP) profile are relative to **1**.

Table 236. Definition of the Requestor Profile Block

Name	Offset	Length	Type	Description
BLKLEN	0	2	Bin	Length of REQDB
RESERV1	2	2	Bin	Reserved
BLKNME	4	8	Char	Block name
REQID	12	16	Char	Requestor ID
NETID	28	8	Char	Network ID
ACCTNO	36	32	Char	Network account number
USERID	68	32	Char	Network user ID
PASSWD	100	16	Char	Network password
MSGUCL	116	8	Char	Network message user class
INDDNAME	124	8	Char	Receive file name
NETCLS	132	1	Char	Network message class
NETCHG	133	1	Char	Network charge code
NETACK	134	1	Char	Network acknowledgment
NETVCHK	135	1	Char	Destination verification
NETRETN	136	3	Char	Mailbox retention period
NETEDIO	139	1	Char	EDI receive option
NETEDIP	140	1	Char	EDI processing override
STGFRMTO	141	1	Char	Storage format override
STGFRMT	142	1	Char	Storage format
NETCMDMBR	143	8	Char	Net commands PDS member
TIMEOUT	151	4	Char	Communication line timeout value
NTACKPGM	155	8	Char	Remote Network Acknowledgments processing program
ALTNETPHONE	163	32	Char	Alternate data line phone
COMPRESS	195	1	Char	Compression
PRIORITY	196	1	Char	Delivery priority
FILLER	197	15	Char	Reserved for WebSphere Data Interchange
DESCRIPT	213	30	Char	Member description
LOGLOCK	243	1	Char	Logical lock flag

Table 236. Definition of the Requestor Profile Block (continued)

Name	Offset	Length	Type	Description
LASTUID	244	17	Char	User ID that performed the latest update
LASTUDT	261	4	Bin	Date and time of the latest update

REQDB field descriptions

BLKLEN
A 2-byte binary field that contains the length of the REQDB data block. A REQDB is 264 bytes.

RESERV1
Reserved.

BLKNME
The name of the mailbox (requestor) profile block. **EDIREQDB**.

REQID
The name (key) used to refer to this requestor.

NETID
The name (key) that identifies the network. The value in this field must match the name of a member in the network profile.

ACCTNO
The account number that the network assigns to the requestor. The entry must be left-justified. For sending and receiving EDI standard transactions using ISA/IEA enveloping, the last position must be blank.

USERID
The user ID that the network assigns to the requestor. The entry must be left-justified.

PASSWD
The requestor's password for using the network. When connecting to IBM services, the first 8 characters are the current password, and the second 8 characters are the new password (used for changing the password).

MSGUCL
The message user class. A user-defined code that trading partners agree to use for identifying classes of information to be sent or received. Examples of classes are DEPT01, X12, MSG, FILE, EDI, and UCS. Leave this field blank to indicate that all information for the mailbox is to be sent or received.

INDDNAME

The name of the file into which information is received from the network. The translator processes EDI standard transactions from this file.

NETCLS

Indicates the status of the data being sent. Applies only to sending. Valid values are:

(blank) Normal status
T Test status

If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used.

The Expedite parameter for network profiles is MODE.

NETCHG

Indicates how charges are shared between sender and receiver. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

1 Receiver pays all charges.
2 Receiver pays all charges if agreed to, or charges are split between sender and receiver.
3 Receiver pays all charges if agreed to, or charges are split between sender and receiver if agreed to. Otherwise, the sender pays all charges (default).
4 Charges are split between sender and receiver if agreed to. Otherwise, the sender pays all charges.
5 Charges are split between sender and receiver.
6 Sender pays all charges.

The Expedite parameter for network profiles is CHARGE.

NETACK

Indicates which network acknowledgments (receipt, delivery, purge) you want to receive when sending to this trading partner. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

(blank) No acknowledgments
R Receipt only
D Delivery only
B Both receipt and delivery
A Purge only
C Both receipt and purge
E Either receipt or purge
F Receipt and either delivery or purge

The Expedite parameter for network profiles is ACK.

NETVCHK

Indicates whether the destination is verified before sending occurs. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

N No verification (default)
Y Verification required
F Attempt to verify, but send even if the destination is not verified (useful for intersystem addressing)

The Expedite parameter for network profiles is VERIFY.

NETRETN

The number of days that data is to be kept in the network mailbox before it is purged, if it is not received. Enter blanks or zeroes to use the default number. For Expedite Base/z/OS, the valid range is **001-180**. For Expedite/CICS, the valid range is **01-99**, left-justified. If your request specifies a trading partner, the value is taken from the trading partner profile, and the value in this field is not used.

NETEDIO

A value that indicates whether you want to store EDI segments in the receiving file as separate records. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

Y Ends records at the segment delimiter (default)
N Does not end records at the segment delimiter

The Expedite parameter is EDIOPT.

NETEDIP

Indicates whether EDI data you receive has special EDI processing (breaking records by the segment delimiter). If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

Y Performs EDI processing if the common data header indicates that the data is in EDI standard format (default)
N Omits EDI processing, regardless of the common data header

The Expedite parameter for network profiles is AUTOEDI.

STGFRMTO

Indicates whether you want to use the storage format defined in the common data header. If there is no common data header, the format indicated in the storage format field is used. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

Y Uses the storage format as defined in the common data header (default)
N Ignores the storage format defined in the common data header

The Expedite parameter for network profiles is DLMOVERRIDE.

STGFRMT
Indicates to the network how data is stored for free-form messages and files.

Valid values for several IBM products are listed below. Contact a representative of each network you are using for all available values.

For Expedite Base/z/OS (IEBASE), valid values are:

C   Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Program source code defined with variable length records is the type of file generally sent with this option. Output records do not include the carriage return and line-feed characters.

L   Precedes each record with a 2-byte hexadecimal record length. Select this option when sending data in fixed format or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.

N   Stores data as it is received. Output records are built based on the record length of the data set allocated to receive the data.

For Expedite/CICS, valid values are:

A   Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Program source code defined with variable length records is the type of file generally sent with this option. Output records do not include the carriage return and line-feed characters.

L   Precedes each record with a 2-byte hexadecimal record length. Select this option when sending a data set defined in fixed format or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.

O   Other (free-form).

The Expedite parameter for network profiles is DELIMITED.

NETCMDMBR
The name of a PDS member that will be allocated to the ddname of EDINTCMD. This member will contain the commands that you want to pass to a network. WebSphere Data Interchange reads the commands from the PDS member and writes them to the `Network input file` specified in the network profile member after all substitutable variable tags have been resolved by WebSphere Data Interchange.

TIMEOUT
The maximum allowable time that the data line for communications can be idle without being dropped. If you specify a trading partner when requesting network activity (send or receive), the value for this field is taken from the trading partner profile. Otherwise, the value for this field is taken from the mailbox (requestor) profile.

NTACKPGM

The name of a program that will be used to process network acknowledgments from a secondary network. This program is used only if you are using a gateway as your primary network to connect to a secondary network, and have requested and received network acknowledgments (into the `Network acknowledgment file`) from the secondary network.

Note: When a gateway is used to connect to another VAN, the gateway is referred to as the primary network because it is the network with which WebSphere Data Interchange interfaces. The other VAN is referred to as the secondary or remote network because WebSphere Data Interchange goes through the gateway to reach the other network. When the gateway is used to connect directly to a trading partner's site or when the gateway is used as the only network, there is no secondary network.

ALTNETPHONE

The alternate phone number to dial to connect to your network.

COMPRESS

Indicates to Expedite Base/z/OS whether to compress the data. Applies only to sending. Valid values are:

Y   Compresses the data.
N   Does not compress the data.
T   Uses the Expedite Base/z/OS compression table to decide whether to compress the data.

PRIORITY

Indicates to Expedite Base/z/OS and Expedite/CICS how to prioritize message delivery. Valid values are:

(blank) Normal delivery
P   High priority

FILLER

Reserved for WebSphere Data Interchange.

DESCRIPT

The description of this profile member.

LOGLOCK

The profile member logical lock flag (for WebSphere Data Interchange use only).

LASTUID

The user ID of the last person to update this profile member.

LASTUDT

The date and time that this profile member was last updated.

Appendix B. Sample programs

This appendix provides information about sample programs and exit routines. Many of these are provided with the product, and reference is given to where you can find the supplied code. For other samples, the code is illustrated within this appendix.

Creating tagged import files from fixed format files

Sample programs included in the product enable you to convert a fixed format (flat) file into a tagged import file for importing trading partner profile (TPPROF) members, send transaction usages, and receive transaction usages.

These sample programs and JCL include extensive documentation that describes how they work and how they can be used.

These programs are:
- A COBOL program named EDIXF2T, which is the main program executed.
- An Assembler program named EDIXTAGF, which is link edited with EDIXTAGF, and is a formatting service to create the tags.
- Sample JCL named EDIXF2T used to define the fixed flat file and to execute the conversion program EDIXF2T.

These programs and JCL are provided as source and can be modified, compiled, and link edited to suit your individual needs.

Initializing and terminating WebSphere Data Interchange

Sample programs for initialization and termination of WebSphere Data Interchange are provided for COBOL, PL/I, and C.

COBOL initialization/termination example

A COBOL program that illustrates how to initialize and terminate WebSphere Data Interchange using the API is provided with the product. The library member name for this program is `EDI.V3R2M0.SEDICBL2(FXXCOBA)`.

PL/I initialization/termination example

A PL/I program that illustrates how to initialize and terminate WebSphere Data Interchange using the API is provided with the product. The library member name for this program is `EDI.V3R2M0.SEDIPLI2(FXXPLIA)`.

C initialization/termination example

The following C program illustrates how to initialize and terminate WebSphere Data Interchange using the API within subroutines `DIinit` and `DIterm`.

```
/* ---------------------------------------------------------------- */
/* Get the control block definitions                                */
/* ---------------------------------------------------------------- */
```

Initializing and terminating WebSphere Data Interchange

```c
#include "stdio.h"              /* C I/O routines              */
#include "disnb.h"              /* get SNB definition          */
#include "diccb.h"              /* get CCB definition          */
#include "difcb.h"              /* get FCB definition          */
/* ------------------------------------------------------------------ */
/* Prototypes for internal functions                                  */
/* ------------------------------------------------------------------ */
   static int check_ierr(ccb*);
   static int check_terr(ccb*);
   int
main()
 {
   ccb   DIccb;           /* WebSphere Data Interchange common block  */
   /* ------------------------------------------------------------------ */
   /* Call a routine to Initialize WebSphere Data Interchange         */
   /* ------------------------------------------------------------------ */
   if (!DIinit(&DIccb)) {
       /* ------------------------------------------------------------- */
       /* Application logic goes here                                  */
       /* ------------------------------------------------------------- */
       /* ------------------------------------------------------------- */
       /* Call a routine to Terminate WebSphere Data Interchange      */
       /* ------------------------------------------------------------- */
       DIterm(&DIccb);
   }
/* ------------------------------------------------------------------ */
/* A subroutine to initialize WebSphere Data Interchange              */
/* ------------------------------------------------------------------ */
   int
DIinit(CCBptr)
   ccb  *CCBptr;                /* Pointer to Common Control Block    */
 {
   snb  SNB;                    /* Local snb                          */
   fcb  FUNCBLK;                /* Local fcb                          */
   /* ------------------------------------------------------------- */
   /* Initialize the CCB                                          */
   /*  1. Clear it                                                */
   /*  2. Set CCB length                                          */
   /*  3. Move in language indicator                              */
   /* ------------------------------------------------------------- */
   memset(CCBptr,'0',sizeof(ccb));
   CCBptr->zccbll=sizeof(ccb);
   memcpy(CCBptr->zccblpid,"ENU   ",6);
   /* ------------------------------------------------------------- */
   /* Initialize the SNB for environmental services              */
   /* 1. Clear it                                                 */
   /* 2. Set SNB length                                           */
   /* 3. Set the number of parameters that will be passed        */
   /* 4. Set the name of the SERVICE being called                */
   /* ------------------------------------------------------------- */
   memset(&SNB,'0',sizeof(SNB));
   SNB.zsnbll=sizeof(SNB);
   SNB.zsnbpc=5;
   memcpy(SNB.zsnbname,"ENVSERV ",8);
   /* ------------------------------------------------------------- */
```

```
   /* Initialize the FUNCBLK for the function wanted               */
   /* 1. Set the length of the function block                      */
   /* 2. Set the function value to 1 indicating INITIALIZATION     */
   /* ----------------------------------------------------------- */
   FUNCBLK.zfcbll=sizeof(fcb);
   FUNCBLK.zfcbfunc=1;
   /* ----------------------------------------------------------- */
   /* Make the call to FXXZC to initialize the ENVIRONMENT and     */
   /*  check for any errors after the call.  The fourth parameter to */
   /*  FXXZC must be changed to the APPLICATION NAME.  This        */
   /*  determines which activity log the transactions and/or error  */
   /*  messages are logged to.                                     */
   /* ----------------------------------------------------------- */
   printf("INITIALIZE - SET UP WebSphere Data Interchange ENVIRONMENTn");
   fxxzc(&SNB,CCBptr,&FUNCBLK,"APPLNAME","SYSID");
   return check_ierr(CCBptr);
 }
/* ----------------------------------------------------------- */
/* A subroutine to terminate WebSphere Data Interchange        */
/* ----------------------------------------------------------- */
   int
DIterm(CCBptr)
   ccb  *CCBptr;          /* Pointer to Common Control Block    */
 {
   snb  SNB;                        /* Local snb                 */
   fcb  FUNCBLK;                    /* Local fcb                 */
   /* ----------------------------------------------------------- */
   /* Initialize the SNB for environmental services               */
   /* 1. Clear it                                                 */
   /* 2. Set SNB length                                           */
   /* 3. Set the number of parameters that will be passed         */
   /* 4. Set the name of the SERVICE being called                 */
   /* ----------------------------------------------------------- */
   memset(&SNB,'0',sizeof(SNB));
   SNB.zsnbll=sizeof(SNB);
   SNB.zsnbpc=3;
   memcpy(SNB.zsnbname,"ENVSERV ",8);
   /* ----------------------------------------------------------- */
   /* Initialize the FUNCBLK for the function wanted              */
   /* 1. Set the length of the function block                     */
   /* 2. Set the function value to 2 indicating TERMINATION       */
   /* ----------------------------------------------------------- */
   FUNCBLK.zfcbll=sizeof(fcb);
   FUNCBLK.zfcbfunc=2;
   /* ----------------------------------------------------------- */
   /* Make the call to FXXZC to terminate the ENVIRONMENT and     */
   /*  check for any errors after the call.                       */
   /* ----------------------------------------------------------- */
   printf("TERMINATE the WebSphere Data Interchange ENVIRONMENTn");
   fxxzc(&SNB,CCBptr,&FUNCBLK);
   return check_terr(CCBptr);
 }
/* ----------------------------------------------------------- */
/* A subroutine to check the results of a terminate            */
/* ----------------------------------------------------------- */
```

```
      int
check_terr(CCBptr)
   ccb *CCBptr;              /* Pointer to CCB
control block        */
 {
    int   lrc;              /* Local return value              */
    /* ---------------------------------------------------------- */
    /* Check Return Codes received from TERMINATION               */
    /*  return 1 if there are errors                              */
    /*  return 0 if there are NO errors                           */
    /* ---------------------------------------------------------- */
    lrc=1;
    switch ((int)CCBptr->zccbrc) {
       case 0:
          /* ------------------------------------------------------ */
          /* ZERO indicates termination was successful              */
          /* ------------------------------------------------------ */
          lrc=0;
          printf("WebSphere Data Interchange TERMINATION SUCCESSFULn");
          break;
        default:
          /* ------------------------------------------------------ */
          /* Invalid return code.  Display the return codes and     */
          /* terminate processing                                   */
          /* ------------------------------------------------------ */
          printf("INVALID RETURN CODE FROM WebSphere Data Interchangen");
          printf("RETURN CODE = %ld, EXTENDED RETURN CODE = %ldn",
                 CCBptr->zccbrc,CCBptr->zccberc);
          break;
     }
   return lrc;
 }
/* ------------------------------------------------------------------- */
/* A subroutine to check the results of an initialize                  */
/* ------------------------------------------------------------------- */
  int
check_ierr(CCBptr)
  ccb *CCBptr;                /* Pointer to CCB control block     */
{
    int   lrc;               /* Local return value               */
    /* ---------------------------------------------------------- */
    /* Check Return Codes received from Initialization            */
    /*  return 1 if there are errors                              */
    /*  return 0 if there are NO errors                           */
    /* ---------------------------------------------------------- */
    lrc=1;
    switch ((int)CCBptr->zccbrc) {
       case 0:
          /* ------------------------------------------------------ */
          /* ZERO indicates initialization was successful and       */
          /* regular processing can continue                        */
          /* ------------------------------------------------------ */
          lrc=0;
          printf("WebSphere Data Interchange INITIALIZATION SUCCESSFULn");
          break;
```

```
        case 4:
           /* -------------------------------------------------------- */
           /* FOUR indicates initialization was not successful.        */
           /* Display the return code and extended return code and     */
           /* terminate processing.                                    */
           /* -------------------------------------------------------- */
           printf("WebSphere Data Interchange INITIALIZATION FAILEDn");
           printf("RETURN CODE = %ld, EXTENDED RETURN CODE = %ldn",
                   CCBptr->zccbrc,CCBptr->zccberc);
           break;
         default:
           /* -------------------------------------------------------- */
           /* Invalid return code.  Display the return codes and       */
           /* terminate processing                                     */
           /* -------------------------------------------------------- */
           printf("INVALID RETURN CODE FROM WebSphere Data Interchangen");
           printf("RETURN CODE = %ld, EXTENDED RETURN CODE = %ldn",
                   CCBptr->zccbrc,CCBptr->zccberc);
           break;
    }
  return lrc;
}
```

Querying the Transaction Store

Sample programs for querying the WebSphere Data Interchange Transaction Store are provided for COBOL and PL/I.

Querying the Transaction Store using COBOL

A sample COBOL CICS program shows how to invoke the Utility to query the Transaction Store. The program library member name is `EDI.V3R2M0.SEDICBL2(FXXCOBU)`

Querying the Transaction Store using PL/I

A sample PL/I CICS program shows how to invoke the Utility to query the Transaction Store. The program library member name is `EDI.V3R2M0.SEDIPLI2(FXXPLIU)`

Translating and queueing for send using C

This C program shows how to translate application data to EDI format and envelopes the EDI data for subsequent sending to a trading partner using the API. It uses the `TRterm` subroutine which is in the End Translation sample (see "Ending translation using C" on page 723).

```
/* ------------------------------------------------------------------ */
/* Include the definition files                                       */
/* ------------------------------------------------------------------ */
#include "stdio.h"              /* C I/O definitions                  */
#include "disnb.h"              /* SNB definition                     */
#include "diccb.h"              /* CCB definition                     */
#include "difcb.h"              /* FCB definition                     */
#include "ditrcb.h"             /* TRCB definition                    */
```

Translating and queueing for send using C

```
#include "didblk.h"              /* TRIDB and TRODB definitions      */

/* ----------------------------------------------------------------- */
/* Prototype internal functions                                      */
/* ----------------------------------------------------------------- */
   static int check_error(ccb*);
   int
main()
 {
   snb   TRsnb;                   /* SNB for translator               */
   ccb   DIccb;                   /* WDI common blk                   */
   fcb   TRfcb;                   /* Translator function block        */
   TRcb  TPCB;                    /* Translator control block         */
   DATAblk *TPIDB,*TPODB;         /* Pointers to data blocks          */
   /* -------------------------------------------------------------- */
   /* Call a routine to Initialize WebSphere Data Interchange        */
   /* -------------------------------------------------------------- */
   if (!DIinit(&DIccb)) {
       /* ---------------------------------------------------------- */
       /* Initialize was successful so continue processing           */
       /* ---------------------------------------------------------- */
       /* ---------------------------------------------------------- */
       /* Prepare the translator SNB                                 */
       /* 1.  Initialize to zeros                                    */
       /* 2.  Set the SNB length                                     */
       /* 3.  Set the number parameters passed to the translator     */
       /* 4.  Set the name of the translator service (TRANPROC)      */
       /* ---------------------------------------------------------- */
       memset(&TRsnb,'0',sizeof(TRsnb));
       TRsnb.zsnbll=sizeof(snb);
       TRsnb.zsnbpc=6;
       memcpy(TRsnb.zsnbname,"TRANPROC",8);
       /* ---------------------------------------------------------- */
       /* Prepare the translator FCB                                 */
       /* 1. Set the FCB length                                      */
       /* 2. Set the function for PRODUCTION SEND TRANSLATE (131)     */
       /* ---------------------------------------------------------- */
       TRfcb.zfcbll=sizeof(fcb);
       TRfcb.zfcbfunc=131;
       /* ---------------------------------------------------------- */
       /* Prepare the translator control block                       */
       /* ---------------------------------------------------------- */
       memset(&TPCB,' ',sizeof(TPCB));
       TPCB.blklen = sizeof(TPCB);
       memcpy(TPCB.blknme,"EDITRCB ",8);
       /* ---------------------------------------------------------- */
       /* set ATFID to data format ID that was defined using the     */
       /* Data Format screens.  This format describes the  */
       /* structure of your Application data.                        */
       /* ---------------------------------------------------------- */
       memcpy(TPCB.atfid,"POSEND",6);
       /* ---------------------------------------------------------- */
       /* Set the internal trading partner ID for this trading partner */
       /* that was defined using Trading Partner transaction screens */
       /* ---------------------------------------------------------- */
```

```
        memcpy(TPCB.intpid,"XYZCOMPANYINTPID",16);
        /* ----------------------------------------------------------- */
        /* Prepare the INPUT and OUTPUT data blocks                    */
        /* ----------------------------------------------------------- */
        TPIDB = (DATAblk*) malloc(32767);
        TPIDB->blklen = 32767;
        memcpy(TPIDB->blknme,"EDITRIN ",8);
            /* MOVE APPLICATION DATA TO TPIDB.data           */
            /* MOVE APPLICATION DATA length to TPIDB.datlen *
        TPODB = (DATAblk*) malloc(32767);
        TPODB->blklen = 32767;
        memcpy(TPODB->blknme,"EDITROUT",8);
        /* ----------------------------------------------------------- */
        /* Call the translator to TRANSLATE application data           */
        /* ----------------------------------------------------------- */
        printf("TRANSLATE - TRANSLATE AND QUEUE TO SENDn");
        fxxzc(&TRsnb,&DIccb,&TRfcb,&TPCB,TPIDB,TPODB);
        if (!check_error(&DIccb)) {
            /* --------------------------------------------------------- */
            /* There were no errors, call translator with a TERMINATION  */
            /* request to finish building and to queue the transaction   */
            /* data                                                      */
            /* --------------------------------------------------------- */
            TRterm(&TRsnb,&DIccb,&TRfcb,&TPCB,TPIDB,TPODB);
        }
        /* ----------------------------------------------------------- */
        /* Terminate WebSphere Data Interchange                        */
        /* ----------------------------------------------------------- */
        DIterm(&DIccb);
    }
  }
    static int
check_error(CCBptr)
    ccb   *CCBptr;           /* Pointer to common block                   */
  {
    int    lrc;             /* return code                               */
    /* --------------------------------------------------------------- */
    /* Check the return codes from a SEND TRANSLATE request            */
    /* --------------------------------------------------------------- */
    lrc=0;
    switch ((int)CCBptr->zccbrc) {
        case 0:
            /* ------------------------------------------------------- */
            /* Return code is zero, the translator was successful      */
            /* ------------------------------------------------------- */
            printf("Translation was successfuln");
            break;
         case 8:
            /* ------------------------------------------------------- */
            /* If the return code is EIGHT and the Extended Return Code*/
            /* is from ONE to SIX, this indicates a Translation Error. */
            /* If the Extended Return Code is greater than or equal to */
            /* TEN, a non-translation error has occurred and           */
            /* translator will have terminated itself automatically    */
            /* ------------------------------------------------------- */
```

Translating and queueing for send using C

```c
            switch ((int)CCBptr->zccberc) {
               case 1:              /* data element error              */
                  printf("Data element level error occurredn");
                  break;
               case 2:              /* segment level error             */
                  printf("Segment level occurredn");
                  break;
               case 3:              /* transaction level error         */
               case 4:              /* function group level error      */
               case 5:              /* envelope level error            */
               case 6:              /* invalid data in input file      */
                  printf("Unexpected error during send processingn");
                  printf("Return code = %ld, extended return code = %ld",
                         CCBptr->zccbrc,CCBptr->zccberc);
                  break;
               default:
                  lrc = 1;
                  printf("Non translation error occurredn");
                  printf("Return code = %ld, extended return code = %ld",
                         CCBptr->zccbrc,CCBptr->zccberc);
                  break;
            }
      case 12:
            /* ------------------------------------------------------- */
            /* A severe error occurred                                 */
            /* ------------------------------------------------------- */
            lrc = 1;
            printf("A severe error occurred in translationn");
            printf("Return code = %ld, extended return code = %ld",
                   CCBptr->zccbrc,CCBptr->zccberc);
            break;
      default:
            /* ------------------------------------------------------- */
            /* An invalid return code was returned                     */
            /* ------------------------------------------------------- */
            lrc = 1;
            printf("An invalid return code from the translatorn");
            printf("Return code = %ld, extended return code = %ld",
                   CCBptr->zccbrc,CCBptr->zccberc);
            break;
   }
   return lrc;
}
```

Sending queued data using C

This C program shows how to send previously-translated-and-enveloped data to a trading partner using the API.

```c
/* --------------------------------------------------------------------- */
/* Get the control block definitions                                     */
/* --------------------------------------------------------------------- */
#include "stdio.h"                  /* C I/O library                      */
#include "disnb.h"                  /* SNB definition                     */
#include "diccb.h"                  /* CCB definition                     */
```

```
#include "difcb.h"                 /* FCB definition                      */
#include "dicmcb.h"                /* TRCB definition                     */
#include "ditpdb.h"                /* TPPDB definition                    */
/* ------------------------------------------------------------------ */
/* Provide function prototypes                                         */
/* ------------------------------------------------------------------ */
   static int check_error(ccb*);
   int
main()
 {
   ccb    DIccb;         /* Common Control Block                       */
   snb    CMsnb;         /* SNB for communications                     */
   fcb    CMfcb;         /* FCB for communications                     */
   CMcb   CMCB;          /* Communication control block                */
   TPPdb  CMTPPDB;       /* Trading Partner Data Block                 */
   /* ------------------------------------------------------------------ */
   /* Call function for WebSphere Data Interchange initialization       */
   /* ------------------------------------------------------------------ */
   if (DIinit(&DIccb))
      return 1;
   /* ------------------------------------------------------------------ */
   /* Initialize the SNB for communications support                     */
   /* 1. Initialize the block to zeros                                  */
   /* 2. Set block length                                               */
   /* 3. Set the number of parameters to communications                */
   /* 4. Set the name of the communications service (COMM)              */
   /* ------------------------------------------------------------------ */
   memset(&CMsnb,'0',sizeof(snb));
   CMsnb.zsnbll=sizeof(snb);
   CMsnb.zsnbpc=6;
   memcpy(CMsnb.zsnbname,"COMM    ",8);
   /* ------------------------------------------------------------------ */
   /* Initialize the function block for communications                  */
   /* 1. Set the length of the block                                    */
   /* 2. Set the function to SEND TRANSACTION DATA                      */
   /* ------------------------------------------------------------------ */
   CMfcb.zfcbll=sizeof(fcb);
   CMfcb.zfcbfunc=211;
   /* ------------------------------------------------------------------ */
   /* Initialize the control block for communications                   */
   /* ------------------------------------------------------------------ */
   memset(&CMCB,' ',sizeof(CMCB));
   CMCB.blklen=sizeof(CMCB);
   memcpy(CMCB.blknme,"EDICMCB",7);
   /* ------------------------------------------------------------------ */
   /* Set the network operation (netop) to indicate that we want to     */
   /* send X12 data (SENDX12)                                           */
   /* ------------------------------------------------------------------ */
   memcpy(CMCB.netop,"SENDX12",7);
   /* ------------------------------------------------------------------ */
   /* Set the requestor id (reqid) equal to the member ID of an         */
   /* entry in the REQUESTOR profile.                                   */
   /* ------------------------------------------------------------------ */
   memcpy(CMCB.reqid,"XYZCOMPANYPOSEND",16);
   /* ------------------------------------------------------------------ */
```

Sending queued data using C

```c
   /* NULLS in the ename field indicates the message user class from */
   /* the REQUESTOR profile will be used                             */
   /* -------------------------------------------------------------- */
   memset(CMCB.ename,'0',sizeof(CMCB.ename));
   /* -------------------------------------------------------------- */
   /* Initialize the trading partner Data block                      */
   /* -------------------------------------------------------------- */
   memset(&CMTPPDB,' ',sizeof(CMTPPDB));
   CMTPPDB.blklen=sizeof(CMTPPDB);
   memcpy(CMTPPDB.blknme,"EDITPPDB",8);
   /* -------------------------------------------------------------- */
   /* Issue the call to send the X12 data                            */
   /* -------------------------------------------------------------- */
   printf("SEND TRANSACTION DATAn");
   fxxzc(&CMsnb,&DIccb,&CMfcb,&CMCB,&CMTPPDB,(void*)0);
   if (check_error(&DIccb)) {
      /* ----------------------------------------------------------- */
      /* Add code here to process a failed SEND request              */
      /* ----------------------------------------------------------- */
   }
   /* -------------------------------------------------------------- */
   /* Terminate WebSphere Data Interchange                           */
   /* -------------------------------------------------------------- */
   DIterm(&DIccb);
   return 0;
 }
   static int
check_error(CCBptr)
   ccb   *CCBptr;      /* Pointer to the common block                */
 {
 int  lrc;            /* return code                                 */
   lrc=1;
   /* -------------------------------------------------------------- */
   /* Process according to the return code in the CCB                */
   /* -------------------------------------------------------------- */
   switch ((int)CCBptr->zccbrc) {
      case 0:
         /* ----------------------------------------------------------- */
         /* Data was successfully sent                                  */
         /* ----------------------------------------------------------- */
         printf("Data successfully sent to the networkn");
         lrc=0;
         break;
      case 4:
         /* ----------------------------------------------------------- */
         /* Warning from communications                                 */
         /* ----------------------------------------------------------- */
         printf("A warning was received when SENDING datan");
         printf("Return code = %ld, extended return code = %ldn",
                CCBptr->zccbrc,CCBptr->zccberc);
         lrc=0;
         break;
      case 8:
      case 12:
         /* ----------------------------------------------------------- */
```

```
          /* An error was returned by communications                 */
          /* ------------------------------------------------------- */
          printf("An error was received when SENDING datan");
          printf("Return code = %ld, extended return code = %ldn",
                  CCBptr->zccbrc,CCBptr->zccberc);
          break;
      default:
          /* ------------------------------------------------------- */
          /* An invalid return code from communications              */
          /* ------------------------------------------------------- */
          printf("An invalid return code when SENDING datan");
          printf("Return code = %ld, extended return code = %ldn",
                  CCBptr->zccbrc,CCBptr->zccberc);
          break;
   }
   return lrc;
 }
```

Ending translation using C

This C program shows how to end translation of application data and envelope it for subsequent sending to a trading partner using the API.

```
/* ------------------------------------------------------------------ */
/* Get control block definitions                                      */
/* ------------------------------------------------------------------ */
#include "stdio.h"              /* C I/O library                      */
#include "disnb.h"              /* SNB definition                     */
#include "diccb.h"              /* CCB definition                     */
#include "difcb.h"              /* FCB definition                     */
#include "ditrcb.h"             /* TRCB definition                    */
#include "didblk.h"             /* TRIDB and TRODB definitions        */


/* ------------------------------------------------------------------ */
/* Prototype for internal function                                    */
/* ------------------------------------------------------------------ */
   static void check_error(ccb*);
   int
 TRterm(SNBptr,CCBptr,TPCBptr,TPIDB,TPODB)
   snb    *SNBptr;        /* Pointer to SNB for translator        */
   ccb    *CCBptr;        /* Pointer to common block              */
   TRcb   *TPCBptr;       /* Pointer to translator control block  */
   DATAblk *TPIDB;        /* Pointer to input data block          */
   DATAblk *TPODB;        /* Pointer to output data block         */
 {
   fcb    TRfcb;          /* function block for translator termination */
   /* --------------------------------------------------------------- */
   /* Set up the function block for TERMINATION of the translator     */
   /* --------------------------------------------------------------- */
   TRfcb.zfcbll = sizeof(fcb);
   TRfcb.zfcbfunc = 1000;
   /* --------------------------------------------------------------- */
   /* Make the call to terminate translation                          */
   /* --------------------------------------------------------------- */
   printf("END_TRANSLATOR - Terminate call to translatorn");
```

Ending translation using C

```
      fxxzc(SNBptr,CCBptr,&TRfcb,TPCBptr,TPIDB,TPODB);
      check_error(CCBptr);
   }
      static void
   check_error(CCBptr)
      ccb    *CCBptr;          /* Pointer to common block                  */
   {
      switch ((int)CCBptr->zccbrc) {
         case 0:
            /* -------------------------------------------------------- */
            /* Termination of the translator was successful             */
            /* -------------------------------------------------------- */
            printf("TRANSACTION PROCESSOR TERMINATED SUCCESSFULLYn");
            break;
         case 8:
         case 12:
            /* -------------------------------------------------------- */
            /* Termination failed                                       */
            /* -------------------------------------------------------- */
            printf("TRANSACTION PROCESSOR TERMINATION FAILEDn");
            printf("Return code = %ld , extended return code = %ldn",
                   CCBptr->zccbrc,CCBptr->zccbrc);
            break;
         default:
            /* -------------------------------------------------------- */
            /* Invalid return code                                      */
            /* -------------------------------------------------------- */
            printf("Invalid return code from TERMINATION calln");
            printf("Return code = %ld , extended return code = %ldn",
                   CCBptr->zccbrc,CCBptr->zccbrc);
            break;
      }
   }
```

Receiving data from a network using C

This C program shows how to receive data from a network using the API.

```
/* ------------------------------------------------------------------- */
/* Get the control block definitions                                   */
/* ------------------------------------------------------------------- */
#include "stdio.h"                /* C I/O library                     */
#include "disnb.h"                /* SNB definition                    */
#include "diccb.h"                /* CCB definition                    */
#include "difcb.h"                /* FCB definition                    */
#include "dicmcb.h"               /* TRCB definition                   */
#include "ditpdb.h"               /* TPPDB definition                  */
/* ------------------------------------------------------------------- */
/* Prologues for internal functions                                    */
/* ------------------------------------------------------------------- */
   static int check_error(ccb*);
   int
main()
 {
   ccb    DIccb;         /* Common Control Block                        */
```

```
snb   CMsnb;        /* SNB for communications                    */
fcb   CMfcb;        /* FCB for communications                    */
CMcb  CMCB;         /* Communication control block               */
TPPdb CMTPPDB;      /* Trading Partner Data Block                */
/* ---------------------------------------------------------------- */
/* Call function for WebSphere Data Interchange initialization    */
/* ---------------------------------------------------------------- */
if (DIinit(&DIccb))
   return 1;
/* ---------------------------------------------------------------- */
/* Initialize the SNB for communications support                  */
/* 1. Initialize the block to zeros                               */
/* 2. Set block length                                           */
/* 3. Set the number of parameters to communications            */
/* 4. Set the name of the communications service (COMM)         */
/* ---------------------------------------------------------------- */
memset(&CMsnb,'0',sizeof(snb));
CMsnb.zsnbll=sizeof(snb);
CMsnb.zsnbpc=6;
memcpy(CMsnb.zsnbname,"COMM    ",8);
/* ---------------------------------------------------------------- */
/* Initialize the function block for communications               */
/* 1. Set the length of the block                                */
/* 2. Set the function to RECEIVE                                */
/* ---------------------------------------------------------------- */
CMfcb.zfcbll=sizeof(fcb);
CMfcb.zfcbfunc=232;
/* ---------------------------------------------------------------- */
/* Initialize the control block for communications                */
/* ---------------------------------------------------------------- */
memset(&CMCB,' ',sizeof(CMCB));
CMCB.blklen=sizeof(CMCB);
memcpy(CMCB.blknme,"EDICMCB",7);
/* ---------------------------------------------------------------- */
/* Set the network operation (netop) to indicate that we want to  */
/* receive X12 data (RECVX12)                                     */
/* ---------------------------------------------------------------- */
memcpy(CMCB.netop,"RECVX12",7);
/* ---------------------------------------------------------------- */
/* Set the requestor id (reqid) equal to the member ID of an      */
/* entry in the REQUESTOR profile.                               */
/* ---------------------------------------------------------------- */
memcpy(CMCB.reqid,"XYZCOMPANYPODEPT",16);
/* ---------------------------------------------------------------- */
/* acctyp of "D" means to receive for this trading partner only   */
/* ---------------------------------------------------------------- */
*CMCB.acctyp='D';
/* ---------------------------------------------------------------- */
/* datatyp  of "D" indicates a DDNAME is used rather than DSNAME  */
/* ---------------------------------------------------------------- */
*CMCB.datatyp='D';
/* ---------------------------------------------------------------- */
/* NULLS in the ename field indicates the message user class from */
/* the REQUESTOR profile will be used                            */
/* ---------------------------------------------------------------- */
```

Receiving data from a network using C

```c
      memset(CMCB.ename,'0',sizeof(CMCB.ename));
      /* --------------------------------------------------------------- */
      /* Set record length indicator (RESRECL) to indicate that the      */
      /* lrecl of the output file should be used as the record length    */
      /* --------------------------------------------------------------- */
      *CMCB.resrecl = 'S';
      /* --------------------------------------------------------------- */
      /* Initialize the trading partner Data block                       */
      /* --------------------------------------------------------------- */
      memset(&CMTPPDB,' ',sizeof(CMTPPDB));
      CMTPPDB.blklen=sizeof(CMTPPDB);
      memcpy(CMTPPDB.blknme,"EDITPPDB",8);
      /* --------------------------------------------------------------- */
      /* Issue the call to receive the X12 data                          */
      /* --------------------------------------------------------------- */
      printf("RECEIVE IMMEDIATEn");
      fxxzc(&CMsnb,&DIccb,&CMfcb,&CMCB,&CMTPPDB,(void*)0);
      if (!check_error(&DIccb)) {
         /* ------------------------------------------------------------ */
         /* Add code here to call translator to processed received data  */
         /* ------------------------------------------------------------ */
      }
      /* --------------------------------------------------------------- */
      /* Terminate WebSphere Data Interchange                            */
      /* --------------------------------------------------------------- */
      DIterm(&DIccb);
      return 0;
 }
   static int
check_error(CCBptr)
   ccb   *CCBptr;      /* Pointer to the common block                    */
 {
   int   lrc;         /* return code                                     */
   lrc=1;
   /* --------------------------------------------------------------- */
   /* Process according to the return code in the CCB                 */
   /* --------------------------------------------------------------- */
   switch ((int)CCBptr->zccbrc) {
      case 0:
         /* ------------------------------------------------------------ */
         /* Data was successfully received                               */
         /* ------------------------------------------------------------ */
         printf("Data successfully received from the networkn");
         lrc=0;
         break;
      case 4:
         /* ------------------------------------------------------------ */
         /* Warning from communications                                  */
         /* ------------------------------------------------------------ */
         printf("A warning was received when RECEIVING datan");
         printf("Return code = %ld, extended return code = %ldn",
                CCBptr->zccbrc,CCBptr->zccberc);
         lrc=0;
         break;
      case 8:
```

```
        case 12:
           /* ------------------------------------------------------------ */
           /* An error was returned by communications                      */
           /* ------------------------------------------------------------ */
           printf("An error was received when RECEIVING datan");
           printf("Return code = %ld, extended return code = %ldn",
                   CCBptr->zccbrc,CCBptr->zccberc);
           break;
        default:
           /* ------------------------------------------------------------ */
           /* An invalid return code from communications                   */
           /* ------------------------------------------------------------ */
           printf("An invalid return code when RECEIVING datan");
           printf("Return code = %ld, extended return code = %ldn",
                   CCBptr->zccbrc,CCBptr->zccberc);
           break;
    }
    return lrc;
 }
```

Translating received data using C

This C program shows how to deenvelope previously-received EDI data and translate it to application format using the API.

```
/* -------------------------------------------------------------------- */
/* Include the definition files                                         */
/* -------------------------------------------------------------------- */
#include "stdio.h"               /* C I/O definitions              */
#include "disnb.h"               /* SNB definition                 */
#include "diccb.h"               /* CCB definition                 */
#include "difcb.h"               /* FCB definition                 */
#include "ditrcb.h"              /* TRCB definition                */
#include "didblk.h"              /* TRIDB and TRODB definitions    */
/* -------------------------------------------------------------------- */
/* prototypes                                                           */
/* -------------------------------------------------------------------- */
   static int check_error(ccb*);
   int
main()
 {
   snb   TRsnb;                  /* SNB for translator             */
   ccb   DIccb;                  /* Common control block           */
   fcb   TRfcb;                  /* Translator function block      */
   TRcb  TPCB;                   /* Translator control block       */
   DATAblk *TPIDB,*TPODB;        /* Pointers to data blocks        */
   /* ---------------------------------------------------------------- */
   /* Call a routine to Initialize WebSphere Data Interchange         */
   /* ---------------------------------------------------------------- */
   if (!DIinit(&DIccb)) {
      /* ------------------------------------------------------------ */
      /* Initialize was successful so continue processing            */
      /* ------------------------------------------------------------ */
      /* ------------------------------------------------------------ */
      /* Prepare the translator SNB                                  */
```

Translating received data using C

```
/* 1.  Initialize to zeros                                  */
/* 2.  Set the SNB length                                   */
/* 3.  Set the number parameters passed to the translator   */
/* 4.  Set the name of the translator service (TRANPROC)    */
/* -------------------------------------------------------- */
memset(&TRsnb,'0',sizeof(TRsnb));
TRsnb.zsnbll=sizeof(snb);
TRsnb.zsnbpc=6;
memcpy(TRsnb.zsnbname,"TRANPROC",8);
/* -------------------------------------------------------- */
/* Prepare the translator FCB                               */
/* 1. Set the FCB length                                    */
/* 2. Set the function for PRODUCTION SEND TRANSLATE (131)   */
/* -------------------------------------------------------- */
TRfcb.zfcbll=sizeof(fcb);
TRfcb.zfcbfunc=212;
/* -------------------------------------------------------- */
/* Prepare the translator control block                     */
/* -------------------------------------------------------- */
memset(&TPCB,' ',sizeof(TPCB));
TPCB.blklen = sizeof(TPCB);
memcpy(TPCB.blknme,"EDITRCB ",8);
/* -------------------------------------------------------- */
/* set REQID to REQUESTOR profile that was defined using    */
/* profile screens.  The requestor profile contains the     */
/* DDNAME of the file that was received and is to be translated.*/
/* -------------------------------------------------------- */
memcpy(TPCB.reqid,"XYZCOMPANYPODEPT",16);
/* -------------------------------------------------------- */
/* Prepare the INPUT and OUTPUT data blocks                 */
/* -------------------------------------------------------- */
TPIDB = (DATAblk*) malloc(32767);
TPIDB->blklen = 32767;
memcpy(TPIDB->blknme,"EDITRIN ",8);
TPODB = (DATAblk*) malloc(32767);
TPODB->blklen = 32767;
memcpy(TPODB->blknme,"EDITROUT",8);
/* -------------------------------------------------------- */
/* Call the translator to TRANSLATE STANDARD data to APPL format*/
/* -------------------------------------------------------- */
for (;!DIccb.zccbrc;) {
   printf("TRANSLATE STANDARD DATA to APPLICATION FORMATn");
   fxxzc(&TRsnb,&DIccb,&TRfcb,&TPCB,TPIDB,TPODB);
   if (!check_error(&DIccb)) {
      /* -------------------------------------------------- */
      /* Add code here to process the application data       */
      /* placed in TRODB by the translator.                 */
      /* -------------------------------------------------- */
   }
}
/* -------------------------------------------------------- */
/* Terminate WebSphere Data Interchange                     */
/* -------------------------------------------------------- */
DIterm(&DIccb);
}
```

```
  }
    static int
check_error(CCBptr)
    ccb   *CCBptr;         /* Pointer to common block                 */
  {
    int    lrc;            /* return code                             */
    /* --------------------------------------------------------------- */
    /* Check the return codes from a RECEIVE TRANSLATE request         */
    /* --------------------------------------------------------------- */
    lrc=0;
    switch ((int)CCBptr->zccbrc) {
       case 0:
           /* ------------------------------------------------------- */
           /* Return code is zero, the translator was successful      */
           /* ------------------------------------------------------- */
           printf("Translation was successfuln");
           break;
        case 8:
           /* ------------------------------------------------------- */
           /* If the return code is EIGHT and the Extended Return Code*/
           /* is from ONE to SIX, this indicates a Translation Error. */
           /* If the Extended Return Code is greater than or equal to */
           /* TEN, a non-translation error has occurred and           */
           /* translator will have terminated itself automatically    */
           /* ------------------------------------------------------- */
           switch ((int)CCBptr->zccberc) {
              case 1:              /* data element error              */
                 printf("Data element level error occurredn");
                 CCBptr->zccbrc=CCBptr->zccberc=0;
                 break;
              case 2:              /* segment level error             */
                 printf("Segment level occurredn")
                 CCBptr->zccbrc=CCBptr->zccberc=0;
                 break;
              case 3:              /* transaction level error         */
              case 4:              /* function group level error      */
              case 5:              /* envelope level error            */
              case 6:              /* invalid data in input file      */
                 lrc=1;
                 printf("Error in Envelope format or contentn");
                 printf("Return code = %ld, extended return code = %ld",
                        CCBptr->zccbrc,CCBptr->zccberc);
                 CCBptr->zccbrc=CCBptr->zccberc=0;
                 break;
              default:
                 lrc = 1;
                 printf("Non translation error occurredn");
                 printf("Return code = %ld, extended return code = %ld",
                        CCBptr->zccbrc,CCBptr->zccberc);
                 break;
           }
        case 12:
           /* ------------------------------------------------------- */
           /* A severe error occurred                                 */
           /* ------------------------------------------------------- */
```

Translating received data using C

```
        lrc = 1;
        printf("A severe error occurred in translationn");
        printf("Return code = %ld, extended return code = %ld",
               CCBptr->zccbrc,CCBptr->zccberc);
        break;
    default:
        /* ------------------------------------------------------- */
        /* An invalid return code was returned                     */
        /* ------------------------------------------------------- */
        lrc = 1;
        printf("An invalid return code from the translatorn")
        printf("Return code = %ld, extended return code = %ld",
               CCBptr->zccbrc,CCBptr->zccberc);
        break;
  }
  return lrc;
}
```

Generating reports

Sample programs are provided for generating several reports.

Generating a data extract report

Two COBOL programs that illustrate how to generate a Transaction Store report using as input the output from a previous PERFORM ENVELOPE DATA EXTRACT Utility command are provided with the product. The library member names for these programs are `EDI.V3R2M0.SEDICBL1(EDISAMR1)` and `EDI.V3R2M0.SEDICBL1(EDISAMS1)`.

Generating a network activity report

A COBOL program that illustrates how to generate a Transaction Store report using as input the output from a previous PERFORM TRADING PARTNER CAPABILITY DATA EXTRACT Utility command is provided with the product. The library member name for this program is `EDI.V3R2M0.SEDICBL1(EDISAMT1)`.

Initializing, invoking, and terminating HOT-DI

Sample COBOL programs are provided that show how to initialize, invoke, and terminate HOT-DI.

COBOL HOT-DI initialization example

This COBOL CICS program shows how to initialize a HOT-DI session using the API. The library member name for this program is `EDI.V3R2M0.SEDICBL2(FXXHOT1)`.

COBOL HOT-DI invocation example

This COBOL CICS program shows how to invoke the WebSphere Data Interchange Utility through a HOT-DI session using the API. The library member name for this program is `EDI.V3R2M0.SEDICBL2(FXXHOT2)`.

COBOL HOT-DI termination example

This COBOL CICS program shows how to terminate a HOT-DI session using the API. The library member name for this program is `EDI.V3R2M0.SEDICBL2(FXXHOT3)`.

Invoking response programs

Response programs are user applications that the WebSphere Data Interchange Utility (or continuous receive facility) invokes. For more information about response programs, see "Response applications" on page 381.

COBOL response program example

This COBOL CICS program shows some of the function of a simple continuous receive response program. The library member name for this program is `EDI.V3R2M0.SEDICBL2(FXXRESP)`.

Field exit programs

Sample field exit programs are provided that show different functions. You can copy these samples and customize them for your application.

Sample 1

This C program is an example of a Send/Receive Translation field user exit. The library member name for this program is `EDI.V3R2M0.SEDICCC1(EDITRX1)`.

Sample 2

This C program is an example of a Send/Receive Translation field user exit. It takes the value that has been accumulated by EDITRX1 and returns it for mapping. The library member name for this program is `EDI.V3R2M0.SEDICCC1(EDITRX2)`.

Sample 3

This COBOL program is an example of a Send/Receive Translation field user exit. The library member name for this program is `EDI.V3R2M0.SEDICBL1(EDICOBE)`.

Test for filter type

This C program is an example of a Send/Receive Translation filter user exit. It checks the Security Profile to determine the type of filtering to be done and invokes the appropriate routine. The library member name for this program is `EDI.V3R2M0.SEDICCC1(EDITRF4)`.

Filtration exit examples

Several filter example programs are provided with the product: their descriptions follow. For more information on the filtration routines and the parameter definitions required for various languages, see "Filtering routine" on page 339.

Filtration exit examples

Hexadecimal filter example

This C program is an example of a Send/Receive Translation filter user exit and is invoked for hexadecimal filtering. The library member name for this program is EDI.V3R2M0.SEDICCC1(EDITRF1).

ASCII filter example

This C program is an example of a Send/Receive Translation filter user exit and is invoked for ASCII filtering. The library member name for this program is EDI.V3R2M0.SEDICCC1(EDITRF2).

ASCII/BAUDOT filter example

This C program is an example of a Send/Receive Translation filter user exit and is invoked for ASCII/BAUDOT filtering. The library member name for this program is EDI.V3R2M0.SEDICCC1(EDITRF3).

Authentication examples

Two authentication example programs are provided with the product: their descriptions follow. For more information about authentication routines, see "Authentication routine" on page 335.

Sample 1

This C program is an example of a Send/Receive Translation authentication user exit. It is invoked for the authentication of data using the IBM 4753 Network Security Processor. The library member name for this program is EDI.V3R2M0.SEDICCC1(EDITRAA).

Sample 2

This C program is an example of a Send/Receive Translation authentication user exit. It is invoked for the authentication of data using the IBM Common Cryptographic Architecture Cryptographic API.

```
/* ----------------------------------------------------------------- */
/* $MAJOR                                                            */
/*                                                                    */
/* Module Name: CCAAA                                                */
/*                                                                    */
/* Descriptive Name: Authentication routine                          */
/*                                                                    */
/* STATUS:                                                           */
/*                                                                    */
/* Function: This program is an example of an authentication         */
/*           routine which uses the IBM Common Cryptographic         */
/*           Architecture Cryptographic Application Programming       */
/*           Interface as defined in reference SC40-1675.            */
/*                                                                    */
/*           This program is invoked by WebSphere Data Interchange    */
/*           during the enveloping or de-enveloping process          */
/*                                                                    */
/* Dependencies: none                                                */
/*                                                                    */
```

```
/* Restrictions: None                                              */
/*                                                                 */
/* Language:    C                                                  */
/*                                                                 */
/* Attributes:   Reentrant, AMODE(31) RMODE(ANY)                   */
/*            INCLUDE OBJ(CCAAA)                                    */
/*            INCLUDE OBJ(FXXZCITF)                                 */
/*            INCLUDE OBJ(FXXZC)                                    */
/*            ENTRY FXXZCITF                                        */
/*            NAME CCAAA(R)                                         */
/*                                                                 */
/* NOTES:  Since this program is written in C, the entry point     */
/*         must be FXXZCITF, which is provided by the WDI product. */
/*         FXXZCITF establishes the necessary C environment        */
/*         and branches to the main entry point of the program.    */
/*                                                                 */
/*         Parameters passed from WDI can be above the line so this */
/*         program has to be 31 bit addressable                    */
/*                                                                 */
/*                                                                 */
/* PARAMETERS:   IN - Service Name Block  (snb)                    */
/*                    Common Block        (ccb)                    */
/*                    Function Block      (fcb)                    */
/*                    Authentication handle (fh)                   */
/*                    Authentication key  (ak)                     */
/*                    Security data block (spdb)                   */
/*                    Buffer size                                  */
/*                    Buffer containing input data                 */
/*                    Length of data in input buffer              */
/*                    Number of characters remaining that would    */
/*                        not fit into the input buffer            */
/*                    Address for return of the MAC value          */
/*                                                                 */
/*         IN OUT - None                                           */
/*                                                                 */
/*            OUT - CAS COMMON BLOCK                               */
/*                                                                 */
/*               RC     ERC        Meaning                         */
/*               8      21         Invalid function code           */
/*               8      24         Error getting data              */
/*                                                                 */
/*         * As defined by the processor implementing the CCA      */
/*                                                                 */
/* ---------------------------------------------------------------- */
#include <stdefs.h>
#pragma linkage (CSNBMGN,OS)      /* MAC Generate Routine          */
#include "disnb.h"                /* SNB definition                */
#include "diccb.h"                /* CCB definition                */
#include "difcb.h"                /* FCB definition                */
#include "dispdb.h"               /* Security data block definition */
  /* ---------------------------------------------------------------- */
  /* Constant definitions used by this program                      */
  /* ---------------------------------------------------------------- */
#define NULLPTR      (void *) 0            /* Null pointer          */
#define EXISTS       0                     /* Existence check       */
```

Authentication examples

```
#define MAC_GEN      1                    /* Generate a MAC        */
#define MAC_VER      2                    /* Verify a MAC          */
#define KEY_SIZE     16                   /* Key size              */
#define MAC_SIZE     4                    /* MAC length            */
/* ---------------------------------------------------------------- */
/* Static data used by program                                      */
/* ---------------------------------------------------------------- */
static fcb  getfcb={4,1};                 /* Used to get more data  */
static char *MAC_METHOD = "X9.9-1 ";      /* MAC X9.9-1 method      */
static char *ONLY       = "ONLY    ";     /* No segmenting          */
static char *FIRST      = "FIRST   ";     /* First segment of data  */
static char *MIDDLE     = "MIDDLE  ";     /* Middle segment of data */
static char *LAST       = "LAST    ";     /* Last segment of data   */
static char *MAC_LENGTH = "MACLEN4 ";     /* MAC length of 4 bytes  */
main(snbptr,ccbptr,fcbptr,fh,ak,spdbptr,
         bufsize,bufin,datalen,rbc,macval)
    snb     *snbptr;    /* Service name block pointer set up by WDI */
                        /* to invoke this program                   */
    ccb     *ccbptr;    /* Common block pointer used by WDI for all */
                        /* function requests                        */
    fcb     *fcbptr;    /* Function block pointer which indicates    */
                        /* the direction of the authentication      */
                        /* 1=SEND(MAC_GEN),  2=RECEIVE(MAC_VER)      */
    long    *fh;        /* Auth. handle which is used when GETTING  */
                        /* more data                                */
    char    *ak;        /* 16 byte key name for the process         */
    spdb    *spdbptr;   /* Security profile member that specified   */
                        /* this program should be called            */
    long    *bufsize;   /* The size of the input buffer             */
    char    *bufin;     /* Buffer that contains the source data     */
    long    *datalen;   /* Length of data within the input buffer   */
    long    *rbc;       /* The number of source bytes remaining     */
                        /* that would not fit into the input buffer */
    long    *macval;    /* Pointer where MAC value should be returned*/
{
    long    rule_count;        /* Rule array count                  */
    char    rule_array[5][8];  /* Rule array                        */
    char    min_in[8];         /* Minimum size buffer to work with  */
    char    *local_in;         /* Local input buffer pointer        */
    long    local_dl;          /* Length of data in local buffer    */
    char    key_token[64];     /* Key token for authentication      */
    long    local_long;        /* Filler variable                   */
    char    chain_vect[18];    /* Chaining vector                   */
    char    local_macval[8];   /* Must pass routine 8 byte area     */
    char    *rule;             /* Current rule to follow            */
/* ---------------------------------------------------------------- */
/* Do some special processing if this is an GEN or VER request.     */
/* ---------------------------------------------------------------- */
if ((fcbptr->zfcbfunc == MAC_GEN)
    (fcbptr->zfcbfunc == MAC_VER))
{
    /* ---------------------------------------------------------------- */
    /* The key passed to this routine by WDI (ak) is a 16-byte key      */
    /* name that is taken from either the WDI mapping records for       */
    /* generation or the S1S or S2S security segments for               */
```

```
/* validation.  The CCA MACgen and MACver calls require that     */
/* a CCA MAC  key be provided as a key token value.  It is your   */
/* responsibility to write code to handle the storage             */
/* of internal key tokens in a key storage data set so that an    */
/* association can be made between the name of a key (key label)  */
/* and the value of that key (CCA MAC  key value).  You           */
/* must also provide a key management routine that can be called  */
/* from this routine.  The function of this key management routine*/
/* would be to accept a 16-byte key label and type as input and   */
/* return as output a 64-byte key identifier.  For more           */
/* information on key labels, key types, and key identifiers, see */
/*    "IBM Common Cryptographic Architecture Cryptographic        */
/*     Application Programming Interface Reference (SC40-1675)"    */
/*                                                               */
/* If your security subsystem is the Integrated Cryptographic     */
/* Feature (ICRF) with the Integrated Cryptographic Service       */
/* Facility / MVS (ICSF/MVS), you should replace "key_transform"  */
/* by a routine that calls whatever program your installation     */
/* uses to manage DATA keys and MAC keys.  For more information   */
/* on ICSF/MVS, see the following ICSF/MVS publications:          */
/*    "General Information (GC23-0093)"                           */
/*    "Administrator's Guide (SC23-0097)"                         */
/*    "Application Programmer's Guide (SC23-0098)"                */
/*    "System Programmer's Guide (SC23-0096)"                     */
/* --------------------------------------------------------------- */
memset(key_token, ' ', sizeof(key_token));
key_transform(ak,"MAC     ",key_token);
/* --------------------------------------------------------------- */
/* Determine what you are asked to do and process.                */
/* --------------------------------------------------------------- */
switch(fcbptr->zfcbfunc)
   {
   case EXISTS:
       /* ------------------------------------------------------ */
       /* Existence check returns with success.                  */
       /* ------------------------------------------------------ */
      break;
   case MAC_GEN:
   case MAC_VER:
       /* ------------------------------------------------------ */
       /* Generate a MAC value both times and let                */
       /* WDI handle the verification.                           */
       /* ------------------------------------------------------ */
       /* ------------------------------------------------------ */
       /* Set up some initial values.                            */
       /* ------------------------------------------------------ */
      rule_count = 3;                    /* Method, control, length */
      memset(min_in, ' ', sizeof(min_in));
      memset(chain_vect, 0x00,sizeof(chain_vect));
      memset(rule_array, ' ', sizeof(rule_array));
      memcpy(rule_array[0], MAC_METHOD, sizeof(rule_array[0]));
      memcpy(rule_array[2], MAC_LENGTH, sizeof(rule_array[2]));
      memcpy(key_token, ak, KEY_SIZE);
       /* ------------------------------------------------------ */
       /* Set up local pointers and sizes. You want to           */
```

```
/* ensure that you have a buffer of at least 8          */
/* bytes.                                               */
/* ----------------------------------------------------- */
local_dl = *datalen;
if (*bufsize < 8)
    {
    memcpy(min_in, bufin, *bufsize);
    local_in = min_in;
    }
else
    local_in = bufin;
/* ----------------------------------------------------- */
/* In order to chain the data, you have to have        */
/* at least 8 bytes for the -FIRST- segment.           */
/* ----------------------------------------------------- */
if (*rbc)
    {
    if (local_dl < 8)
        {
            /* --------------------------------------------- */
            /* Attempt to fill the buffer up.               */
            /* --------------------------------------------- */
        local_long = 8 - local_dl;
        fxxzc(fh,ccbptr,&getfcb,
            &local_in[local_dl],&local_long);
        if (ccbptr->zccbrc)
            {
            ccbptr->zccbrc = 8;      /* Error ...            */
            ccbptr->zccberc = 24;    /* Error getting data   */
            return;
            }
        local_dl += local_long;
        }
        /* ------------------------------------------------- */
        /* If you have more bytes left, you should          */
        /* have filled the buffer up to 8 for               */
        /* the -FIRST- segment.                             */
        /* ------------------------------------------------- */
    rule = (*rbc) ? FIRST : ONLY;
    }
else
    rule = ONLY;
memcpy(rule_array[1], rule, sizeof(rule_array[1]));
/* ----------------------------------------------------- */
/* Call the MAC generate verb.                          */
/* ----------------------------------------------------- */
CSNBMGN(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
        NULLPTR, key_token, &local_dl, local_in, &rule_count,
        rule_array, chain_vect, local_macval);
if (ccbptr->zccbrc)                    /* Error? ...          */
    return;
while (*rbc)
    {
        /* ------------------------------------------------- */
        /* Get some more data.                              */
```

```
                        /* ------------------------------------------------- */
                    fxxzc(fh, ccbptr, &getfcb, local_in, &local_dl);
                    if (ccbptr->zccbrc)
                        {
                        ccbptr->zccbrc = 8;          /* Error ...             */
                        ccbptr->zccberc = 24;        /* Error getting data   */
                        return;
                        } /* end if */
                        /* ------------------------------------------------- */
                        /* Reset the segmenting control.                     */
                        /* ------------------------------------------------- */
                    memcpy(rule_array[1], ((*rbc) ? MIDDLE : LAST),
                          sizeof(rule_array[1]));
                        /* ------------------------------------------------- */
                        /* Call the MAC generate verb.                       */
                        /* ------------------------------------------------- */
                    CSNBMGN(&(ccbptr->zccbrc),&(ccbptr->zccberc),NULLPTR,
                          NULLPTR,key_token,&local_dl,local_in,&rule_count,
                          rule_array,chain_vect,local_macval);
                    if (ccbptr->zccbrc)              /* Error?
...                  */
                        return;
                    } /* end while */
                memcpy(macval,local_macval,MAC_SIZE); /* Copy 4-byte MAC    */
                break;
            default:
                    /* ------------------------------------------------------- */
                    /* Wrong function code.  Set error and return.           */
                    /* ------------------------------------------------------- */
                ccbptr->zccbrc = 8;                   /* Error ...            */
                ccbptr->zccberc = 21;                 /* Invalid function code */
                break;
            } /* end switch */
    return;
    } /* end main */
```

Encryption examples

Two encryption example programs are provided with the product: their descriptions follow. For more information on the encryption routines, see "Encryption routine" on page 331.

Sample 1

This C program is an example of a Send/Receive Translation encryption user exit. It is invoked for the encryption/decryption of data using the IBM 4753 Network Security Processor. The library member name for this program is `EDI.V3R2M0.SEDICCC1(EDITREE)`.

Sample 2

This C program is an example of a Send/Receive Translation encryption user exit. It is invoked for the encryption/decryption of data using the IBM Common Cryptographic Architecture Cryptographic API.

Encryption examples

```
/* ----------------------------------------------------------------- */
/* $MAJOR                                                            */
/*                                                                   */
/* Module Name: CCAEE                                                */
/*                                                                   */
/* Descriptive Name: Encryption/Decryption routine                  */
/*                                                                   */
/* Status:                                                           */
/*                                                                   */
/* Function: This program is an example of a encryption/decryption   */
/*           routine which uses the IBM Common Cryptographic         */
/*           Architecture Cryptographic Application Programming      */
/*           Interface as defined in reference SC40-1675.            */
/*                                                                   */
/*           This program is invoked by WebSphere Data Interchange   */
/*           during the enveloping or de-enveloping process          */
/*                                                                   */
/* Dependencies: none                                                */
/*                                                                   */
/* Restrictions: none                                                */
/*                                                                   */
/* Language:     C                                                   */
/*                                                                   */
/* Attributes:   Reentrant, AMODE(31) RMODE(ANY)                     */
/*               INCLUDE OBJ(CCAEE)                                  */
/*               INCLUDE OBJ(FXXZCITF)                               */
/*               INCLUDE OBJ(FXXZC)                                  */
/*               ENTRY FXXZCITF                                      */
/*               NAME CCAEE(R)                                       */
/*                                                                   */
/* Notes:        Since this program is written in C, the entry       */
/*               point must be FXXZCITF, which is provided by the    */
/*               WDI product. FXXZCITF establishes the necessary     */
/*               C environment and branches to the main entry point  */
/*               of the program.                                     */
/*                                                                   */
/*               Parameters passed from WDI can be above the line so */
/*               this program has to be 31 bit addressable.          */
/*                                                                   */
/* Parameters:   IN - Service Name Block   (snb)                     */
/*                    Common Block         (ccb)                     */
/*                    Function Block       (fcb)                     */
/*                    Encryption Handle    (fh)                      */
/*                    Encryption key value (ek)                      */
/*                    Security Data Block  (spdb)                    */
/*                    Buffer size                                    */
/*                    Buffer containing input data                  */
/*                    Buffer containing output data                 */
/*                    Length of data in input buffer                */
/*                    Number of characters remaining that would     */
/*                     not fit into the input buffer                */
/*                    Initialization vector return address          */
/*                                                                   */
/*         IN OUT - None                                             */
/*                                                                   */
```

```
/*              OUT - CAS Common Block                        */
/*                                                            */
/*              RC     ERC    Meaning                         */
/*              0      0      Success                         */
/*              8      21     Invalid function code           */
/*              8      22     CFB method not supported        */
/*              8      23     Error returning data to WDI     */
/*              8      24     Error getting data from WDI     */
/*              8      26     Invalid input data              */
/*              8      11     Keyname not known               */
/*              *      *      *                                */
/*                                                            */
/*              * As defined by processor implementing the   */
/*                CCA                                         */
/*                                                            */
/* ---------------------------------------------------------- */
#include <stdefs.h>
#include "disnb.h"              /* SNB definition            */
#include "diccb.h"              /* CCB definition            */
#include "difcb.h"              /* FCB definition            */
#include "dispdb.h"             /* Security data block definition */
#pragma linkage (CSNBRNG,OS)    /* Random Number Generate    */
#pragma linkage (CSNBENC,OS)    /* Encipher routine          */
#pragma linkage (CSNBDEC,OS)    /* Decipher routine          */
   /* ---------------------------------------------------------- */
   /* Constant definitions used by this program               */
   /* ---------------------------------------------------------- */
#define EXISTS   0                  /* Existence check         */
#define ENCRYPT  1                  /* Encrypt data            */
#define DECRYPT  2                  /* Decrypt data            */
#define GETIV    3                  /* Get initialization vector */
#define CBC_TYPE '1'                /* CBC ciphering type      */
#define IV_SIZE  8                  /* Initialization vector size */
#define KEY_SIZE 16                 /* Input key size          */
#define NULLPTR  (void *) 0         /* NULL pointer            */
#define True     1
#define False    0
   /* ---------------------------------------------------------- */
   /* Static data used by this program                        */
   /* ---------------------------------------------------------- */
static fcb  getfcb = {4,1};         /* Used to get more data   */
static fcb  putfcb = {4,2};         /* Used to put data        */
static char *RANDOM  = "RANDOM ";  /* Used for random number  */
static char *CBC_METH = "CBC    "; /* CBC ciphering method    */
static char *ICV_INIT = "INITIAL "; /* Starting ICV value     */
   /* ---------------------------------------------------------- */
   /* Local function prototypes                               */
   /* ---------------------------------------------------------- */
    void pad_buff(char *buff,long data_len,long *crypt_len);
/* ---------------------------------------------------------- */
/* Main entry point begins here.                              */
/* ---------------------------------------------------------- */
main(snbptr, ccbptr, fcbptr, fh, ek, spdbptr,
          bufsize, bufin, bufout, datalen, rbc, iv)
    snb    *snbptr;    /* Service name block pointer set up by WDI  */
```

Encryption examples

```
    ccb     *ccbptr;    /* Common block pointer used by WDI        */
    fcb     *fcbptr;    /* Function block pointer which indicates  */
                        /* what to do:                             */
                        /* 0=EXISTS, 1=ENCRYPT, 2=DECRYPT, 3=GETIV */
    long    *fh;        /* Encryption handle - used with GET and PUT */
    char    *ek;        /* Encryption key name                     */
    spdb    *spdbptr;   /* Security profile member                 */
    long    *bufsize;   /* The size of the input/output buffers    */
    char    *bufin;     /* Buffer that contains the source data    */
    char    *bufout;    /* Buffer for the output data              */
    long    *datalen;   /* Length of data in the input buffer      */
    long    *rbc;       /* The number of bytes remaining of the source*/
                        /* that would not fit into the input buffer  */
    char    *iv;        /* Initialization vector return area       */
{
    long   rule_count;        /* Rule array count                  */
    char   rule_array[8];     /* Rule array                        */
    char   form[8];           /* Form of random number             */
    char   min_in[16];        /* Minimum size buffer to work with  */
    char   min_out[16];       /* Minimum output buffer to work out */
    char   *local_in;         /* Local input buffer pointer        */
    char   *local_out;        /* Local output buffer pointer       */
    long   local_bs;          /* Size of local buffer              */
    long   local_dl;          /* Length of data in local buffer    */
    char   key_token[64];     /* CCA DATA key token                */
    char   siv[IV_SIZE];      /* Save original IV value            */
    long   filler;            /* Filler variable                   */
    long   bytes_left;        /* Number bytes remaining after input */
                              /* buffer adjusted for encryption    */
    long   crypt_in;          /* Encryption/decryption length      */
    long   crypt_len;         /* Encryption/decryption length      */
    char   chain_vect[18];    /* Chaining vector                   */
    int    padded = False;    /* Loop exit variable                */
    /* ------------------------------------------------------------ */
    /* Set up some initial values.                                  */
    /* ------------------------------------------------------------ */
rule_count = 1;
memset(min_in, ' ', sizeof(min_in));
memset(min_out, ' ', sizeof(min_out));
memset(rule_array, ' ', sizeof(rule_array));
memcpy(rule_array, CBC_METH, sizeof(rule_array));
memset(key_token, ' ', sizeof(key_token));
/* ------------------------------------------------------------ */
/* Do some special processing if this is an encrypt or decrypt  */
/* request.                                                     */
/* ------------------------------------------------------------ */
if ((fcbptr->zfcbfunc == ENCRYPT)
   (fcbptr->zfcbfunc == DECRYPT)) {
    /* ------------------------------------------------------------ */
    /* Check for the correct ciphering method.  CCA only defines the */
    /* CBC method, so reject a request for the Cipher Feedback method */
    /* ------------------------------------------------------------ */
    if (spdbptr->encrtype != CBC_TYPE)
{
        ccbptr->zccbrc = 8;                  /* Error ...            */
```

```
        ccbptr->zccberc = 22;              /* Invalid cipher method   */
        return;
    }
    /* ------------------------------------------------------------- */
    /* The key passed to this routine by WDI (ek) is a 16-byte key   */
    /* name that is taken from either the WDI mapping records for     */
    /* encryption or the S1S or S2s security segments for             */
    /* decryption.  The CCA Encipher and Decipher calls require that  */
    /* a CCA DATA key be provided as a key token value.  It is        */
    /* your responsibility to write code to handle the storage        */
    /* of internal key tokens in a key storage data set so that an    */
    /* association can be made between the name of a key (key label)  */
    /* and the value of that key (CCA DATA key value).  You           */
    /* must also provide a key management routine that can be called  */
    /* from this routine.  The function of this key management routine*/
    /* would be to accept a 16-byte key label and type as input and   */
    /* return as output a 64-byte key identifier.  For more           */
    /* information on key labels, key types, and key identifiers, see */
    /*    "IBM Common Cryptographic Architecture Cryptographic        */
    /*     Application Programming Interface Reference (SC40-1675)"    */
    /*                                                                 */
    /* If your security subsystem is the Integrated Cryptographic     */
    /* Feature (ICRF) with the Integrated Cryptographic Service       */
    /* Facility / MVS (ICSF/MVS), you should replace "key_transform"  */
    /* by a routine that calls whatever program your installation     */
    /* uses to manage DATA keys and MAC keys.  For more information   */
    /* on ICSF/MVS, see the following ICSF/MVS publications:          */
    /*    "General Information (GC23-0093)"                           */
    /*    "Administrator's Guide (SC23-0097)"                        */
    /*    "Application Programmer's Guide (SC23-0098)"               */
    /*    "System Programmer's Guide (SC23-0096)"                    */
    /* ------------------------------------------------------------- */
    key_transform(ek,"DATA    ",key_token);
}
    /* ------------------------------------------------------------- */
    /* Determine what you are asked to do and process accordingly.   */
    /* ------------------------------------------------------------- */
switch(fcbptr->zfcbfunc)
    {
    case EXISTS:
        /* ----------------------------------------------------- */
        /* Existence check returns with success.                 */
        /* ----------------------------------------------------- */
        break;
    case GETIV:
        /* ----------------------------------------------------- */
        /* Call the Random Number generate verb to get a random  */
        /* number.  Return codes from the CSNBRNG routine will be */
        /* returned to your caller.                              */
        /* ----------------------------------------------------- */
        memcpy(form, RANDOM, sizeof(form));
        CSNBRNG(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, form, iv);
        break;
    case ENCRYPT:
```

Encryption examples

```
/* --------------------------------------------------------- */
/* Tuck away the original IV value for later encryption.  */
/* --------------------------------------------------------- */
memcpy(siv,iv,IV_SIZE);
/* --------------------------------------------------------- */
/* The size of the buffers passed by WDI is                */
/* specified in the Security Profile.  This program needs a */
/* buffer size of at least 16 bytes.  If you specified     */
/* a size less than 16, use a buffer internal to this      */
/* routine rather than the buffers passed by WDI.          */
/* --------------------------------------------------------- */
if (*bufsize < 16) {
    memcpy(min_in, bufin, *bufsize);
    local_in = min_in;
    local_out = min_out;
    local_bs = sizeof(min_in);
} else {
    local_in = bufin;
    local_out = bufout;
    local_bs = *bufsize;
}
local_dl = *datalen;
/* --------------------------------------------------------- */
/* Now that your working buffer size is right,             */
/* you might have gotten data in that is less than         */
/* 8 bytes. First, attempt to fill the buffer              */
/* up to 8 bytes. If that doesn't work, pad it.            */
/* --------------------------------------------------------- */
if (local_dl < 8) {
    if (*rbc) {                          /* Any bytes not passed?*/
        filler = 8 - local_dl;
        fxxzc(fh,ccbptr,&getfcb, &local_in[local_dl],&filler);
        if (ccbptr->zccbrc) {
            ccbptr->zccbrc = 8;        /* Error ...            */
            ccbptr->zccberc = 24;      /* Error getting data  */
            return;
        }
        if ((filler+local_dl) < 8)
    {
            pad_buff(local_in,filler+local_dl,&local_bs);
            padded = True;
        }
        local_dl = 8;              /* Either padded or filled*/
    } else {
        pad_buff(local_in, local_dl, &local_bs);
        local_dl = 8;
        padded = True;
    } /* end if(rbc) */
} /* end if(local_dl) */
/* --------------------------------------------------------- */
/* Adjust the buffer length to a multiple of 8             */
/* and set the number of bytes that will not be            */
/* processed the first pass.                               */
/* --------------------------------------------------------- */
bytes_left = local_dl % 8;
```

```
crypt_len = local_dl - (bytes_left);
if (crypt_len > 8) {
    crypt_len -= 8;
    bytes_left += 8;
}
crypt_in = crypt_len;
/* ---------------------------------------------------------- */
/* Start looping until all data is processed.                 */
/* You know you are finished when some type of                */
/* padding has been done.                                     */
/* ---------------------------------------------------------- */
for(;;) {
    /* ---------------------------------------------------- */
    /* Call the encipher verb and check for errors.         */
    /* ---------------------------------------------------- */
    filler = 0;
    crypt_len = crypt_in;
    CSNBENC(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, key_token, &crypt_len, local_in, iv,
            &rule_count, rule_array, &filler, chain_vect,
            local_out);
    if (ccbptr->zccbrc)
        return;
    /* ---------------------------------------------------- */
    /* Use the WDI provided interface to return the encrypted*/
    /* data to WDI.                                         */
    /* ---------------------------------------------------- */
    fxxzc(fh, ccbptr, &putfcb, local_out, &crypt_len);
    if (ccbptr->zccbrc) {
        ccbptr->zccbrc = 8;       /* Error ...             */
        ccbptr->zccberc = 23;     /* Error putting data    */
        return;
    }
    /* ---------------------------------------------------- */
    /* If padding was done, you are finished.               */
    /* Else, continue with next set of data.                */
    /* ---------------------------------------------------- */
    if (padded)
        break;
    /* ---------------------------------------------------- */
    /* Ensure that any bytes not processed in the           */
    /* first pass are accounted for.                        */
    /* ---------------------------------------------------- */
    if (bytes_left)
        memcpy(local_in,local_in+crypt_in,bytes_left);
    /* ---------------------------------------------------- */
    /* Check for any data that was not passed to            */
    /* you in the original input buffer.                    */
    /* ---------------------------------------------------- */
    if (bytes_left >= crypt_in)
{
        bytes_left -= crypt_in;
    } else {
        if (*rbc) {
            filler = crypt_in - bytes_left;
```

```
                fxxzc(fh,ccbptr,&getfcb,
                    &local_in[bytes_left],&filler);
                if (ccbptr->zccbrc) {
                    ccbptr->zccbrc = 8;  /* Error ...              */
                    ccbptr->zccberc = 24;/* Error getting data    */
                    return;
                }
            } else
                filler = 0;
            /* ------------------------------------------------- */
            /* Check to see if any padding needs to be done.     */
            /* ------------------------------------------------- */
            if ((filler+bytes_left) < crypt_in)
    {
                pad_buff(local_in, filler+bytes_left, &crypt_in);
                padded = True;
            }
            bytes_left = 0;                    /* Set back to zero      */
        }
        /* ----------------------------------------------------- */
        /* Continuation is accomplished by copying the first 8   */
        /* bytes of the chaining vector into the initialization  */
        /* vector for the next call.                             */
        /* ----------------------------------------------------- */
        memcpy(iv,chain_vect,IV_SIZE);
    } /* end for(;;) */
    /* --------------------------------------------------------- */
    /* Encrypt the IV using ECB.  The CBC method with an IV of   */
    /* 0 and 8 bytes of data is the same.                        */
    /* --------------------------------------------------------- */
    crypt_len = IV_SIZE;
    memset(iv,0x00,IV_SIZE);
    CSNBENC(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, key_token, &crypt_len, siv, iv,
            &rule_count, rule_array, &filler, chain_vect, min_out);
    /* --------------------------------------------------------- */
    /* Check for error encrypting the IV.                        */
    /* --------------------------------------------------------- */
    if (ccbptr->zccbrc)
        return;
    /* --------------------------------------------------------- */
    /* Return the encrypted IV to the caller.                    */
    /* --------------------------------------------------------- */
    memcpy(iv,min_out,IV_SIZE);
    break;
  case DECRYPT:
    /* --------------------------------------------------------- */
    /* Encrypted data MUST be a multiple of 8.                   */
    /* --------------------------------------------------------- */
    if (((*rbc + *datalen)%8 != 0)
 ((*rbc + *datalen) <= 0))
 {
        ccbptr->zccbrc = 8;                    /* Error ...           */
        ccbptr->zccberc = 26;                  /* Input data error    */
        return;
```

```
}
/* ---------------------------------------------------------- */
/* Call the decipher verb to decipher the IV.  You are using */
/* the CBC method with an IV of 0 and data length of 8 which */
/* is the same as the ECB method.                            */
/* ---------------------------------------------------------- */
crypt_len = IV_SIZE;
memcpy(min_in, iv, IV_SIZE);
memset(form, 0x00, sizeof(form));
CSNBDEC(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
        NULLPTR, key_token, &crypt_len, min_in, form,
        &rule_count, rule_array, chain_vect, iv);
/* ---------------------------------------------------------- */
/* Check for errors deciphering the IV.                      */
/* ---------------------------------------------------------- */
if (ccbptr->zccbrc)
    return;
/* ---------------------------------------------------------- */
/* The size of the buffers passed by WDI is                  */
/* specified in the Security Profile.  This program needs a  */
/* buffer size of at least 8 bytes.  If you specified a size */
/* less than 8, use a buffer internal to this                */
/* routine rather than the buffers passed by WDI.            */
/* ---------------------------------------------------------- */
local_dl = *datalen;
if (*bufsize < 8) {
    memcpy(min_in, bufin, *bufsize);
    local_in = min_in;
    local_out = min_out;
    local_bs = sizeof(min_in);
} else {
    local_in = bufin;
    local_out = bufout;
    local_bs = *bufsize;
} /* end if */
/* ---------------------------------------------------------- */
/* Now that your working buffer size is right, you might have*/
/* received data that is less than 8 bytes.  Encrypted data  */
/* MUST be a multiple of 8 bytes.  If you did not receive    */
/* 8, there must be more out there to get.                   */
/* ---------------------------------------------------------- */
if (local_dl < 8) {
    filler = 8 - local_dl;          /* Get only up to 8      */
    fxxzc(fh,ccbptr,&getfcb, &local_in[local_dl],&filler);
    if (ccbptr->zccbrc) {
        ccbptr->zccbrc = 8;         /* Error ...             */
        ccbptr->zccberc = 24;       /* Error getting data    */
        return;
    }
    local_dl = 8;
}
 /* ---------------------------------------------------------- */
 /* Adjust the buffer length to a multiple of 8               */
 /* and set the number of bytes that will not be             */
 /* processed the first pass. If you were passed             */
```

Encryption examples

```
            /* a buffer length that was not a multiple of         */
            /* 8, you will process only the highest multiple       */
            /* you can get on the first call to decipher,          */
            /* then any bytes left with the residual not           */
            /* passed to us.                                       */
            /* ----------------------------------------------------- */
bytes_left = local_dl % 8;
crypt_len = local_dl - bytes_left;
            /* ----------------------------------------------------- */
            /* Start looping until all data is processed.          */
            /* You break out of this loop when you determine       */
            /* that all characters have been processed and         */
            /* subtracted the number of pad characters.            */
            /* ----------------------------------------------------- */
for(;;) {
    /* ----------------------------------------------------- */
    /* Call the decipher verb and check for errors.        */
    /* ----------------------------------------------------- */
    crypt_in = crypt_len;
    CSNBDEC(&(ccbptr->zccbrc), &(ccbptr->zccberc), NULLPTR,
            NULLPTR, key_token, &crypt_len, local_in, iv,
            &rule_count, rule_array, chain_vect, local_out);
    if (ccbptr->zccbrc)
        return;
    /* ----------------------------------------------------- */
    /* Put the data out. If no data is left to pro-        */
    /* cess, the last byte will contain the number         */
    /* of pad characters in ASCII. When this is            */
    /* encountered, you exit the loop.                     */
    /* ----------------------------------------------------- */
    if (*rbc) {
        fxxzc(fh, ccbptr, &putfcb, local_out, &crypt_len);
        if (ccbptr->zccbrc) {
            ccbptr->zccbrc = 8;      /* Error ...            */
            ccbptr->zccberc = 23;    /* Error putting data   */
            return;
        } /* end if */
    } else {
        /* ----------------------------------------------------- */
        /* The last position must be the number                */
        /* of pad characters in ASCII. If this                 */
        /* is not true, exit in error.                         */
        /* ----------------------------------------------------- */
        filler = local_out[crypt_len-1] & 0x0F;
        if ((filler < 1)  (filler > 8))
{
            ccbptr->zccbrc = 8;      /* Error ...            */
            ccbptr->zccberc = 26;    /* Input data error     */
            return;
        } /* end if */
        /* ----------------------------------------------------- */
        /* Flush output minus pad characters.                  */
        /* ----------------------------------------------------- */
        crypt_len = crypt_len - filler;
        fxxzc(fh, ccbptr, &putfcb, local_out, &crypt_len);
```

```
                  if (ccbptr->zccbrc) {
                      ccbptr->zccbrc = 8;        /* Error ...          */
                      ccbptr->zccberc = 23;      /* Error putting data  */
                      return;
                  } /* end if */
                  break;                         /* Exit the loop       */
              } /* end if */
              /* ------------------------------------------------------ */
              /* Ensure that any bytes not processed in the             */
              /* first pass are accounted for. This only                */
              /* needs to be done for the first pass.                   */
              /* ------------------------------------------------------ */
              if (bytes_left)
                  memcpy(local_in, local_in+crypt_in, bytes_left);
              /* ------------------------------------------------------ */
              /* Get any data that was not passed to you in             */
              /* the original input buffer. You wouldn't have           */
              /* gotten this far if there wasn't anything               */
              /* there.                                                 */
              /* ------------------------------------------------------ */
              filler = crypt_in - bytes_left;
              fxxzc(fh,ccbptr,&getfcb, &local_in[bytes_left],&filler);
              if (ccbptr->zccbrc) {
                  ccbptr->zccbrc = 8;        /* Error ...          */
                  ccbptr->zccberc = 24;      /* Error getting data  */
                  return;
              }
              crypt_len = filler + bytes_left;
              bytes_left = 0;                    /* Set back to zero    */
              /* ------------------------------------------------------ */
              /* Change the initialization vector for the next call.   */
              /* ------------------------------------------------------ */
              memcpy(iv,chain_vect,IV_SIZE);
          } /* end for(;;) */
          break;
      default:
          /* ------------------------------------------------------ */
          /* Wrong function code.  Set error and return.            */
          /* ------------------------------------------------------ */
          ccbptr->zccbrc = 8;                 /* Error ...          */
          ccbptr->zccberc = 21;               /* Invalid function code */
          break;
  } /* end switch */
  return;
} /* end main */
/* ---------------------------------------------------------------- */
/* Routine: pad_buff                                                */
/*                                                                  */
/* Purpose: Pad a buffer with ASCII 0s and put the number of places */
/*          padded in the last position.                            */
/*                                                                  */
/* In     : buff     - The buffer to pad                            */
/*          data_len - The length of data in the buffer             */
/*          crypt_len - Current maximum buff size                   */
/*                                                                  */
```

Encryption examples

```
/* Out    : Modified buffer (buff) and data length (crypt_len)   */
/*                                                                */
/* -------------------------------------------------------------- */
void pad_buff(buff, data_len, crypt_len)
    char *buff;                    /* Input buffer to pad         */
    long  data_len;                /* Current data length in buffer */
    long  *crypt_len;              /* Current cipher length of buffer*/
    long num_pad = 0;              /* Number of characters padded */
    /* -------------------------------------------------------------- */
    /* If the buffer length is on a 8-byte boundary (Including      */
    /* zero), force the first pad. 'crypt_len' is always on an      */
    /* 8-byte boundary and 'data_len' is always less than it, so    */
    /* you can never overflow your buffer.                          */
    /* -------------------------------------------------------------- */
if ((data_len%8) == 0) {
    buff[data_len++] = 0x30;         /* Force the first pad        */
    num_pad = 1;
} /* end if */
    /* -------------------------------------------------------------- */
    /* Pad with ASCII 0s.                                           */
    /* -------------------------------------------------------------- */
for (;data_len%8;num_pad++, ++data_len)
    buff[data_len] = 0x30;
    /* -------------------------------------------------------------- */
    /* Put number of pad characters in ASCII in last position       */
    /* and change the crypt length to new buffer size.             */
    /* -------------------------------------------------------------- */
buff[data_len-1] = (num_pad & 0x0F)  0x30;
*crypt_len = data_len;
} /* end pad_buff() */
```

Get Envelope service example

This COBOL CICS program is an example of an outbound envelope user exit
(PERFORM ENVELOPE...IEXIT(EDIGETE) ITYPE(UE)). The library member name for
this program is EDI.V3R2M0.SEDICBL2(FXXGETE).

Put Envelope service example

This COBOL CICS program is an example of an inbound envelope user exit
(PERFORM DEENVELOPE...IEXIT(EDIPUTE) ITYPE(UE)). The library member name
for this program is EDI.V3R2M0.SEDICBL2(FXXPUTE).

Inbound envelope program example

This COBOL CICS program is an example of an inbound envelope user exit
(PERFORM DEENVELOPE...IEXIT(EDIPUTE) ITYPE(UE)). The library member name
for this program is EDI.V3R2M0.SEDICBL2(FXXIENV).

Outbound envelope program example

This COBOL CICS program is an example of an outbound envelope program (PERFORM ENVELOPE...IEXIT(EDIOENV) ITYPE(PG)). The library member name for this program is `EDI.V3R2M0.SEDICBL2(FXXOENV)`.

VANICICS network program example

This COBOL CICS program shows the basic structure of a VANICICS network program. The library member name for this program is `EDI.V3R2M0.SEDICBL2(FXXNETP)`.

VANICICS network program example

Appendix C. Space calculation examples

Space requirements for tables and files

The following tables describe the database records and formulas used for determining the space requirements for each of the WebSphere Data Interchange tables and files:

- Table 237
- Table 238 on page 753
- Table 239 on page 757
- Table 240 on page 761
- Table 241 on page 765
- Table 242 on page 766

The table headings have the following meanings:

Record name   Identifies the DB2 table names and VSAM file names of the WebSphere Data Interchange database records that you have installed.

Product   Specifies if the record applies to DB2, or to VSAM, or to both.

Type   Specifies whether the database record is defined as a DB2 table, DB2 index, or as a VSAM file..

Description   Specifies the database record and formulas to assist in estimating DASD allocation. References to database records are given using the naming convention of EDIxxxx. For example, the control string table is EDICSTX..

Table 237. Database records (general)

Record name	Product/ Type	Description
SCREENS	z/OS/VSAM file	WebSphere Data Interchange Screens. Contains the text for all screens displayed by WebSphere Data Interchange. Static file unless updated during PTF application
HELPS	z/OS/VSAM file	WebSphere Data Interchange Help Text. Contains the help text for all online help provided by WebSphere Data Interchange. Static file unless updated during PTF application
EDISSTK	ALL/DB2 table	SAP tracking file - Maintains SAP control information and EDI subsystem status to create the SAP status record SSTKNUM = TSTHNUM
EDISSTKX	ALL/DB2 table	SAP Tracking Unique Index - One record for each SAP record SSTKXNUM = SSTKNUM
EDISSTXX	ALL/DB2 table	SAP tracking file index. An index used to maintain the EDISSTK table. One record for each trading partner, interchange control number, and receiver id in EDISSTK. SSTXXNUM = TSTHNUM

Space requirements for tables and files

Table 237. Database records (general) (continued)

Record name	Product/ Type	Description
EDIMSGS	ALL/DB2 table	Message Table. Contains the text for all messages issued by WebSphere Data Interchange. This is a static table except during PTF application when messages can be added or updated. MSGNUM = 2448 (for base 3.2)
EDIMSGSX	ALL/DB2 index	Message Table Index. There is one index entry for each message. MSGSXNUM = MSGSNUM
EDIELOG	ALL/DB2 table	Event Log Table. Contains all the event log entries for all the application IDs used. EDIIMP is the application for logging messages during product administration. EDIFFS is the application for logging messages during utility execution. ELOGNUM = Number of total event log entries
EDIELOGX	ALL/DB2 index	Event Log Table Index. There is one index entry for each event log entry by application ID. ELOGXNUM = ELOGNUM
EDIELOG1	ALL/DB2 index	Event Log Table Index. There is one index entry for each event log entry by application ID and user ID. ELOG1NUM = ELOGNUM
EDIELOG2	ALL/DB2 index	Event Log Table Index. There is one index entry for each event log entry by application ID and user ID. ELOG2NUM = ELOGNUM
EDIENVP	ALL/DB2 table	Enveloping Plug-in Description - one row exists for each envelope type supported. The table is fixed in size. ENVP = 5
EDIENVPX	ALL/DB2 index	Enveloping Plug-in Table Index - One row exists for each entry of the EDIENVP table. ENVPX = ENVP
EDICSTX	ALL/DB2 table	Control String Header and Detail. Control string used to speed execution during translation. There are Map CS, DF CS, and STD CS. In general, there is one entry for each EDIADFHDR entry, one for each EDISTDSTH entry, and four entries for each EDIMAPHDR entry. CSTXNUM = TDIDNUM + (4 * TPTXNUM)
EDICSTXX	ALL/DB2 index	Control String Unique Index. A unique index created on the key value to EDICSTX. There is one index entry for each EDICSTX entry. CSTXXNUM = CSTXNUM

Table 238. Standards database records

Record name	Product/ Type	Description
EDISTDDEH	ALL/DB2 table	Standard Data Element Definition. Provides information about a data element within a standard. One entry exists for each data element defined in each standard. These are some of the standards shipped by WebSphere Data Interchange: **EDI902** 304 **TDCC28** 863 **UCSV3R1** 323 **X12V2R2** 710 **X12V2R3** 765 **X12V3R4** 823 **X12V3R1** 901 **E** 34 **I** 22 **T** 22 **U** 21 **X** 28 SCDENUM = one of the above values or your own estimate
EDISTDDEHX	ALL/DB2 index	Standard Data Element Definition Unique Index. A unique index created on the key value to EDISTDDEH. There is one index entry for each EDISTDDEH entry. SCDEXNUM = SCDENUM
EDISTDDEHY	ALL/DB2 index	Standard Data Element Definition Index. An index created for each STDID in EDISTDDEH. There is one index entry for each EDISTDDEH entry. SCDEYNUM = SCDENUM
EDISTDDED	ALL/DB2 table	Standard Element Detail (Composite DE). Provides information about usage of a subelement within an element. One entry exists for each element defined in each standard of a Composite DE. SCEDNUM = your own estimate
EDISTDDEDU	ALL/DB2 index	Standard Element Detail Unique Index. A unique index created on the key value to EDISTDDED. There is one index entry for each EDISTDDED entry. SCEDUNUM = SCEDNUM
EDISTDDEDX	ALL/DB2 index	Standard Element Detail Unique Index. A unique index created on COMPID, STDID, and POSNO value of EDISTDDED. There is one index entry for each EDISTDDED entry. SCEDXNUM = SCEDNUM
EDISTDDEDY	ALL/DB2 index	Standard Element Detail Index. An index created on DEID, COMPID, and STDID value of EDISTDDED. There is one index entry for each EDISTDDED entry. SCEDYNUM = SCEDNUM
EDISTDDEDZ	ALL/DB2 index	Standard Element Detail Index. An index created on STDID and COMPID value of EDISTDDED. There is one index entry for each EDISTDDED entry. SCEDZNUM = SCEDNUM

Space requirements for tables and files

Table 238. Standards database records (continued)

Record name	Product/Type	Description
EDISTDCDN	ALL/DB2 table	Standard Element Detail (Composite DE) Note Definition. Provides information about composite data element notes within a standard. One entry exists for each CDE defined within a standard. SCCDNUM = your own estimate
EDISTDCDNX	ALL/DB2 index	Standard Composite Data Element Note Unique Index. A unique index created on the key value to EDISTDCDN. There is one index entry for each EDISTDCDN entry. SCCDXNUM = SCCDNUM
EDISTDSGD	ALL/DB2 table	Standard Data Element Usage. Records usage of a data element within a segment. One of these exists each time an element is used within a segment. These are some of the standards shipped by WebSphere Data Interchange: **EDI902** 1012 **TDCC28** 3677 **UCSV3R1** 1083 **X12V2R2** 2830 **X12V2R3** 3001 **X12V3R4** 3382 **X12V3R1** 3707 **E** 46 **I** 26 **T** 22 **U** 25 **X** 32 SCDUNUM = one of the above values or your own estimate
EDISTDSGDU	ALL/DB2 index	Standard Data Element Usage Unique Index. A unique index created on the key value to EDISTDSGD. There is one index entry for each EDISTDSGD entry. SCDUUNUM = SCDUNUM
EDISTDSGDX	ALL/DB2 index	Standard Data Element Usage Unique Index. A unique index created on segment, standard, and posno of EDISTDSGD. There is one index entry for each EDISTDSGD entry. SCDUXNUM = SCDUNUM
EDISTDSGDY	ALL/DB2 index	Standard Data Element Usage Unique Index. An index created on the data element ID of EDISTDSGD. There is one index entry for each EDISTDSGD entry. SCDUYNUM = SCDUNUM
EDISTDSGDZ	ALL/DB2 index	Standard Data Element Usage Unique Index. An index created on the segment and standard ID of EDISTDSGD. There is one index entry for each EDISTDSGD entry. SCDUZNUM = SCDUNUM

Table 238. Standards database records (continued)

Record name	Product/ Type	Description
EDISTDSGH	ALL/DB2 table	Standard Segment Definition. Provides information about a segment within a standard. One entry exists for each segment defined in each standard. These are some of the standards shipped by WebSphere Data Interchange: **EDI902**　　　60 **TDCC28**　　　514 **UCSV3R1**　　155 **X12V2R2**　　400 **X12V2R3**　　433 **X12V3R4**　　464 **X12V3R1**　　520 **E**　　　　　6 **I**　　　　　6 **T**　　　　　6 **U**　　　　　6 **X**　　　　　6 SCSGNUM = one of the above values or your own estimate
EDISTDSGHX	ALL/DB2 index	Standard Segment Definition Unique Index. A unique index created on the key value to EDISTDSGH. There is one index entry for each EDISTDSGH entry. SCSGXNUM = SCSGNUM
EDISTDSGN	ALL/DB2 table	Standard Segment Note Definition. Provides information about segment notes within a standard. One entry exists for each segment note associated with a standard segment. SCSNNUM = your own estimate
EDISTDSGNX	ALL/DB2 index	Standard Segment Note Unique Index. A unique index created on the key value to EDISTDSGN. There is one index entry for each EDISTDSGN entry. SCSNXNUM = SCSNNUM
EDISTDSTH	ALL/DB2 table	Standard Definition. Defines the name, version, release, and default envelope type that should be used for a particular standard. One entry exists for each EDI standard and for each Envelope standard defined on the system. In most cases at least two of the X, E, I, T, or U envelope standards will be installed, plus any EDI standards that are applied or copied. **Note:** If you apply or copy a standard, this action adds entries to all EDISTDxxx tables. SCSTNUM = Number of standards installed
EDISTDSTHX	ALL/DB2 index	Standard Definition Unique Index. A unique index created on the key value to EDISTDSTH. There is one index entry for each EDISTDSTH entry. SCSTXNUM = SCSTNUM

Space requirements for tables and files

Table 238. Standards database records (continued)

Record name	Product/ Type	Description
EDISTDTXD	ALL/DB2 table	Standard Segment Usage. Records usage of a segment within a transaction. There is one entry for each time a segment is used within a transaction. These are some of the standards shipped by WebSphere Data Interchange: **EDI902** 15 **TDCC28** 127 **UCSV3R1** 32 **X12V2R2** 19 **X12V2R3** 25 **X12V3R4** 28 **X12V3R1** 39 SCSUNUM = one of the above values or your own estimate
EDISTDTXDX	ALL/DB2 index	Standard Segment Usage Unique Index. A unique index created on the key value to EDISCSU. There is one index entry for each EDISCSU entry. SCSUXNUM = SCSUNUM
EDISTDTXDY	ALL/DB2 index	Standard Segment Usage Index. An index created on the key value to segment and standard ID of EDISTDTXD. There is one index entry for each EDISTDTXD entry. SCSUYNUM = SCSUNUM
EDISTDTXDZ	ALL/DB2 index	Standard Segment Usage Index. An index created on the segment value of EDISTDTXD. There is one index entry for each EDISTDTXD entry. SCSUZNUM = SCSUNUM
EDISTDTXH	ALL/DB2 table	Standard Transaction Definition. Provides information about a transaction within a standard. There is one entry for each transaction defined in a standard. These are some of the standards shipped by WebSphere Data Interchange: **EDI902** **TDCC28** **UCSV3R1** **X12V2R2** **X12V2R3** **X12V3R4** **X12V3R1** SCTXNUM = one of the above values or your own estimate
EDISTDTXHX	ALL/DB2 index	Standard Transaction Definition Unique Index. A unqiue index created on the key value to EDISTDTXH. There is one index entry for each EDISTDTXH entry. SCTXXNUM = SCTXNUM
EDISTDTXHY	ALL/DB2 index	Standard Transaction Definition Index. An index created on the standard ID value of EDISTDTXH. There is one index entry for each EDISTDTXH entry. SCTXYNUM = SCTXNUM
EDISTDTXN	ALL/DB2 table	Standard Transaction Note Definition. Provides information about transaction notes within a standard. There is one entry for each transaction note associated with a standard transaction. SCTNNUM = your estimate

Table 238. Standards database records (continued)

Record name	Product/ Type	Description
EDISTDTXNX	ALL/DB2 index	Standard Transaction Note Unique Index. A unique index created on the key value to EDISTDTXN. There is one index entry for each EDISTDTXN entry. SCTNXNUM = SCTNNUM
EDISTDENV	ALL/DB2 table	Standard Envelope Standard Definition. Provides enveloping information about a standard. There is one entry for each envelope standard. SCEVNUM = your estimate
EDISTDENVX	ALL/DB2 index	Standard Envelope Standard Unique Index. A unique index created on the key value to EDISTDENV. There is one index entry for each EDISTDENV entry. SCEVXNUM = SCEVNUM

Table 239. Maps database records

Record name	Product/ Type	Description
EDIMAPHEAD	ALL/DB2 table	Map Header. Contains information relative to the mapping between an data format and a standard transaction. There is one entry for each map created. A good starting point for this number might be the number of data formats (EDIADFHEAD) that you have plus the number of standard transactions that you are interested in (EDISTDTXH). TPTXNUM = Number of mappings
EDIMAPHEADX	ALL/DB2 index	Map Header Unique Index. A unique index created on the key value to EDIMAPHEAD. There is one index entry for each EDIMAPHEAD entry. TPTXXNUM= TPTXNUM
EDIMAPAPPLCNTL	ALL/DB2 table	Map Application Control. Contains information about Map AC fields. There is from one to seven rows for each map created. TPACNUM = Number of AC records
EDIMAPAPPLCNTLX	ALL/DB2 index	Map Application Control Unique Index. A unique index created on the key value to EDIMAPAPPLCNTL. There is one index entry for each EDIMAPAPPLCNTL entry. TPACXNUM= TPACNUM
EDIMAPAPPLCNTLY	ALL/DB2 index	Map Application Control Index. An index created on the map ID value of EDIMAPAPPLCNTL. There is one index entry for each EDIMAPAPPLCNTL entry. TPACZNUM= TPACNUM

Space requirements for tables and files

Table 239. Maps database records (continued)

Record name	Product/ Type	Description
EDIMAPSEG	ALL/DB2 table	Map Segment Usage. Copied from EDISTDSGH and EDISTDSGD entries and keeps track of the use of a segment in the mapping. There is one of these for each mapped segment defined in the transaction and one for each segment selected in a repeated mapping. SSSTD = Average number of segments in a transaction (mapped and not mapped) SSRMAP = Average number of segments MAPPED in a REPEATED mapping TPSGNUM = TPTXNUM * (SSSTD + SSRMAP)
EDIMAPSEGX	ALL/DB2 index	Map Segment Usage Unique Index. An index created on key value to EDIMAPSEG. There is one index entry for each EDIMAPSEG entry. TPSGXNUM= TPSGNUM
EDIMAPSEGY	ALL/DB2 index	Map Segment Usage Index. An index created on the Map value of EDIMAPSEG. There is one index entry for each EDIMAPSEG entry. TPSGYNUM= TPSGNUM
EDIMAPSEGZ	ALL/DB2 index	Map Segment Usage Index. An index created on the HL key values of EDIMAPSEG. There is one index entry for each EDIMAPSEG entry. TPSGZNUM= TPSGNUM
EDIMAPELE	ALL/DB2 table	Map Data Element Definition. Copy of key information in EDISTDDEH and EDISTDSGD entries made when a data element is mapped. To estimate the number of entries, multiply the average number of segments that are mapped by the average number of fields per segment to get the average number of EDIMAPELE entries per EDIMAPHEAD entry. SSMAP = Average number of MAPPED segments per map ELSS = Average number of elements per selected segment (mapped or not mapped) TPDDNUM = TPTXNUM * SSMAP * ELSS
EDIMAPELEX	ALL/DB2 index	Map Data Element Definition Unique Index. A unqiue index created on the key value to EDIMAPELE. There is one index entry for each EDIMAPELE entry. TPDDXNUM= TPDDNUM
EDIMAPELEY	ALL/DB2 index	Map Data Element Definition Index. An index created on the map occurrence value to EDIMAPELE. There is one index entry for each EDIMAPELE entry. TPDDYNUM= TPDDNUM
EDIMAPGBLVAR	ALL/DB2 table	Map Global Variables. TPGVNUM = Number of global variables used
EDIMAPGBLVARX	ALL/DB2 index	Map Global Variables Unique Index, A unique index created on the key value to EDIMAPGBLVAR. There is one index entry for each EDIMAPGBLVAR entry. TPGVXNUM = TPGVNUM

Table 239. Maps database records (continued)

Record name	Product/ Type	Description
EDIMAPSYNTAX	ALL/DB2 table	Map Syntax. TPSYNUM = Number of syntax rows
EDIMAPSYNTAX	ALL/DB2 index	Map Syntax.Unique Index. A unique index created on the key value to EDIMAPSYNTAX. There is one index entry for each EDIMAPSYNTAX entry. TPSYXNUM = TPSYNUM
EDIMAPLCLVAR	ALL/DB2 table	Map Local Variables. TPLVNUM = Number of local variable entries
EDIMAPLCLVAR	ALL/DB2 index	Map Local Variables Unique Index. A unique index created on the key value to EDIMAPLCLVAR. There is one index entry for each EDIMAPLCLVAR entry. TPLVXNUM = TPLVNUM
EDIMAPREF	ALL/DB2 table	Map References. TPRFNUM = Number of map references
EDIMAPREFX	ALL/DB2 index	Map References Unique Index. A unique index created on the key value to EDIMAPREF. There is one index entry for each EDIMAPREF entry. TPRFXNUM = TPRFNUM
EDIMAPNODES	ALL/DB2 table	Map Nodes. TPMNNUM = Number of map node entries
EDIMAPNODESX	ALL/DB2 index	Map Nodes Unique Index. A unique index created on the key value to EDIMAPNODES. There is one index entry for each EDIMAPNODES entry. TPMNXNUM = TPMNNUM
EDIMAPCMDS	ALL/DB2 table	Map Commands. Contains information about mapping commands. There is one entry for each map command created. TPCMDNUM = Number of map commands
EDIMAPCMDSX	ALL/DB2 index	Map Commands Unique Indesx. A unique index created on the key value to EDIMAPCMDS. There is one index entry for each EDIMAPCMDS entry. TPCMDXUM = TPCMDNUM
EDIRULE	ALL/DB2 table	Trading Partner Rules. Keeps track of how TRANSFORM maps are used and execution options of same. See EDITPRT and EDITPST, which are the same for SEND/RECEIVE maps. TPRUNUM = your estimate
EDIRULEX	ALL/DB2 index	Trading Partner Rules Unique Index. A unique index created on the key value to EDIRULE. There is one index entry for each EDIRULE entry. TPRUXNUM = TPRUNUM

Space requirements for tables and files

Table 239. Maps database records (continued)

Record name	Product/ Type	Description
EDIRULE1	ALL/DB2 index	Trading Partner Rules Index. An index created on the Format values to EDIRULE. There is one index entry for each EDIRULE entry. TPRU1NUM = TPRUNUM
EDIRULE2	ALL/DB2 index	Trading Partner Rules Index. An index created on the Sending TP value to EDIRULE. There is one index entry for each EDIRULE entry. TPRU2NUM = TPRUNUM
EDIRULE3	ALL/DB2 index	Trading Partner Rules Index. An index created on the Receiving TP value to EDIRULE. There is one index entry for each EDIRULE entry. TPRU3NUM = TPRUNUM
EDITPRT	ALL/DB2 table	Map Receive Usage. Records the use of a mapping by a particular trading partner. This is only used for transactions being received: there is one entry for each trading partner using a particular transaction. TPTXNUM = Total number of mapppings PRECV = Percentage of mappings (TPTXNUM) that are receive mappings TPPROFNUM = Total number of trading partners TPRECV = Percentage of tradings partners you receive data from TPRTNUM = (TPTXNUM * PRECV) * (TPPROFNUM * TPRECV)
EDITPRTX	ALL/DB2 index	Map Receive Usage Unique Index. A unique index created on the key value in EDITPRT. There is one index entry for each EDITPRT entry. TPRTXNUM = TPRTNUM
EDITPRTY	ALL/DB2 index	Map Receive Usage Index. An index created on the std transaction and tp nickname values of EDITPRT. There is one entry for each EDITPRT entry. TPRTYNUM = TPRTNUM
EDITPRTZ	ALL/DB2 index	Map Receive Usage Index. An index created on the TP nickname values of EDITPRT. There is one entry for each EDITPRT entry. TPRTZNUM = TPRTNUM

Table 239. Maps database records (continued)

Record name	Product/ Type	Description
EDITPST	ALL/DB2 table	Map Send Usage. Records the use of a mapping by a particular trading partner. Used only for transactions being sent. There is one entry for each trading partner using this transaction with a given data format ID and internal trading partner ID. Usually a single internal trading partner ID is defined per trading partner. TPTXNUM = Total number of mappings PSEND = Percentage of mappings (TPTXNUM) that are send mappings TPPROFNUM = Total number of trading partners TPSEND = Percentage of trading partners you send to TPSTNUM = (TPTXNUM * PSEND) * (TPPROFNUM * TPSEND)
EDITPSTX	ALL/DB2 index	Map Send Usage Unique Index. A unique index created on the key value in EDITPST. There is one index entry for each EDITPST entry. TPSTXNUM = TPSTNUM
EDITPSTY	ALL/DB2 index	Map Send Usage Unique Index. An index created on the TP nickname of EDITPST. There is one index entry for each EDITPST entry. TPSTYNUM = TPSTNUM
EDITPSTZ	ALL/DB2 index	Map Send Usage Unique Index. An index created on the internal TP ID value of EDITPST. There is one index entry for each EDITPST entry. TPSTZNUM = TPSTNUM

Table 240. Data formats database records

Record name	Product/ Type	Description
EDIADFDICT	ALL/DB2 table	Application Format Dictionary Definition. Contains information about a collection (dictionary) of related data format objects. There is one entry for each dictionary TDDICT = your estimate
EDIADFDICTX	ALL/DB2 index	Application Format Dictionary Unique Index. A unique index created on the key value of EDIADFDICT. There is one index entry for each EDIADFDICT entry. TDDICTX = TDDICT
EDIADFRECIDINFO	ALL/DB2 table	Application Format Record ID Information. Contains information about the location and characteristics of the Record ID of a data format. There is one entry for each unique location of a data format record ID TDRECID = your estimate

Table 240. Data formats database records (continued)

Record name	Product/ Type	Description
EDIADFRECIDINFO	ALL/DB2 index	Application Format Record ID Information Unique Index. A unique index created on the key value of EDIADFRECIDINFO. There is one index entry for each EDIADFRECIDINFO entry. TDRECIDX = TDRECID
EDIADFHEADER	ALL/DB2 table	Application Format Definition. Contains information about an data format. There is one entry for each data format defined. An estimate of the number of input and output data definitions contained in the applications that are going to be EDI enabled. This corresponds to a 01 level-number in COBOL. TDIDNUM = Number of data formats
EDIADFHEADERX	ALL/DB2 index	Application Format Definition Unique Index. A unique index created on the key value of EDIADFHEADER. There is one index entry for each EDIADFHEADER entry. TDIDXNUM = TDIDNUM
EDIADFHDRMEM	ALL/DB2 table	Application Format Detail. One row exists for each loop, record, or structure within the header. TDHDRM = your estimate
EDIADFHDRMEMX	ALL/DB2 index	Application Format Detail Unique Index. A unique index created on the key value of EDIADFHDRMEM. There is one index entry for each EDIADFHDRMEM entry. TDHDRMX = TDHDRM
EDIADFHDRMEMY	ALL/DB2 index	Application Format Detail Index. An index created on the key value of EDIADFHDRMEM. There is one index entry for each EDIADFHDRMEM entry. TDHDRMY = TDHDRM
EDIADFLOOP	ALL/DB2 table	Application Loop Definition. There is one entry for each loop required by the data format. TDLOOP = your estimate
EDIADFLOOPX	ALL/DB2 index	Application Loop Definition Unique Index. A unique index created on the key value to EDIADFLOOP. There is one index entry for each EDIADFLOOP entry. TDLOOPX = TDLOOP
EDIADFLOOPY	ALL/DB2 index	Application Loop Definition Index. An index created on the dictionary value of EDIADFLOOP. There is one index entry for each EDIADFLOOP entry. TDLOOPY = TDLOOP
EDIADFLOOPZ	ALL/DB2 index	Application Loop Definition Index. An index created on the recordidinfo value of EDIADFLOOP. There is one index entry for each EDIADFLOOP entry. TDLOOPZ = TDLOOP

Table 240. Data formats database records (continued)

Record name	Product/ Type	Description
EDIADFLOOPMEM	ALL/DB2 table	Application Loop Detail. There is one entry for each loop or record within the Loop TDLOOPM = your estimate
EDIADFLOOPMEMX	ALL/DB2 index	Application Loop Detail Unique Index. A unique index created on the key value to EDIADFLOOPMEM. There is one index entry for each EDIADFLOOPMEM entry. TDLOOPMX = TDLOOPM
EDIADFRECORD	ALL/DB2 table	Application Record Definition. There is one entry for each record format defined. An estimate of the number of input/output data definitions contained in the applications that are going to be EDI enabled. This corresponds to a 01 level-number in COBOL. TDREC = your estimate
EDIADFRECORDX	ALL/DB2 index	Application Record Definition Unique Index. A unique index created on the key value to EDIADFRECORD. There is one index entry for each EDIADFRECORD entry. TDRECX = TDREC
EDIADFRECORDY	ALL/DB2 index	Application Record Definition Unique Index. A unique index created on the key value to EDIADFRECORD. There is one index entry for each EDIADFRECORD entry. TDRECY = TDREC
EDIADFRECORDZ	ALL/DB2 index	Application Record Definition Unique Index. A unique index created on the key value to EDIADFRECORD. There is one index entry for each EDIADFRECORD entry. TDRECZ = TDREC
EDIADFRECMEM	ALL/DB2 table	Application Record Detail Definition. There is one entry for each record element of a record. An entry is created for each structure and field used to describe the record. TDRECM = your estimate
EDIADFRECMEMX	ALL/DB2 index	Application Record Detail Unique Index. A unique index created on the key value to EDIADFRECMEM. There is one index entry for each EDIADFRECMEM entry. TDRECMX = TDRECM
EDIADFRECMEMY	ALL/DB2 index	Application Record Detail Index. A unique index created on the key value to EDIADFRECMEM. There is one index entry for each dictionary name and record name value in an EDIADFRECMEM entry. TDRECMY = TDRECM

Space requirements for tables and files

Table 240. Data formats database records (continued)

Record name	Product/ Type	Description
EDIADFSTRUCT	ALL/DB2 table	Application Structure Definition. There is one entry for each structure defined in the dictionary. TDSTR = your estimate
EDIADFSTRUCTX	ALL/DB2 index	Application Structure Definition Unique Index. A unique index created on the key value to EDIADFSTRUCT. There is one index entry for each EDIADFSTRUCT entry. TDSTRX = TDSTR
EDIADFSTRUCTY	ALL/DB2 index	Application Structure Definition Index. An index created on the dictionary name, structure name value of EDIADFSTRUCT. There is one index entry for each EDIADFSTRUCT entry. TDSTRY = TDSTR
EDIADFSTRUCTMEM	ALL/DB2 table	Application Structure Detail Definition. There is one entry for each structure element of a structure. An entry is created for each subordinate structure and field used to describe the structure. TDSTRM = your estimate
EDIADFSTRUCTMEMX	ALL/DB2 index	Application Structure Detail Unique Index. A unique index created on the key value to EDIADFSTRUCTMEM. There is one index entry for each EDIADFSTRUCTMEM entry. TDSTRMX = TDSTRM
EDIADFSTRUCTMEMY	ALL/DB2 index	Application Structure Detail Index. An index created on the dictionary name, structure name value to EDIADFSTRUCTMEM. There is one index entry for each EDIADFSTRUCTMEM entry. TDSTRMY = TDSTRM
EDIADFFIELD	ALL/DB2 table	Application Field Definition. There is one entry for each field defined in the dictionary. TDFLD = your estimate
EDIADFFIELDX	ALL/DB2 index	Application Field Unique Index. A unique index created on the key value to EDIADFFIELD. There is one index entry for each EDIADFFIELD entry. TDFLDX = TDFLD
EDIADFFIELDY	ALL/DB2 index	Application Field Index. An index created on the dictionary name, structure name value to EDIADFFIELD. There is one index entry for each EDIADFFIELD entry. TDFLDY = TDFLD
EDIXMLDICT	ALL/DB2 table	XML Dictionary Definition. There is one entry for each XML Dictionary defined. An XML Dictionary is a collection of XML DTDs. XMLD = your estimate

Table 240. Data formats database records (continued)

Record name	Product/ Type	Description
EDIXMLDICTX	ALL/DB2 index	XML Dictionary Unique Index. A unique index created on the key value to EDIXMLDICT. There is one index entry for each EDIXMLDICT entry. XMLDX = XMLD
EDIDTDHDR	ALL/DB2 table	XML DTD Header Definition. There is one entry for each XML DTD defined in the dictionary. XMLDTD = your estimate
EDIDTDHDR	ALL/DB2 index	XML DTD Unique Index. A unique index created on the key value to EDIDTDHDR. There is one index entry for each EDIDTDHDR entry. XMLDTDX= XMLDTD
EDIDTD	ALL/DB2 table	XML DTD Detail. There is one entry for each line of an XML DTD. XMLDET = your estimate
EDIDTDX	ALL/DB2 index	XML DTD Detail Unique Index. A unique index created on the key value to EDIDTD. There is one index entry for each EDIDTD entry. XMLDETX= XMLDET

Table 241. Trading Partner database records

Record name	Product/ Type	Description
EDIPSTP	ALL/DB2 table	Trading Partner Profile. There is one entry for each trading partner. PSTPNUM = Number of trading partners
EDIPSTPX	ALL/DB2 index	Trading Partner Profile Unique Index. There is one index entry for each EDIPSTP entry. PSTPXNUM = PSTPNUM
EDIPSTP1	ALL/DB2 index	Trading Partner Profile Unique Index. There is one index entry for each EDIPSTP entry. PSTP1NUM = PSTPNUM
EDIPSTP2	ALL/DB2 index	Trading Partner Profile Unique Index. There is one index entry for each EDIPSTP. PSTP2NUM = PSTPNUM
EDIPSTP3	ALL/DB2 index	Trading Partner Profile UniqueIndex. There is one index entry for each EDIPSTP. PSTP3NUM = PSTPNUM
EDITPCM	ALL/DB2 table	Comment Table. There is one entry for each trading partner that has associated comment data entered through the WebSphere Data Interchange Client interface. If WebSphere Data Interchange Client is not used, this table is empty. TPCMNUM = PSTPNUM

Table 241. Trading Partner database records (continued)

Record name	Product/ Type	Description
EDITPCMX	ALL/DB2 index	Comment Table Unique Index. A unique index created on the key values in EDITPCM. There is one index entry for each EDITPCM entry. TPCMXNUM = TPCMNUM
EDITPCN	ALL/DB2 table	Trading Partner Comment Table. There is one entry for each Trading Partner/Contact relationship that has associated comment data entered through the WebSphere Data Interchange Client interface. If WebSphere Data Interchange Client is not used, this table is empty. TPCNNUM = PSTPNUM * average number of contacts per trading partner
EDITPCNX	ALL/DB2 index	Trading Partner Contact Table Unique Index. A unique index created on the key value in EDITPCN. There is one entry for each EDITPCN entry. TPCNXNUM = TPCNNUM
EDITPCT	ALL/DB2 table	Contact Table. There is one entry for each contact entered through the WebSphere Data Interchange Client interface. If WebSphere Data Interchange Client is not used, this table is empty. TPCTNUM = Number of contacts
EDITPCTX	ALL/DB2 index	Contact Table Unique Index. A unique index created on the key value in EDITPCT. There is one entry for each EDITPCT entry. TPCTXNUM = TPCTNUM
EDIPROF	ALL/DB2 table	Trading Partner Profile Control Numbers. There is one entry for each sender/receiver combination used. PROFNUM = number of different sender/receiver combinations used
EDIPROFX	ALL/DB2 index	Trading Partner Profile Control Number Table Index. There is one index entry for each sender/receiver pair combination used. PROFXNUM = PROFNUM

Table 242. Set up database records

Record name	Product/ Type	Description
EDIPSAC	ALL/DB2 table	Activity Log Profile. There is one index entry for each log file. WebSphere Data Interchange creates two log files; additional log files are created by the user. PSACNUM = 2 + number of user log files
EDIPSACX	ALL/DB2 index	Activity Log Unique Index. There is one index entry for each Activity Log Profile. PSACXNUM = PSACNUM
EDIPSAD	ALL/DB2 table	User Exit Information Profile. There is one entry for each user exit referenced. WebSphere Data Interchange creates two entries for its own use; additional entries are created by the user to identify user exits used by the installation. PSADNUM = 2 + number of user installation exit programs

Table 242. Set up database records (continued)

Record name	Product/ Type	Description
EDIPSADX	ALL/DB2 index	User Exit Profile Unique Index. There is one index entry for each EDIPSAD entry. PSADXNUM = PSADNUM
EDIPSAP	ALL/DB2 table	Application Defaults Profile. This profile is used to establish settings at an application level (as opposed to the system level) for an invocation of WebSphere Data Interchange. There is one index entry for each application ID. PSAPNUM = 2 + number of user-specified applications
EDIPSAPX	ALL/DB2 index	Application Definition Profile Unique Index. There is one index entry for each EDIPSAP entry. PSAPXNUM = PSAPNUM
EDIPSCR	ALL/DB2 table	Continuous Receive Profile. There is one entry for each unique path by which data is received from a VAN. PSCRNUM = Number of paths to VAN
EDIPSCRX	ALL/DB2 index	Continuous Receive Unique Index. There is one index entry for each EDIPSCR entry. PSCRXNUM = PSCRNUM
EDIPSDI	ALL/DB2 table	WebSphere Data Interchange Control File. This table is static.
EDIPSDIX	ALL/DB2 index	WebSphere Data Interchange Control File Unique Index. This table is static.
EDIPSEE	ALL/DB2 table	E Envelope Profile Table. There is one entry for each E envelope profile member (EDIFACT). PSEENUM = number of E envelope profile members
EDIPSEEX	ALL/DB2 index	E Envelope Profile Table Index. There is one entry for each E envelope profile member (EDIFACT). PSEEXNUM = PSEENUM
EDIPSEI	ALL/DB2 table	Export/Import Control File. This table is static.
EDIPSEIX	ALL/DB2 index	Export/Import Control File Unique Index. This table is static.
EDIPSIE	ALL/DB2 table	I Envelope Profile Table. There is one entry for each I envelope profile member (ICS). PSIENUM = Number of I envelope profile members
EDIPSIEX	ALL/DB2 index	I Envelope Profile Table Index. There is one index entry for each I envelope profile member (ICS). PSIEXNUM = PSIENUM

Space requirements for tables and files

Table 242. Set up database records (continued)

Record name	Product/ Type	Description
EDIPSLP	ALL/DB2 table	Language Profile. There is one entry for each language used: in this release only English (ENU) is supported. PSLPNUM = 1
EDIPSLPX	ALL/DB2 index	Language Profile Unique Index. There is one index entry for each EDIPSLP entry. PSLPXNUM = PSLPNUM
EDIPSMQ	ALL/DB2 table	MQSeries Profile Table. There is one index entry for each MQSeries profile member. PSMQNUM = number of MQSeries profile members
EDIPSMQX	ALL/DB2 index	MQSeries Profile Table Index. There is one index entry for each MQSeries profile member. PSMQXNUM = PSMQNUM
EDIPSNO	ALL/DB2 table	Network Commands Profile. There is one entry for each line of a Network Operation statement (or Network Command); WebSphere Data Interchange supplies several commands to support the Continuous Receive process. PSNONUM = 200 + number of user-supplied statements
EDIPSNOX	ALL/DB2 index	Network Operation Profile Log Unique Index. There is one index entry for each EDIPSNO entry. PSNOXNUM = PSNONUM
EDIPSNP	ALL/DB2 table	Network Profile. There is one entry for each network defined to the system; WebSphere Data Interchange supplies the definitions for eight networks. PSNPNUM = 8 + number of additional networks
EDIPSNPX	ALL/DB2 index	Network Profile Unique Index. There is one index entry for each EDIPSNP entry. PSNPXNUM = PSNPNUM
EDIPSPD	ALL/DB2 table	Profile Definitions. This table contains the definitions of the WebSphere Data Interchange profiles. This is a static file and does not change in size. PSPDNUM = number of profile definitions
EDIPSPDX	ALL/DB2 index	Profile Definitions Unique Index. There is one index entry for each EDIPSPD entry. PSPDXNUM = PSPDNUM
EDIPSRQ	ALL/DB2 table	Mailbox or Requestor Profile. There is one entry for each mailbox used to receive data. PSRQNUM = number of mailboxes
EDIPSRQX	ALL/DB2 index	Requestor Profile Unique Index. There is one index entry for each EDIPSRQ entry. PSRQXNUM = PSRQNUM

Table 242. Set up database records (continued)

Record name	Product/ Type	Description
EDIPSSP	ALL/DB2 table	Security Profile. There is one entry for each security processing grouping. The names of the exit programs to be used for encryption, authorization, filtering, and compression are specified in a named security profile. PSSPNUM = number of security processing groups
EDIPSSPX	ALL/DB2 index	Security Profile Unique Index. There is one index entry for each EDIPSSP entry. PSSPXNUM = PSSPNUM
EDIPSSY	ALL/DB2 table	System Profile. There is one entry for each group of settings of a CICS Persistent Environment. Persistent Environment provides a means of improving performance in CICS. PSSYNUM = Number of setting groupings
EDIPSSYX	ALL/DB2 index	System Profile Unique Index. There is one index entry for each EDIPSSY entry. PSSYXNUM = PSSYNUM
EDIPSTD	ALL/DB2 table	Validation and Translation Table Definitions. Contains the definitions of validation and translation tables. This table is subject to large additions when new standards are applied to your system, or when new applications are added. There is one entry for each validation and translation table in WebSphere Data Interchange. WebSphere Data Interchange provides 49 tables at installation; importing standards increases the number of tables by the number of Code Lists that accompany the Standard. PSTDNUM = 49 + user-created or imported tables
EDIPSTDX	ALL/DB2 index	Table Definitions Unique Index. There is one index entry for each EDIPSTD entry. PSTDXNUM = PSTDNUM
EDIPSTE	ALL/DB2 table	T Envelope Profile Table. There is one entry for each T envelope profile member (UN/TDI). PSTENUM = number of T envelope profile members
EDIPSTEX	ALL/DB2 index	T Envelope Profile Table Index. There is one entry for each T envelope profile member (UN/TDI). PSTEXNUM = PSTENUM
EDIPSTT	ALL/DB2 table	Translation Table. There is one entry for each code of a translation table. WebSphere Data Interchange supplies 953 values in the 49 tables loaded at installation. This table is subject to quite large additions when new standards are applied to your system or when new applications are added. PSTTNUM = (Average number of values * PSTDNUM) + 953
EDIPSTTX	ALL/DB2 index	Translation Table Entry Unique Index. There is one index entry for each EDIPSTT entry. PSTTXNUM = PSTTNUM

Space requirements for tables and files

Table 242. Set up database records (continued)

Record name	Product/ Type	Description
EDIPSTT1	ALL/DB2 index	Translation Table Entry Unique Index. There is one index entry for each EDIPSTT entry. PSTT1NUM = PSTTNUM
EDIPSTV	ALL/DB2 table	Validation Table. There is one entry for each validation table. WebSphere Data Interchange supplies three tables at installation with a total number of 294 values. This table is subject to quite large additions when new standards are applied to your system or when new applications are added. PSTVNUM = (Average number of values *PSTDNUM) + 294
EDIPSTVX	ALL/DB2 index	Validation Table Entry Unique Index. There is one index entry for each EDIPSTV entry. PSTVXNUM = PSTVNUM
EDIPSUE	ALL/DB2 table	U Envelope Profile Table. There is one entry for each type U envelope profile member (UCS). PSUENUM = number of type U envelope profile members
EDIPSUEX	ALL/DB2 index	U Envelope Profile Table Index. There is one index entry for each type U envelope profile member (UCS). PSUEXNUM = PSUENUM
EDIPSXE	ALL/DB2 table	X Envelope Profile Table. There is one entry for each X envelope profile member (X12). PSXENUM = number of type X envelope profile members
EDIPSXEX	ALL/DB2 index	X Envelope Profile Table Index. There is one index entry for each type X envelope profile member (X12). PSXEXNUM = PSXENUM

Transaction Store tables

The tables listed in Table 243 on page 771 are all part of the Transaction Store. The number of entries in the Transaction Store depends on two major factors:

1. The length of time that transactions remain in the database before being purged. You can specify this value when you add transactions to the Transaction Store. The default is 30 days.

2. The elapsed number of days between runnings of the Transaction Store remove utility.

TSLIFE = TRXLIFE + PGRSPAN where:

TRXLIFE   The number of days before a transaction might be purged
PGRSPAN   The number of days between running the purge utility

Along with TRXLIFE and PRGSPAN, the other important number to estimate is the number of transactions per day that you will be processing. The examples below are based on knowing these three pieces of information.

Table 243. Transaction Store tables

Record name	Product/ Type	Description
EDITSTH	ALL/DB2 Table	Transaction Store Transaction Handle. Contains detailed information relative to a transaction. There is one entry for each transaction send translated or de-enveloped. TRXPDAY = Number of transactions per day TSTHNUM = TRXPDAY * TSLIFE
EDITSTHX	ALL/DB2 Index	Transaction Store Transaction Handle Unique Index. A unique index created on the key value to EDITSTH. There is one index entry for each EDITSTH entry. TSTHXNUM = TSTHNUM
EDITSTHO	ALL/DB2 Index	Transaction Store Table Unique Index. TSTHONUM = TSTHNUM
EDITSTI	ALL/DB2 Table	Transaction Store Transaction Image. Contains the standard transaction image for a transaction. There will be one entry for each EDITSTH entry. TSTINUM = TSTHNUM
EDITSTIX	ALL/DB2 Index	Transaction Store Transaction Image Unique Index. A unique index created on the key value to EDITSTI. There is one index entry for each EDITSTI entry. TSTIXNUM = TSTINUM
EDITSTO	ALL/DB2 Table	Transaction Store Transaction Override. Contains the envelope override values for a particular set of transactions. This only applies to send transactions, and only for those transactions that supply override values in the C record or through the API. If overrides are not used, the number is **0**. If overrides are used, at least one record is created for each PERFORM TRANSLATE TO STANDARD command executed. If the override values change, one record is created each time an override value changes from one transaction to the next. TSTONUM = 0
EDITSTOX	ALL/DB2 Index	Transaction Store Transaction Override Unique Index. A unique index created on the key value to EDITSTO. There is one index entry for each EDITSTO entry. TSTOXNUM = TSTONUM
EDITSAU	ALL/DB2 Table	Transaction Store Application Usage. Contains information relative to the translation of a transaction by an application. There is one entry for each EDITSTH entry unless there are many transactions that are de-enveloped but never translated. TSAUNUM = TSTHNUM

Space requirements for tables and files

Table 243. Transaction Store tables (continued)

Record name	Product/ Type	Description
EDITSAUX	ALL/DB2 Index	Transaction Store Application Usage Unique Index X. A unique index created on the key value to EDITSAU. There is one index entry for each EDITSAU entry. TSAUXNUM = TSAUNUM
EDITSAUY	ALL/DB2 Index	Transaction Store Application Usage Index Y. An index created on the BATCHID value used to retrieve transactions. There is one entry for each EDITSAU entry with a different BATCHID value. TSAUYNUM = TSAUNUM
EDITSEV	ALL/DB2 Table	Transaction Store Transaction Envelope. Contains information about an interchange. There is one entry created every time an envelope or de-envelope function is requested. TRXPENV = Average number of transactions per envelope TSEVNUM = TSTHNUM / TRXPENV
EDITSEVX	ALL/DB2 Index	Transaction Store Transaction Envelope Unique Index. A unique index created on the key value to EDITSEV. There is one index entry for each EDITSEV entry. TSEVXNUM = TSEVNUM
EDITSEV1	ALL/DB2 Table	Transaction Store Envelope Unique Index. TSEV1NUM = TSEVNUM
EDITSGP	ALL/DB2 Table	Transaction Store Group. Maintains information about a functional group. There is at least one entry for each interchange sent or received. GRPPENV = Average number of groups per envelope (minimum of 1) TSGPNUM = TSEVNUM * GRPPENV
EDITSGPX	ALL/DB2 Index	Transaction Store Group Unique Index. A unique index created on the key value to EDITSGP. There is one index entry for each EDITSGP entry. TSGPXNUM = TSGPNUM
EDITSTU	ALL/DB2 Table	Transaction Store Transaction Usage. Contains information about the use of a transaction in an interchange. There is one entry each time a transaction is added to or extracted from an interchange. A good assumption is one entry for each EDITSTH entry unless there are a large number of transactions with translation errors or transactions that were never enveloped for some other reason. TSTUNUM = TSTHNUM
EDITSTUX	ALL/DB2 Index	Transaction Store Transaction Usage Unique Index X. A unique index created on the key value to EDITSTU. There is one index entry for each EDITSTU entry. TSTUXNUM = TSTUNUM

Table 243. Transaction Store tables (continued)

Record name	Product/ Type	Description
EDITSTUY	ALL/DB2 Index	Transaction Store Transaction Usage Index Y. An index created on the transaction handle value used to keep track of which transaction is being enveloped. A good assumption is that there is one index entry for for each EDITSTU entry unless a large amount of REENVELOPE activity occurs. TSTUYNUM = TSTUNUM
EDITSLT	ALL/DB2 table	Transaction Store Lock Table. Contains a single record that is used as a lock table when update activity involves the Transaction Store. TSLTNUM = 1

Management Reporting Tables

The database records in Table 244 on page 774 are all part of Management Reporting. Management Reporting database records are defined for two categories:

- Measurement records reflect statistics current as of the effective date of the last PERFORM UPDATE STATISTICS command that was processed.
- Pending records contain statistic entries created by WebSphere Data Interchange processes, but have not yet been updated to the measurement records by the PERFORM UPDATE STATISTICS command.

The number of entries in the Measurement database is dependent on two major factors:

1. MEASLIFE: the number of days that entries are stored in the measurement database before they are purged during the PERFORM REMOVE STATISTICS process.
2. MEASPDAY: the number of statistic entries that are written to the measurement database per EDI processing day.

The number of entries in the Pending database is dependent on two major factors:

1. PENDLIFE: the number of days that entries are stored in the pending database before they are purged during the PERFORM UPDATE STATISTICS process.
2. PENDPDAY: the number of statistic entries that are written to the pending database per EDI processing day.

The following examples are based on knowing these four pieces of information.

Space requirements for tables and files

Table 244. Management reporting tables

Record name	Product/ Type	Description
EDIMRCM	ALL/DB2 table	Management Reporting Communications Measurement. Contains measurement information relative to an interchange. For each active Requestor ID, you can have two cumulative entries (send and receive). Also, each Requestor ID can have a maximum of two daily entries (one send and one receive) per EDI processing day. If a Requestor ID has multiple interchanges during a day, the daily entry created for that day will be updated for each interchange. RIDNUM = Number of Requestor IDs (RID) CUMENTS = Number of cumulative entries = RIDNUM * 2 PRIDINT = Daily percentage of RIDs having an interchange MEASPDAY = (RIDNUM * 2) * PRIDINT MRCMNUM = (MEASPDAY * MEASLIFE) + CUMENTS
EDIMRCMX	ALL/DB2 index	Management Reporting Communications Measurement Unique Index. A unique index created on the key value to EDIMRCM. There is one index entry for each EDIMRCM entry. MRCMXNUM = MRCMNUM
EDIMRPC	ALL/DB2 table	Management Reporting Pending Communications. Contains pending measurement information relative to an interchange. When executing the PERFORM UPDATE STATISTICS command, this data is applied to the EDIMRCM database and then is purged. There is no cumulative record concept in the pending database, therefore for each active Requestor ID, you can have multiple send and receive entries per EDI processing day. RIDNUM = Number of Requestor ID's (RID) SENDNUM = Average number of send interchanges per RID per day RECVNUM = Average number of receive interchanges per RID per day PENDPDAY = (SENDNUM + RECVNUM) * RIDNUM MRPCNUM = PENDPDAY * PENDLIFE
EDIMRPCX	ALL/DB2 index	Management Reporting Pending Communications Unique Index. A unique index created on the key value to EDIMRPC. There is one index entry for each EDIMRPC entry. MRPCXNUM = MRPCNUM

Table 244. Management reporting tables (continued)

Record name	Product/ Type	Description
EDIMRRT	ALL/DB2 table	Management Reporting Receive Usage Measurement. Contains measurement information relative to a receive usage. For each active receive usage you can have one cumulative entry, plus one daily entry per EDI processing day. This table is directly related to the EDITPRT table. Use the table entry to see how to calculate TPRTNUM, total number of receive usages. The following calculations assume you will have at least one transaction per active receive usage (EDITPRT entry) per day. PACTIVE = Daily percentage of receive usages having activity MEASPDAY = TPRTNUM * PACTIVE MRRTNUM = (MEASPDAY * MEASLIFE) + TPRTNUM
EDIMRRTX	ALL/DB2 index	Management Reporting Receive Usage Measurement Unique Index. A unique index created on the key value to EDIMRRT. There is one index entry for each EDIMRRT entry. MRRTXNUM = MRRTNUM

Space requirements for tables and files

Table 244. Management reporting tables (continued)

Record name	Product/ Type	Description
EDIMRPR	ALL/DB2 table	Management Reporting Pending Receive Usage. Contains pending measurement information about a receive usage. When PERFORM UPDATE STATISTICS is executed, this data is added to the EDIMRRT database, and then purged from the Pending database. Because there is no cumulative record function in the Pending database, there can be multiple receive entries per EDI processing day for each active receive usage. Estimating the size of this table can be difficult because its size depends on the RECOVERY scope specified during translation. With transaction recovery scope, table size is easier to estimate because you have an EDIMRPR entry for each received transaction. With interchange recovery scope, you have one entry for each receive usage that is used within the interchange. For example, suppose that an interchange contains 100 transactions. With transaction recovery scope, 100 EDIMRPS entries would be created. With interchange recovery scope, there can be as few as 1 EDIMRPR entry (all transactions have the same receive usage) or as many as 100 EDIMRPR entries (each transaction has a different receive usage). **Transaction level recovery formula:** TRXPDAY = Number of transactions per day. This value is also used in space calculations for EDITSTH PRECV = Percentage of transactions that are receive transactions MRPRNUM = TRXPDAY * PRECV * PENDLIFE **Interchange level recovery formula:** TRXPDAY = Number of transactions per day. This value is also used in space calculations for EDITSTH PRECV = Percentage of transactions that are receive transactions TRXPENV = Average number of transactions per interchange. This value is also used in space calculations for EDITSEV ENVPDAY = Number of interchanges received per day = (TRXPDAY*PRECV) / TRXPENV AVGUSGS = Average number of usages per interchange MRPRNUM = ENVPDAY * AVGUSGS * PENDLIFE
EDIMRPRX	ALL/DB2 index	Management Reporting Pending Receive Usage Unique Index. A unique index created on the key value to EDIMRPR. There is one index entry for each EDIMRPR entry. MRPRXNUM = MRPRNUM

Table 244. Management reporting tables (continued)

Record name	Product/ Type	Description
EDIMRST	ALL/DB2 table	Management Reporting Send Usage Measurement. Contains measurement information relative to send usage of a trading partner mapping transaction ID. For each active send usage you can have one cumulative entry, plus one daily entry per EDI processing day. This table is directly related to the EDITPST table. Use the table entry to see how to calculate TPSTNUM, total number of send usages. The following calculations assume you will have at least one transaction per active send usage (EDITPST entry) per day. PACTIVE = daily percentage of send usages having activity MEASPDAY = TPSTNUM * PACTIVE MRSTNUM = (MEASPDAY * MEASLIFE) + TPSTNUM
EDIMRSTX	ALL/DB2 index	Management Reporting Send Usage Measurement Unique Index. A unique index created on the key value to EDIMRST. There is one index entry for each EDIMRST entry. MRSTXNUM = MRSTNUM

Space requirements for tables and files

Table 244. Management reporting tables (continued)

Record name	Product/ Type	Description
EDIMRPS	ALL/DB2 table	Management Reporting Pending Send Usage. Contains pending measurement information relative to a send usage. When executing the PERFORM UPDATE STATISTICS command, this data is applied to the EDIMRST database and then is purged. There is no cumulative record concept in the pending database, therefore for each active send usage, you can have multiple send entries per EDI processing day. Estimating the size of this table can be difficult because its size depends on the RECOVERY scope specified during translation. With transaction recovery scope, table size is easier to estimate because there is an EDIMRPS entry for each send transaction. With interchange recovery scope, there is one entry for each send usage used within the interchange. For example, suppose that an interchange contains 100 transactions. With transaction recovery scope, 100 EDIMRPS entries would be created. With interchange recovery scope, there can be as few as 1 EDIMRPS entry (all transactions have the same send usage) or as many as 100 EDIMRPS entries (each transaction has a different send usage). **Transaction level recovery formula:** TRXPDAY = Number of transactions per day. This value is also used in space calculations for EDITSTH. PSEND = Percentage of transactions that are send transactions. MRPSNUM = TRXPDAY * PSEND * PENDLIFE **Interchange level recovery formula:** TRXPDAY = Number of transactions per day. This value is also used in space calculations for EDITSTH. PSEND = Percentage of transactions that are send transactions. TRXPENV = Average number of transactions per interchange. This value is also used in space calculations for EDITSEV. ENVPDAY = Number of interchanges sent per day = (TRXPDAY*PSEND) / TRXPENV AVGUSGS = Average number of usages that occur in each interchange. MRPSNUM = ENVPDAY * AVGUSGS * PENDLIFE
EDIMRPSX	ALL/DB2 index	Management Reporting Pending Send Usage Unique Index. A unique index created on the key value to EDIMRPS. There is one index entry for each EDIMRPS entry. MRPSXNUM = MRPSNUM
EDIOWNR	ALL/DB2 table	Management Reporting Table Owner User ID.

Allocation tables

This section describes the DB2 and VSAM allocation parameters shipped with WebSphere Data Interchange, and the two allocation values for the parameters. The secondary allocation parameter supports allocating additional storage up to 123 extents. Table 245 and Table 246 on page 786 also provide information used in the "Space calculation scenario" on page 786.

Notes:

1. FLOOR specifies that a fractional part of a number has been discarded and the next smallest integer is being used.

2. CEILING specifies that a fractional part of a number is rounded up to the next whole integer.

3. Table 250 calculations are made assuming PCTFREE=0 and FREEPAGE=0.

4. DB2 tables always have at least one data page. If the result of (nkb/4) -2 is less than 1, a value of 1 is used in the calculation.

Supplied DB2 database allocation

Table 245 describes all database records used by WebSphere Data Interchange and the number of records that the allocation values will accommodate.

Formulas for DB2 database allocation

This formula was used to determine the number of records per page (rpp):

```
rsz = Record size
ups = Usable page size = 4074
rpp = Records per page = FLOOR(ups/rsz)
```

This formula was used to determine the number of records per page for the index tables:

```
ksz = Index size (record size)
ups = Usable page size = 4067
spp = Sub-pages per page = 8
rpp = Records per page = FLOOR((ups - (spp * (ksz+21)))/(ksz+4))
```

This formula was used to determine the number of records in the Primary Allocation (rpa) for non-index tables:

```
nkb = Number of kilobytes in Primary Allocation
rpa = Number of records in Primary Allocation = (((nkb/4) - 2) * rpp)
```

Table 245. DB2 database allocation

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
EDICSTX	4800	2400	1	4032	1198
EDICSTXX	32	16	143	22	858
EDIELOG	1000	500	9	440	2232
EDIELOGX	1000	500	44	72	10664

Allocation tables

Table 245. DB2 database allocation (continued)

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
EDIELOG1	1000	500	39	80	9424
EDIELOG2	1000	500	35	88	8432
EDIENVP	10	5	29	140	15
EDIENVPX	10	5	320	8	160
EDIMRCM	100	25	101	40	2323
EDIMRCMX	100	25	111	29	2553
EDIMRPC	100	25	92	44	2116
EDIMRPCX	100	25	101	32	2323
EDIMRRT	100	25	42	96	966
EDIMRRTX	100	25	34	88	782
EDIMRPR	100	25	40	100	920
EDIMRPRX	100	25	101	32	2323
EDIMRST	100	25	46	88	1058
EDIMRSTX	100	25	39	79	897
EDIMRPS	100	25	44	92	1012
EDIMRPSX	100	25	101	32	2323
EDIMSGS	1000	500	12	336	2976
EDIMSGSX	1000	500	385	6	95480
EDIOWNR	10	2	339	12	170
EDIPROF	500	100	1	4096	123
EDIPROFX	100	20	61	53	1403
EDIPSAC	10	5	50	80	25
EDIPSACX	10	5	319	8	160
EDIPSAD	50	10	50	81	525
EDIPSADX	10	5	319	8	160
EDIPSAP	10	5	49	82	25
EDIPSAPX	10	5	319	8	160
EDIPSCR	10	5	21	193	11
EDIPSCRX	10	5	188	16	94
EDIPSDI	10	5	6	591	3
EDIPSDIX	10	5	778	1	389
EDIPSEI	10	2	70	58	35
EDIPSEIX	10	2	320	8	160
EDIPSEE	100	50	7	531	161
EDIPSEEX	100	50	320	8	7337

Table 245. DB2 database allocation (continued)

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
EDIPSIE	100	50	16	253	368
EDIPSLP	10	5	46	87	23
EDIPSLPX	10	5	385	6	193
EDIPSMQ	100	50	24	166	552
EDIPSMQX	100	50	320	8	7337
EDIPSNO	150	50	25	158	888
EDIPSNOX	50	10	132	24	1386
EDIPSNP	10	5	15	264	8
EDIPSNPX	10	5	319	8	160
EDIPSPD	10	5	1	4046	1
EDIPSPDX	10	5	319	8	160
EDIPSRQ	50	10	17	234	179
EDIPSRQX	10	5	188	16	94
EDIPSSP	10	5	30	133	15
EDIPSSPX	10	5	319	8	160
EDIPSSY	10	5	60	67	30
EDIPSSYX	10	5	319	8	160
EDIPSTD	500	100	44	92	5412
EDIPSTDX	100	50	319	8	7337
EDIPSTE	100	50	14	284	322
EDIPSTEX	100	50	320	8	7337
EDIPSTP	500	100	4	1012	492
EDIPSTPX	100	50	188	16	4324
EDIPSTP1	100	50	49	64	1127
EDIPSTP2	100	50	83	39	1909
EDIPSTP3	100	50	127	25	2921
EDIPSTT	200	100	38	106	1824
EDIPSTTX	100	50	75	43	1725
EDIPSTT1	100	50	44	71	1012
EDIPSTV	200	100	38	106	1824
EDIPSTVX	100	50	75	43	1725
EDIPSUE	100	50	15	268	345
EDIPSUEX	100	50	320	8	7337
EDIPSXE	100	50	14	275	322
EDIPSXEX	100	50	320	8	7337

Allocation tables

Table 245. DB2 database allocation (continued)

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
EDISTDDEH	5000	2500	1	4222	1248
EDISTDDEHX	300	120	86	38	6205
EDISTDDEHY	50	20	108	30	1124
EDISTDDED	25	12	79	51	336
EDISTDDEDU	25	10	81	40	344
EDISTDDEDX	25	10	81	40	344
EDISTDDEDY	25	10	71	46	374
EDISTDDEDZ	100	40	86	38	1955
EDISTDCDN	50	25	3	1079	32
EDISTDCDNX	50	20	76	43	788
EDISTDSGH	1000	500	1	4000	248
EDISTDSGHX	100	40	86	38	1955
EDISTDSGD	5000	2500	79	51	98592
EDISTDSGDU	2000	800	81	40	40338
EDISTDSGDX	2000	800	81	40	40338
EDISTDSGDY	500	200	320	8	39237
EDISTDSGDZ	1000	400	86	38	21080
EDISTDSGN	100	50	3	1089	69
EDISTDSGNX	50	20	76	43	788
EDISTDSTH	100	50	1	2771	23
EDISTDSTHX	20	5	108	30	321
EDISTDENV	300	150	43	94	3139
EDISTDENVX	50	20	108	30	1124
EDISTDTXD	4000	2000	57	71	56886
EDISTDTXDX	2000	800	77	42	38346
EDISTDTXDY	1000	400	86	38	21080
EDISTDTXDZ	1000	400	320	8	79112
EDISTDTXH	1000	500	1	4012	248
EDISTDTXHX	100	40	86	38	1955
EDISTDTXHY	20	8	108	30	321
EDISTDTXN	50	25	3	1081	32
EDISTDTXNX	100	40	72	45	1656
EDISSTK	40	10	25	159	200
EDISSTKX	20	5	155	20	465
EDISSTXX	20	5	48	65	144

Table 245. DB2 database allocation (continued)

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
EDITPCM	150	50	2	2000	71
EDITPCMX	50	20	75	43	788
EDITPCN	150	100	45	90	1598
EDITPCNX	50	10	57	56	599
EDITPCT	300	100	2	2000	146
EDITPCTX	30	10	81	40	446
EDIRULE	400	100	6	600	588
EDIRULEX	200	50	35	88	1632
EDIRULE1	200	150	19	139	912
EDIRULE2	200	50	189	16	9024
EDIRULE3	200	50	189	16	9024
EDITPRT	40	10	18	217	144
EDITPRTX	20	5	41	76	123
EDITPRTY	20	5	76	43	282
EDITPRTZ	20	5	189	16	564
EDITPST	40	10	12	324	96
EDITPSTX	20	5	37	83	111
EDITPSTY	20	5	68	48	201
EDITPSTZ	20	5	47	67	141
EDIMAPHEAD	50	25	21	192	221
EDIMAPHEADX	100	40	188	16	4324
EDIMAPAPPLCNTL	60	30	12	337	156
EDIMAPAPPLCNTLX	50	20	171	18	1785
EDIMAPAPPLCNTLY	50	20	189	16	1974
EDIMAPSEG	2000	1000	1	2884	498
EDIMAPSEGX	400	160	143	22	14014
EDIMAPSEGY	100	40	189	16	4324
EDIMAPSEGZ	800	320	123	26	24354
EDIMAPELE	5000	2500	1	2821	1248
EDIMAPELEX	2000	800	101	32	50298
EDIMAPELEY	1000	400	132	24	32736
EDIMAPGBLVAR	50	25	22	184	231
EDIMAPGBLVARX	25	10	108	30	455
EDIMAPSYNTAX	5000	2500	49	82	61152
EDIMAPSYNTAXX	2000	800	171	18	84660

Allocation tables

Table 245. DB2 database allocation (continued)

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
EDIMAPLCLVAR	5000	2500	25	162	31200
EDIMAPLCLVARX	2000	800	71	46	34860
EDIMAPREF	5000	2500	12	337	14976
EDIMAPREFX	2000	800	171	18	84660
EDIMAPNODES	5000	2500	2	2085	2496
EDIMAPNODESX	2000	800	171	18	84660
EDIMAPCMDS	5000	2500	2	2072	2496
EDIMAPCMDSX	2000	800	171	18	53286
EDIADFDICT	50	25	1	2166	11
EDIADFDICTX	25	10	108	30	455
EDIRECIDINFO	50	25	1	2178	11
EDIRECIDINFOX	50	20	108	30	1124
EDIADFHEADER	50	25	1	2565	210
EDIADFHEADERX	50	20	71	46	1974
EDIADFHEADERY	50	20	108	30	1124
EDIADFHEADERZ	50	20	108	30	1124
EDIADFHDRMEM	100	50	37	110	851
EDIADFHDRMEMX	50	20	68	48	704
EDIADFHDRMEMY	50	20	71	46	735
EDIADFLOOP	50	25	1	2226	11
EDIADFLOOPX	50	20	53	60	557
EDIADFLOOPY	50	20	108	30	1124
EDIADFLOOPZ	50	20	108	30	1124
EDIADFLOOPMEM	100	50	37	108	851
EDIADFLOOPMEMX	100	40	50	64	1127
EDIADFLOOPMEMY	50	20	53	60	557
EDIADFRECORD	200	100	1	2242	48
EDIADFRECORDX	100	40	53	60	1219
EDIADFRECORDY	50	20	108	30	1124
EDIADFRECORDZ	50	20	108	30	1124
EDIADFRECMEM	1000	500	28	141	6944
EDIADFRECMEMX	300	120	50	64	3577
EDIADFRECMEMY	100	40	53	60	1219
EDIADFSTRUCT	400	200	1	2196	98
EDIADFSTRUCTX	100	40	53	60	1219

Table 245. DB2 database allocation (continued)

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
EDIADFSTRUCTY	50	20	108	30	1124
EDIADFSTRUCTMEM	2000	200	27	149	13446
EDIADFSTRUCTMEMX	600	240	50	64	7252
EDIADFSTRUCTMEMY	200	80	53	60	2544
EDIADFFIELD	5000	2500	1	2551	1248
EDIADFFIELDX	2000	800	53	60	26394
EDIADFFIELDY	1000	400	108	30	26536
EDIXMLDICT	50	25	34	118	357
EDIXMLDICTX	25	10	108	30	455
EDIDTDHDR	5000	2500	2	1398	2496
EDIDTDHDRX	2000	800	53	60	26394
EDIDTD	50	25	12	320	126
EDIDTDX	50	20	50	64	515
EDITSTH	600	120	12	323	1776
EDITSTHX	150	30	272	10	9656
EDITSTHO	24	8	123	26	492
EDITSTI	600	120	2	2000	296
EDITSTIX	150	30	210	14	7455
EDITSTO	600	120	17	237	2516
EDITSTOX	150	30	272	10	9656
EDITSAU	600	120	58	70	8584
EDITSAUX	150	30	210	14	7455
EDITSAUY	150	30	171	18	6071
EDITSEV	600	120	11	501	1184
EDITSEVX	150	30	48	66	1704
EDITSEV1	10	3	319	8	160
EDITSGP	600	120	11	341	1628
EDITSGPX	150	30	38	80	1349
EDITSTU	600	120	16	241	2368
EDITSTUX	150	30	32	94	1136
EDITSTUY	150	30	272	10	9656
EDITSLT	10	5	4074	1	2037

Supplied database allocation for VSAM

Table 246 describes the VSAM records used by WebSphere Data Interchange including the number of records that the allocation values will accommodate.

Formula for VSAM database allocation

This formula was used to determine the number of records per control interval (rci):

```
rsz = Record size
ciz = Control interval size = 4096
rci = FLOOR((ciz-4/(rsz+3))
```

Note: Assume 10 control intervals per track.

Table 246. VSAM database allocation

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	# of Primary Allocation Records
HELPS	105	10	2000	2	2100
SCREENS	20	5	2000	2	400

Space calculation scenario

The items listed below provide the information required to perform the example space calculations. The items are followed by tables showing the DB2 and VSAM space allocations necessary for the example. These tables are followed by a blank worksheet which can be copied and used for your own estimates.

Note: When you are attempting to do a space estimate, do not spend too long trying to arrive at an exact number of fields in a structure or number of elements in a segment. Pick a value that seems reasonable and use it.

The majority of DASD required by WebSphere Data Interchange is used in the Transaction Store, and the amount of DASD required for the Transaction Store depends on the number of transactions processed per day and the average size of a transaction image. The following values have the most impact on your space calculations:

- 100 trading partners.
- 50 requestor IDs.
- 10 transactions sets used but the entire X12V3R1 standard has been applied.
- Average transaction has 30 segments defined with 10 fields per segment and all of these are mapped.
- Half of the mapped data elements required a literal, accumulator, or other special action (validation table, date edit, translate table, and so forth).
- 25000 transactions per month.
- 2000-character transactions.
- 10 data formats.
- Transaction level recovery is specified during translation.

- One mapping for each transaction set and that mapping is used by the 100 trading partners. 50 trading partners will be used for Receive processing and 50 trading partners will be used for Send processing.
- 30 segments mapped per transaction with 10 fields per segment.
- 30 structures per data format with 10 fields per structure.
- Average number of transactions per envelope is eight.
- Each interchange contains one functional group.
- Transactions are retained for one month with a TRANSACTION REMOVE command run once a month.
- Management reporting statistics are retained for two months with a PERFORM UPDATE STATISTICS command run once a week.

DB2 database allocation required for space calculation scenario

Table 247 on page 788 describes the allocation required given the set of assumptions explained in "Space calculation scenario" on page 786.

Formula for DB2 primary allocation (PRIQTY)

This formula was used to determine the DB2 primary allocation amount (PRIQTY):

```
rpp = Records per page
```

(shown in Table 245 on page 779.)

```
psz = DB2 page size = 4096
alu = DB2 allocation unit = 1024 bytes
rnb = Total number of records wanted
pri = Primary allocation = ((FLOOR (rnb/rpp) + 2) *psz)/alu
```

Space calculation scenario

Table 247. DB2 allocation required for the space calculation scenario

Table name	Number of records required	DB2 primary allocation	Comments
EDICSTX	50	208	CSTXNUM = TDIDNUM + (4 * TPTXNUM) CSTXNUM = 10 + (4 * 10) CSTXNUM = 50
EDICSTXX	50	12	CSTXXNUM = CSTXNUM CSTXXNUM = 50
EDISTDDEH	901	96	SCDENUM = EDISCDE entries in X12V3R1
EDISTDDEHX	901	28	SCDEXNUM = SCDENUM SCDEXNUM = 901
EDISTDSGD	3707	184	SCSGNUM = EDISCSG entries in X12V3R1
EDISTDSGDX	3707	136	SCDUXNUM = SCDUNUM SCDUXNUM = 3707
EDISTDSEG	520	48	SCSGNUM = EDISCSG entries in X12V3R1
EDISTDSEGX	520	20	SCSGXNUM = SCSGNUM SCSGXNUM = 520
EDISTDSTH	1	12	SCSTNUM = 1 standard loaded
EDISTDSTHX	1	8	SCSTXNUM = SCSTNUM SCSTXNUM = 1
EDISTDTXD	1622	88	SCSUNUM = EDISCSU entries in X12V3R1
EDISTDTXDX	1622	64	SCSUXNUM = SCSUNUM SCSUXNUM = 1622
EDISTDTRX	39	12	SCTXNUM = EDISCTX entries in X12V3R1
EDISTDTRXX	39	12	SCTXXNUM = SCTXNUM SCTXXNUM = 39
EDIMAPELE	3000	352	SSMAP = Average number of MAPPED segments = 30 ELSS = Average number elements per segment = 10 TPDDNUM = TPTXNUM * SSMAP * ELSS TPDDNUM = 10 * 30 * 10 = 3000
EDIMAPELEX	3000	100	TPDDXNUM = TPDDNUM TPDDXNUM = 3000

Table 247. DB2 allocation required for the space calculation scenario (continued)

Table name	Number of records required	DB2 primary allocation	Comments
EDITPRT	500	96	TPTXNUM = Total number of mappings = 10 PRECV = Percentage of Receive mappings = 50% TPPROFNUM = Total number of trading partners = 100 TPRECV = Percentage trading partners receiving = 100% TPRTNUM = (TPTXNUM * PRECV) * (TPPROFNUM * TPRECV) TPRTNUM = (10 * .5) * (100 * 1) TPRTNUM = 500
EDITPRTX	500	60	TPDDXNUM = TPDDNUM TPDDXNUM = 3000
EDIRULE	1500	324	TPDDNUM = Total number of fields = 3000 SPMAP = Percentage of fields that are mapped with either literal values, accumulators, edits, validation/translation tables, sub-string/concatenate, user exits = 50% TPRUNUM = TPDDNUM * SPMAP TPRUNUM = 3000 * .5 = 1500
EDIRULEX	1500	56	TPRUXNUM = TPRUNUM TPRUXNUM = 1500
EDIMAPSEG	300	44	SSSTD = Average number of segments in a transaction = 30 SSRMAP = Average number repeated mappings = 0 TPSGNUM = TPTXNUM * (SSSTD + SSRMAP) TPSGNUM = 10 * (30 + 0) = 300
EDIMAPSEGX	300	16	TPSGXNUM = TPSGNUM TPSGXNUM = 300
EDITPST	500	152	TPTXNUM = Total number of mappings = 10 PSEND = Percentage of mappings for Send = 50% TPPROFNUM = Total number of trading partners = 100 TPSEND = Percentage of trading partners you send to = 100% TPSTNUM = (TPTXNUM * PSEND) * (TPPROFNUM * TPSEND) TPSTNUM = (10 * .5) * (100 * 1) TPSTNUM = 500

Space calculation scenario

Table 247. DB2 allocation required for the space calculation scenario (continued)

Table name	Number of records required	DB2 primary allocation	Comments
EDITPSTX	500	52	TPSTXNUM = TPSTNUM TPSTXNUM = 500
EDIMAPHEAD	10	12	TPTXNUM = Number of mappings = 10
EDIMAPHEADX	10	12	TPTXXNUM = TPTXNUM TPTXNUM = Number of mappings = 10 TPTXXNUM = 10
EDIADFHEADER	10	12	TDIDNUM = Number of data formats = 10
EDIADFHEADERX	10	12	TDIDXNUM = TPIDNUM TDIDXNUM = 10
EDIADFSTRUCT	3600	240	TDIDNUM = Number of data formats = 10 STFMT = Average number of structures per data format = 30 FLSTR = Average number of fields per structure = 10 TDSTNUM = ((STFMT * 2) + (STFMT * FLSTR)) * TDIDNUM TDSTNUM = ((30 * 2) + (30 * 10)) * 10 TDSTNUM = 3600
EDIADFSTRUCTX	3600	168	TDSTXNUM = TPSTNUM TDSTXNUM = 3600
EDITSTH	50000	16676	TRXPDAY = Number of transactions per day = 25000/30 TSTHNUM = TRXPDAY * TSLIFE TSTHNUM = 833.3 * 60 = 50000 **Note:** TRXLIFE = Days before purge = 30 PRGSPAN = Frequency of remove utility = 30 TSLIFE = TRXLIFE + PGRSPAN TSLIFE = 30 + 30 = 60
EDITSTHX	50000	744	TSTHXNUM = TSTHNUM TSTHXNUM = 50000
EDITSTI	50000	100008	TSTINUM = TSTHNUM = 50000
EDITSTIX	50000	964	TSTIXNUM = TSTINUM TSTIXNUM = 50000
EDITSTO	0	8	TSTONUM = 0 (Overrides not being used)
EDITSTOX	0	8	TSTOXNUM = TSTONUM TSTOXNUM = 0
EDITSAU	50000	3856	TSAUNUM = TSTHNUM = 50000

Table 247. DB2 allocation required for the space calculation scenario (continued)

Table name	Number of records required	DB2 primary allocation	Comments
EDITSAUX	50000	964	TSAUXNUM = TSAUNUM TSAUXNUM = 50000
EDITSAUY	50000	1188	TSAUYNUM = TSAUNUM TSAUYNUM = 50000
EDITSEV	6250	3136	TRXPENV = Average number of transactions per envelope = 8 TSEVNUM = TSTHNUM / TRXPENV TSEVNUM = 50000 / 8 = 6250
EDITSEVX	6250	532	TSEVXNUM = TSEVNUM TSEVXNUM = 6250
EDITSGP	6250	2284	GRPPENV = Average groups per envelope = 1 TSGPNUM = TSEVNUM * GRPPENV TSGPNUM = 6250 * 1 = 6250
EDITSGPX	6250	668	TSGPXNUM = TSGPNUM TSGPXNUM = 6250
EDITSTU	50000	12508	TSTHNUM = TSTHNUM = 50000
EDITSTUX	50000	6260	TSTUXNUM = TSTUNUM TSTUXNUM = 50000
EDITSTUY	50000	744	TSTUYNUM = TSTUNUM TSTUYNUM = 50000

Space calculation scenario

Table 247. DB2 allocation required for the space calculation scenario (continued)

Table name	Number of records required	DB2 primary allocation	Comments
EDIMRCM	3100	178	RIDNUM = Number of Requestor IDs (RID) = 50 CUMENTS = Number of cumulative entries = RIDNUM * 2 CUMENTS = 50 * 2 = 100 PRIDINT = Daily percentage of RIDs having an interchange = 50% MEASPDAY = (RIDNUM * 2) * PRIDINT MEASPDAY = (50 * 2) * .5 = 50 MRCMNUM = (MEASPDAY * MEASLIFE) + CUMENTS MRCMNUM = (50 * 60) + 100 = 3100 **Note:** MEASLIFE = 60 PENDLIFE = 7 RIDNUM = 50 TPRTNUM = 500 TPSTNUM = 500
EDIMRCMX	3100	122	MRCMXNUM = MRCMNUM MRCMXNUM = 3100
EDIMRPC	7000	435	Transaction level recovery formula: RIDNUM = Number of Requestor IDs (RID) = 50 SENDNUM = Average number of send interchanges per RID per day = 10 RECVNUM = Average number of receive interchanges per RID per day = 10 PENDPDAY = (SENDNUM + RECVNUM) * RIDNUM PENDPDAY = (10 + 10) * 50 = 1000 MRPCNUM = PENDPDAY * PENDLIFE MRPCNUM = 1000 * 7 = 7000
EDIMRPCX	7000	302	MRPCXNUM = MRPCNUM MRPCXNUM = 7000

Table 247. DB2 allocation required for the space calculation scenario (continued)

Table name	Number of records required	DB2 primary allocation	Comments
EDIMRRT	15500	1926	PACTIVE = Daily percentage of receive usages having activity = 50%
			MEASPDAY = TPRTNUM * PACTIVE
			MEASPDAY = 500 * .5 = 250
			MRRTNUM = (MEASPDAY * MEASLIFE) + TPRTNUM
			MRRTNUM = (250 * 60) + 500 = 15500
EDIMRRTX	15500	1983	MRRTXNUM = MRRTNUM
			MRRTXNUM = 15500
EDIMRPR	2919	385	Transaction level recovery formula:
			TRXPDAY = Number of transactions per day = 25000/30 = 834
			PRECV = Percentage of transactions that are receive transactions = 50%
			MRPRNUM = TRXPDAY * PRECV * PENDLIFE
			MRPRNUM = 834 * .5 * 7 = 2919
EDIMRPRX	2919	125	MRPRXNUM = MRPRNUM
			MRPRXNUM = 2919
EDIMRST	15500	1774	PACTIVE = Daily percentage of send usages having activity = 50%
			MEASPDAY = TPSTNUM * PACTIVE
			MEASPDAY = 500 * .5 = 250
			MRSTNUM = (MEASPDAY * MEASLIFE) + TPSTNUM
			MRSTNUM = (250 * 60) + 500 = 15500
EDIMRSTX	15500	1728	MRSTXNUM = MRSTNUM
			MRSTXNUM = 15500

Space calculation scenario

Table 247. DB2 allocation required for the space calculation scenario (continued)

Table name	Number of records required	DB2 primary allocation	Comments
EDIMRPS	2919	353	Transaction level recovery formula: TRXPDAY = Number of transactions per day = 25000/30 = 834 PRECV = Percentage of transactions that are receive transactions = 50% MRPRNUM = TRXPDAY * PRECV * PENDLIFE MRPRNUM = 834 * .5 * 7 = 2919
EDIMRPSX	2919	126	MRPSXNUM = MRPSNUM MRPSXNUM = 2919

VSAM database allocation required for space calculation scenario

Table 248 describes the allocation required given the set of assumptions as explained in "Space calculation scenario" on page 786.

Formula for VSAM primary allocation (TRKS)

The VSAM primary allocation amount (TRKS) is determined with:

```
rci = Records per control interval
```

(shown in Table 246 on page 786.)

```
ppt = Number of control intervals per TRACK = 10
      (3380=10, 3330=3, 3340=2, 3350=4, 3375=8)
rnb = Total number of records wanted
pri = Primary allocation = CEILING (((FLOOR (rnb/rci)) / ppt))
```

Note: A minimum of two tracks are allocated. EDISCRN and EDIHELP are not fixed in size.

Table 248. VSAM allocation required for the space calculation scenario

Table name	VSAM Primary Allocation (tracks)
EDISCRN	20
EDIHELP	25

Space calculation worksheets

Use the worksheet in Table 249 on page 795 to help you to calculating your space estimates for DB2 database records supplied.

Table 249. DB2 database allocation worksheet

Table name	Number of records required	DB2 primary allocation
EDICSTX		
EDICSTXX		
EDIELOG		
EDIELOGX		
EDIELOG1		
EDIELOG2		
EDIENVP		
EDIENVPX		
EDIMRCM		
EDIMRCMX		
EDIMRPC		
EDIMRPCX		
EDIMRRT		
EDIMRRTX		
EDIMRPR		
EDIMRPRX		
EDIMRST		
EDIMRSTX		
EDIMRPS		
EDIMRPSX		
EDIMSGS		
EDIMSGSX		
EDIOWNR		
EDIPROF		
EDIPROFX		
EDIPSAC		
EDIPSACX		
EDIPSAD		
EDIPSADX		
EDIPSAP		
EDIPSAPX		
EDIPSCR		
EDIPSCRX		
EDIPSDI		
EDIPSDIX		
EDIPSEI		

Space calculation worksheets

Table 249. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDIPSEIX		
EDIPSEE		
EDIPSEEX		
EDIPSIE		
EDIPSIEX		
EDIPSLP		
EDIPSLPX		
EDIPSMQ		
EDIPSMQX		
EDIPSNO		
EDIPSNOX		
EDIPSNP		
EDIPSNPX		
EDIPSPD		
EDIPSPDX		
EDIPSRQ		
EDIPSRQX		
EDIPSSP		
EDIPSSPX		
EDIPSSY		
EDIPSSYX		
EDIPSTD		
EDIPSTDX		
EDIPSTE		
EDIPSTEX		
EDIPSTP		
EDIPSTPX		
EDIPSTP1		
EDIPSTP2		
EDISPTP3		
EDIPSTT		
EDIPSTTX		
EDIPSTT1		
EDIPSTV		
EDIPSTVX		
EDIPSUE		

Table 249. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDIPSUEX		
EDIPSXE		
EDIPSXEX		
EDISTDDEH		
EDISTDDEHX		
EDISTDDEHY		
EDISTDDED		
EDISTDDEDU		
EDISTDDEDX		
EDISTDDEDY		
EDISTDDEDZ		
EDISTDCDN		
EDISTDCDNX		
EDISTDSGH		
EDISTDSGHX		
EDISTDDSGD		
EDISTDSGDU		
EDISTDSGDX		
EDISTDSGDY		
EDISTDSGDZ		
EDISTDSGN		
EDISTDSGNX		
EDISTDSTH		
EDISTDSTHX		
EDISTDENV		
EDISTDENVX		
EDISTDTXD		
EDISTDTXDX		
EDISTDTXDY		
EDISTDTXDZ		
EDISTDTXH		
EDISTDTXHX		
EDISTDTXHY		
EDISTDTXN		
EDISTDTXNX		
EDISSTK		

Space calculation worksheets

Table 249. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDISSTKX		
EDISSTXX		
EDITPCM		
EDITPCMX		
EDITPCN		
EDITPCNX		
EDITPCT		
EDITPCTX		
EDIRULE		
EDIRULEX		
EDIRULE1		
EDIRULE2		
EDIRULE3		
EDITPRT		
EDITPRTX		
EDITPRTY		
EDITPRTZ		
EDITPST		
EDITPSTX		
EDITPSTY		
EDITPSTZ		
EDIMAPHEAD		
EDIMAPHEADX		
EDIMAPAPPLCNTL		
EDIMAPAPPLCNTLX		
EDIMAPAPPLCNTLY		
EDIMAPSEG		
EDIMAPSEGX		
EDIMAPSEGY		
EDIMAPSEGZ		
EDIMAPELE		
EDIMAPELEX		
EDIMAPELEY		
EDIMAPGBLVAR		
EDIMAPGBLVARX		
EDIMAPSYNTAX		

Table 249. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDIMAPSYNTAXX		
EDIMAPLCLVAR		
EDIMAPLCLVARX		
EDIMAPREF		
EDIMAPREFX		
EDIMAPNODES		
EDIMAPNODESX		
EDIMAPCMDS		
EDIMAPCMDSX		
EDIADFDICT		
EDIADFDICTX		
EDIRECIDINFO		
EDIRECIDINFOX		
EDIADFHEADER		
EDIADFHEADERX		
EDIADFHEADERY		
EDIADFHEADERZ		
EDIADFHDRMEM		
EDIADFHDRMEMX		
EDIADFHDRMEMY		
EDIADFLOOP		
EDIADFLOOPX		
EDIADFLOOPY		
EDIADFLOOPZ		
EDIADFLOOPMEM		
EDIADFLOOPMEMX		
EDIADFLOOPMEMY		
EDIADFRECORD		
EDIADFRECORDX		
EDIADFRECORDY		
EDIADFRECORDZ		
EDIADFRECMEM		
EDIADFRECMEMX		
EDIADFRECMEMY		
EDIADFSTRUCT		
EDIADFSTRUCTX		

Space calculation worksheets

Table 249. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDIADFSTRUCTY		
EDIADFSTRUCTMEM		
EDIADFSTRUCTMEMX		
EDIADFSTRUCTMEMY		
EDIADFFIELD		
EDIADFFIELDX		
EDIADFFIELDY		
EDIXMLDICT		
EDIXMLDICTX		
EDIDTDHDR		
EDIDTDHDRX		
EDIDTD		
EDIDTDX		
EDITSTH		
EDITSTHX		
EDITSTHO		
EDITSTI		
EDITSTIX		
EDITSTO		
EDITSTOX		
EDITSAU		
EDITSAUX		
EDITSAUY		
EDITSEV		
EDITSEVX		
EDITSEV1		
EDITSGP		
EDITSGPX		
EDITSTU		
EDITSTUX		
EDITSTUY		
EDITSLT		

Appendix D. Performance considerations

This appendix provides performance information on WebSphere Data Interchange and WebSphere Data Interchange to help you plan your system requirements.

The following considerations and tuning techniques are highly recommended to achieve optimum WebSphere Data Interchange performance:
- "General optimization techniques"
- "WebSphere Data Interchange/z/OS considerations" on page 802
- "WebSphere Data Interchange CICS considerations" on page 803
- "File maintenance techniques" on page 804
- "Transaction Store query techniques" on page 805

General optimization techniques

- If you are a high-volume EDI customer using a DB2 repository:
  - Define multiple DASD volumes to DB2 storage group EDISTG01 to facilitate spreading DB2 data sets across multiple packs.
  - Define large DASD allocations for Transaction Store DB2 tables EDITSAU, EDITSTH, EDITSTI, EDITSTU, EDITSEV, EDITSGP which are heavily referenced.
  - Monitor the status of DB2 Table data sets on a regular basis (at least twice monthly, more often if possible). Use LISTCAT and DB2 Utilities such as CHECK, RUNSTAT, and REORG to monitor and tune the DB2 table data sets.
  - Add volumes to DB2 storage groups, increase table space DASD, and reorganize DB2 Table data sets when CI, and especially CA, splits are encountered.
- Define the Master Catalog for the high-level data set name qualifier on a separate pack from all other z/OS data sets.
- Log only application data images and EDI standard data images during the testing period. The trading partner profile (TPPROF) controls logging of interchange images. The usage/rule controls the logging of application images. Additionally, do not log profile access attempts if not necessary for your environment. The `Log` action on the Profile menu controls this.
- Log only events during the testing period. The Activity Logs (ACTLOGS) profile, member EDIFFS, `Log` flag controls whether normal events are logged. Error conditions and events that change a transaction's status are recorded even if logging is turned off.
- Minimize translator error messages by setting the &DIERRFILTER keyword.
- Use commands that combine several functions in one; for example, rather than using TRANSLATE TO STANDARD and then ENVELOPE, use TRANSLATE AND ENVELOPE to have both functions performed at the same time.
- If concurrency is an issue and multiple WebSphere Data Interchange invocations use the same trading partner, use the `RECOVERY` keyword to single-thread the multiple DI invocations through the trading partner profile. This is true for outbound translation and when generating functional acknowledgments. Transaction level recovery, rather than envelope level recovery, might be advisable here to help concurrency and improve throughput. This can be accomplished using the RECOVERY keyword.

General optimization techniques

- Specify that WebSphere Data Interchange pass application data structures in groups whenever possible. Passing data structures in groups requires less interaction, less overhead, and less I/O than passing data structures separately.
- Avoid excessive use of the keyword literals **&E**, **&IF**, **&ERR**, and **&ASSERT** in highly repetitive loop maps. Although conditional logic map is a valuable feature but, if used excessively, it degrades translation performance and increases processing costs.
- Compile maps to generate control strings for your EDI envelope standards to reduce I/O associated with retrieving envelope definitions.
- If you are currently using a validation level of 2, consider using the keyword literals **&DIVALTYPE** and **&DIVALLEVEL** as an alternative. You can limit validation to specific, important elements using these keywords.

WebSphere Data Interchange/z/OS considerations

- Define FFSWORK data set in the WebSphere Data Interchange/z/OS utility JCL as virtual I/O to drastically reduce I/O exceptions relating to the work data set.

```
//FFSWORK DD DSN=&&FFSWORK,DISP=(NEW,DELETE,DELETE),
//         DCB=(RECFM=V,BLKSIZE=32760),
//         UNIT=VIO,SPACE=(TRK,(3,3))
```

- Use a record length greater than 512 with a large block size for envelope files to reduce I/O during envelope and de-envelope operations.
- Define the transaction input data sets (APDATA01, APDATA02, and so forth) on separate DASD from other WebSphere Data Interchange/z/OS data sets to greatly reduce data set contention.
- If translating large files, the following formula can be used to determine virtual storage requirements. (The same formula pertains to both inbound and outbound translation.)

```
Virtual storage required = Largest application transaction image
            + EDI interchange length + 4 MB overhead
```

Largest application transaction image: The length of the largest transaction in application format (input to TRANSLATE AND ENVELOPE, or output from DEENVELOPE AND TRANSLATE).

If multiple EDI transactions are being processed, only consider the largest transaction in application format because WebSphere Data Interchange releases the application image as each transaction is processed. In most cases, the application image is many times larger than the EDI counterpart because application fields are fixed length. Also, data format structures passed as a group tend to drastically increase this length due to exorbitant padding.

EDI interchange image: The length of the entire EDI interchange, including header and trailer segments.
EDIFACT
        UNB through UNZ
ANSI X12
        ISA through IEA
UCS   BG through EG

UN/TDI STX through SCH
ICS   ICS through ICE

4 MB overhead: The amount of storage required for loading programs, I/O buffering, etc. WebSphere Data Interchange obtains most storage above the 16 MB line.

Note: The z/OS default limit for above-the-line storage is 32 MB. Specifying a JCL REGION of between 16 MB and 32 MB reduces this default limit. Only change the JCL REGION specification when your estimation exceeds 32 MB, or use REGION=8 MB to remove the limit. For detailed information about the REGION parameter, refer to z/OS JCL documentation.

WebSphere Data Interchange CICS considerations

- For CICS/ESA installations, the WebSphere Data Interchange Persistent Environment can be used. The Persistent Environment is designed to minimize read operations involved in translating data.
- Make sure transaction EDIX is enabled. This is done with CEMT I TRAN(EDIX).
- The WebSphere Data Interchange Utility single-threads through print files, exception files, report files, tracking files, and query files. The names of these files can be specified in the Utility Control Information block passed by the application to the Utility. If concurrency is an issue, choose unique names for these files.
- If concurrency is an issue, spread functional acknowledgments to separate TS queues by setting the FUNACKFILE keyword.
- For CICS/ESA installations, have EDIMSG defined as a CICS data table, if possible. Use the CICS RDO facility to do this. However, be aware that changes made in batch or TSO are not reflected immediately in CICS data tables.
- If you run many small, concurrent WebSphere Data Interchange translations involving numerous trading partners, you can develop HOT-DI applications to drive WebSphere Data Interchange.
- If translating large files, the following formula can be used to determine virtual storage requirements. (The same formula pertains to both inbound and outbound translation.)

```
Virtual storage required = Largest application transaction image
                + EDI interchange length + 4 MB overhead
```

Largest application transaction image: The length of the largest transaction in application format (input to TRANSLATE AND ENVELOPE, or output from DEENVELOPE AND TRANSLATE).

If multiple EDI transactions are being processed, you need only consider the largest transaction in the application format because WebSphere Data Interchange releases the application image as each transaction is processed. In most cases, the application image is many times larger than the EDI counterpart because application fields are fixed length. Also, data format structures passed as a group tend to drastically increase this length due to exorbitant padding.

WebSphere Data Interchange CICS considerations

EDI interchange image: The length of entire EDI interchange, including header and trailer segments.

EDIFACT
> UNB through UNZ

ANSI X12
> ISA through IEA

UCS   BG through EG

UN/TDI   STX through SCH

ICS   ICS through ICE

4 MB overhead: The amount of storage required for loading programs, I/O buffering, etc. WebSphere Data Interchange obtains most storage above the 16 MB line.

In CICS, most storage will be obtained from extended DSA. (This is a CICS 3.2 EDSALIN SIT parameter.) You must also consider other CICS activity, such as concurrent WebSphere Data Interchange tasks. Concurrent tasks each require 3 MB of additional overhead.

Maximize throughput by using concurrent WebSphere Data Interchange tasks. There should not be more than six of these at any one time. Use the CICS RDO TRANClass command to limit concurrent activity.

File maintenance techniques

- Delete or archive unused and outdated data from WebSphere Data Interchange/z/OS data sets on a regular basis. Small data sets result in better performance.
- Do not track unneeded, outdated transactions in the Transaction Store database. Performance is degraded when a database query range includes these unimportant records. Execute PURGE and REMOVE commands regularly to delete unneeded records. Also, use an appropriate purge interval. For more information on the file content and space calculations, see Appendix C, "Space calculation examples," on page 751
- Specify a WHERE clause for REMOVE commands to reduce the scope of the database query to transactions with a PURGE - DATE EXPIRED status. Before transactions can be removed, you must run a database query to determine the remove eligibility of expired transactions. Narrowing the query to only eligible transactions improves performance. For example, if the default purge interval of 30 days is used, transactions added in the last 30 days are not eligible for removal. The command could be: REMOVE TRANSACTIONS WHERE HANDLE(*-999) TO(*-30).
- If the Transaction Store information is not used or needed for your environment, turn off the writing of these records through options in the Application Defaults (APPDEFS) profile. However, be aware that if ENVELOPE or TRANSLATE TO APPLICATION is used, it is essential that Transaction Store records be written as this ensures their retrieval.
- Archive event log entries regularly. For more information on the archive utility, see "Using sample JCL" on page 407 Or, if records therein are not needed, use the sample install JCL to DELETE, DEFINE, and LOAD the log file. Sample JCL is contained in target library SEDIINS1. Member EDIJVSDF contains JCL to delete and define the log. Only run the applicable job step for the event log in question.

Transaction Store query techniques

The WHERE option specified during a database query plays a significant role in performance. A database query takes place with various PERFORM statements, the most common of which are ENVELOPE, TRANSLATE TO APPLICATION, PRINT, PURGE, and REMOVE. These Transaction Store utility functions require a search and retrieval of information contained in the Transaction Store.

The following techniques apply to all PERFORM commands that require a database selection/query. These same techniques can be utilized with the interactive Transaction Store facility. The following list contains criteria and the order in which the criteria are checked. (All fields at the same indentation level are checked in the order listed.)

```
        ----------------------------------------------------------------
        Transaction Handle
        Batch ID
        Direction
        All information in Transaction Handle Record(s)

            Table:  EDITSTH

            Application Control field
            Direction
            Standard Transaction ID
            Transaction Handle
            Date added to store
            Time added to store
            Trading Partner Nickname
            Internal Trading Partner ID
            Network ID
            Envelope Type
            Earliest purge date
            Earliest envelope date
        All information in Application Record(s)

            Table:  EDITSAU

            Application ID
            Data Format ID
            Batch ID
            Translation Error Level
            Delivered date
            Delivered time
        All information in Transaction Usage Record(s)

            Table:  EDITSTU

            Interchange Control Number
            Group Control Number
            Transaction Control Number
            Interchange Receiver ID
        All information in Group Usage Record(s)
```

Transaction Store query techniques

```
        Table:  EDITSGP

        Application Sender ID
        Application Receiver ID
        Functional Acknowledgment Pending
   All information in Envelope Usage Record(s)
        Table:  EDITSEV

        Interchange Sender ID
        Send date
        Send time
        Network Status
        Network Acknowledgment Pending
        Envelope date
        Envelope time
   -------------------------------------------------------------------
```

You do not need to understand all the relationships between the files and records to utilize the preceding information. However, be aware that generally there is one record for each EDI transaction in each of the previous tables. With this in mind, correctly applying the following two rules can significantly improve query performance:

1. Directly limit the query to a specific range of transactions using a handle range, batch ID, and/or direction (send or receive). Since the primary file is the transaction handle file (EDITSTH), and the primary key to this file is the transaction handle, specifying a handle range reduces the number of records accessed. When specifying a handle range, the selection program is given a partial key, thereby excluding inapplicable records.

   Note: When executing the WebSphere Data Interchange Utility you can use (*-*n*) in the From and To values of the HANDLE range where the asterisk (*) represents the current system date and *n* represents the number of days before it.

   Handle, Direction, and Batch ID are the only fields that directly narrow the search window. Further criteria specifications, such as Trading Partner Nickname or Standard Transaction ID, require a scan of all records obtained using Handle, Direction, or Batch ID.

2. You can limit the query using the transaction handle record (EDITSTH). Searching for specific criteria in the transaction handle record is the next best way to limit the selection. These values are not keys, but work better than using the transaction usage record, group usage record, or envelope usage record. For example, if you translate and send many transactions to a specific trading partner and want to retrieve these specific transactions, use the trading partner nickname instead of the interchange receiver ID. The trading partner nickname is more specific, and so is faster to retrieve than the interchange receiver ID and more efficient for performing queries.

Appendix E. Mapping the MQRFH2 header to the JMS API

This appendix explains how the MQRFH2 mapping to the Java Message Service (JMS) API works when you are communicating with a JMS Client.

The MQSeries Java Message Service (JMS) implementation uses the Message Content Descriptor (MCD) folder of the MQRFH2 to carry information about the message, as described in the MQRFH2 header. By default, the Message Domain (MSD) property is used to identify whether the message is a text, bytes, stream, map, or object message. This value is set depending on the type of the JMS messages.

If the application calls setJMSType, it can set the MCD type field to a value of its choosing. This type field can be read by the WebSphere MQ Integrator message flow, and a receiving JMS application can use the getJMSType method to retrieve its value. This applies to all kinds of JMS messages.

When a JMS application creates a text or bytes message, the application can set MCD folder fields explicitly by calling the setJMSType method and passing in a string argument in a special Universal Resource Identifier (URI) format as follows:

```
mcd://domain/[set]/[type][?format=fmt]
```

This URI form allows an application to set the MCD to a domain that is not one of the standard jms_xxxx values; for example, to domain mrm. It also allows the application to set any or all of the MCD set, type, and format fields if desired.

The string argument to setJMSType is interpreted as follows:
1. If the string does not appear to be in the special URI format (for example, it does not start with mcd://), the string is added to the MCD folder as the type field.
2. If the string does start with mcd:// and conforms to the URI format, and the message is a Text or Bytes message, the URI string is split into its constituent parts. The domain part overrides the jms_text or jms_bytes value that would otherwise have been generated, and the remaining parts (if present) are used to set the set, type, and format fields in the MCD. Note that set, type, and format are all optional.
3. If the string starts with mcd:// and the message is a Map, Stream or Object message, the setJMSType call throws an exception. You cannot override the domain, or provide a set or format for these classes of messages, but you can provide a type if you wish.

When an MQ message is received with an MSD other than one of the standard jms_xxxx values, it is instantiated as a JMS text or bytes message and a URI-style JMSType is assigned to it. The receiving application can read this using the getJMSType method.

Glossary of terms and abbreviations

This glossary defines WebSphere Data Interchange terms and abbreviations used in this book. If you do not find the term you are looking for, see the index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute. Copies may be ordered from the American National Standards Institute, 11 West 42 Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

A

AAR. Association of American Railroads. Represents the railroad industry in areas such as standards, public relations, and advertising.

acknowledgment. See *functional acknowledgment*, *network acknowledgment*.

ADF. *See data format*.

ANSI. American National Standards Institute.

ANSI ASC X12. ANSI Accredited Standards Committee X12, which develops and maintains generic standards for business transactions for EDI.

application. A program that processes business information. An application that requests services from WebSphere Data Interchange is an enabled application.

application data. The actual data in an application data file.

application data format. See *data format*.

application default profile. Identifies business applications, such as purchasing and accounts receivable, to WebSphere Data Interchange and sets specific WebSphere Data Interchange processing defaults for an application.

B

base structure. The data structure that contains all the data structures and data fields that define the application data for a single transaction.

binary format (BIN). Representation of a decimal value in which each field must be 2 or 4 bytes long. The sign (+ or -) is in the far left bit of the field, and the number value is in the remaining bits of the field. Positive numbers have a 0 in the sign bit. Negative numbers have a 1 in the sign bit and are in twos complement form.

C

CICS. Customer Information Control System.

CD-ROM. Compact Disk-Read Only Memory; a storage medium for large amounts of data needed external to the personal computer.

client-server. A computing environment in which two or more machines work together to achieve a common task.

code list. A table, supplied by WebSphere Data Interchange or defined by the user, that contains all acceptable values for a single data field.

composite data element. In EDI standards, a group of related subelements, such as the elements that make up a name and address.

compound element. An item in the source or target document that contains child items. Examples are EDI segments and composite data elements, data format records and structures, and XML elements.

control number. Numbers (or masks used to create numbers) that are used to identify an interchange, group, or EDI transaction.

control string. An object compiled from a map, data format, and EDI standard transaction; it contains the instructions used by the translator to translate a document from one format to another.

Glossary

control structure. The beginning and ending segments (header and trailer) of standard enveloped transmissions.

Customer Information Control System (CICS). An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs.

customize. To alter to suit the needs of a company, such as removing from an EDI standard the segments and data elements that the company does not use.

D

data dictionary. A file containing the definitions of all the data elements of an EDI standard.

data element. A single item of data in an EDI standard, such as a purchase order number. Corresponds to a data field in a data format.

data element delimiter. A character, such as an asterisk (*), that follows the segment identifier and separates each data element in a segment. See also *element separator* and *segment ID separator*.

data field. A single item of data in a data format, such as a purchase order number. Corresponds to a data element in an EDI standard.

data format. A description of the application data for a particular transaction. A data format is composed of loops, records, data structures, and fields.

data format dictionary. A file that contains data format components.

data format record. A group of logically related fields set up as a record in a data format.

data format structure. A group of related data fields in a data format, such as the fields making up the line item of an invoice. Corresponds to a composite data element in an EDI standard.

DataInterchange/MVS. The IBM DataInterchange product used on the host; pieces include a TSO parameter entry mechanism and a

translator. The functionality available in this product is now available in WebSphere Data Interchange for z/OS.

DataInterchange/MVS-CICS. The CICS-based IBM DataInterchange product. The functionality available in this product is now available in WebSphere Data Interchange for z/OS.

data structure. A group of related data fields in a data format, such as the fields making up the line item of an invoice. Corresponds to a segment in a standard.

data transformation map. One of three supported map types. A data transformation map is a set of mapping instructions that describes how to translate data from a source document into a target document. Both the source and target documents can be one of several support document types.

DB2. Database 2, an IBM relational database management system.

ddname. Data definition name.

decimal notation. The character that represents a decimal point in the data.

delimiter. A character that terminates a string of characters, such as the value contained in a data element.

DI Client. WebSphere Data Interchange Client; the Windows-based, client/server interface for WebSphere Data Interchange.

dictionary. See *data dictionary*.

document. A business document that is exchanged between two enterprises as part of a business process, such as a purchase order or invoice. A document within WebSphere Data Interchange is singular. For example, it cannot contain multiple purchase orders. A document can also be represented in any syntax. For example, an XML purchase order and an EDI purchase order are both documents.

Document Type Definition (DTD). A list of all components included in the XML document and

their relationship to each other. This defines the structure of an XML document.

domain. The data structure or group of data structures in a data format to and from which you should restrict the mapping of EDI repeating segments and loops.

DTD. See *Document Type Definition*.

E

EDI. Electronic data interchange.

EDIA. Electronic Data Interchange Association.

EDI administrator. The person responsible for setting up and maintaining WebSphere Data Interchange.

EDI message. See message.

EDI standard. The industry-supplied, national, or international formats to which information is converted, allowing different computer systems and applications to interchange information.

EDI transaction. A single business document, such as an invoice.

EDI transaction set. A group of logically related data that make up an electronic business document, such as an invoice or purchase order.

EDIFACT. Electronic Data Interchange for Administration Commerce and Transport. See UN/EDIFACT.

electronic data interchange (EDI). A method of transmitting business information over a network, between business associates who agree to follow approved national or industry standards in translating and exchanging information.

electronic transmission. The means by which information is transferred between parties, such as over a public network.

element. See *data element*.

element separator. A character that separates the data elements in a segment. See also *data element delimiter*.

encryption. The encoding and scrambling of data. Data is encrypted by the sender and decrypted by the receiver using a predetermined program and unique electronic key.

event. An occurrence that is important to a user's computer tasks, such as a software error, sending a transaction, or acknowledging a message.

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that was derived from, and is a subset of SGML. It is used to represent structured documents and data.

F

field. See *data field*.

floating segment. A segment of an EDI standard that may exist in many positions relative to other segments.

forward translation table. A user-defined table that translates data values that differ between trading partners. For example, if a manufacturer and supplier have different part numbers for the same item, each company can use its own part number and have it converted to the other company's part number during translation. Forward translation tables translate local values to standard values.

functional acknowledgment. An electronic acknowledgment returned to the sender to indicate acceptance or rejection of EDI transactions.

functional group. One or more transaction sets of a similar type transmitted from the same location, enclosed by functional group header and trailer segments.

Glossary

G

global variable.. A variable that is shared among all instances of all documents within a translation session.

H

header. A control structure that indicates the start of an electronic transmission.

hierarchical loop. A technique for describing the relationship of data entities which are related in a parent/child manner, like a corporate organization chart. Used in mapping to group related data elements and segments such as trading partner address.

HL. *See hierarchical loop*.

I

IBM Global Network. The IBM communications network that provides products and services to IBM customers.

ICS. International Control Segments.

import. The process of taking WebSphere Data Interchange objects exported on another WebSphere Data Interchange system and incorporating them into the receiving system.

Information Exchange. A commerce engine of IBM Interchange Services for e-business that permits users to send and receive information electronically.

interchange. The exchange of information between trading partners.

J

JCL. Job Control Language.

K

key. In a profile member, the field that identifies the member. For example, the key for members of the trading partner profile is the trading partner nickname.

L

literal. In mapping, a value that is constant for each occurrence of the translation. If you provide the literal value during mapping, the translator does not have to refer repeatedly to the source to obtain the value.

local variable. A variable that is specific to the instance of the document in which it is being used.

log file. A file in which events are recorded.

logging. The recording of events in time sequence.

loop. A repeating group of related segments in a transaction set or a repeating group of related records and loops in a data format.

loop ID. A unique code identifying a loop and the number of times the group can be repeated.

loop repeat. A number indicating the maximum number of times a loop can be used in a transaction set.

M

mailbox. If you use a mail type protocol to exchange messages with your trading partners, you will have one or more registered mailboxes. The mailbox profile is used in WebSphere Data Interchange to define your mailboxes and any associated preferences.

map. A set of instructions that indicate to WebSphere Data Interchange how to translate data from one format to another.

map rule. An association between a data transformation map and a trading partner.

maximum use. A number indicating the maximum number of times a segment can be used in a transaction set or the maximum number of times that a data format loop or record can repeat.

message. A free-form, usually short, communication to a trading partner. In UN/EDIFACT standards, a group of logically related data that make up an electronic business document, such as an invoice. A message is equivalent to a document.

message log. The file in which WebSphere Data Interchange Client logs messages about errors that occur within the client.

multiple-occurrence mapping. A form of mapping in which all occurrences of a loop or repeating segment are mapped to the same repeating structure in the data format.

N

network acknowledgment. A response from the network indicating the status of an interchange envelope, such as sent or received.

network commands. The commands that you want WebSphere Data Interchange to pass to your network, defined in the network commands profile. In the host product, this file is named NETOP.

O

ODETTE. Organization for Data Exchange through Teletransmission in Europe.

P

parse. To break down into component parts.

path qualified mapping. A form of mapping in which all occurrences of a repeating compound or simple data element are mapped to a repeating compound or simple data element in another document.

PDS. Partitioned data set.

PDS members. Groups of related information stored in partitioned data sets.

profile. Descriptive information about trading partners, network connections, and so on. Each profile can contain one or more objects or members. For example, the trading partner profile contains members for your trading partners (one member for trading partner address).

program directory. A document shipped with each release of a product that describes the detailed content of the product.

Q

qualifier. A data element which gives a generic segment or data element a specific meaning. Qualifiers are used in mapping single or multiple occurrences.

R

receive map. One of three supported map types. A receive map is a set of mapping instructions that describe how to translate an EDI standard transaction into a proprietary application data document.

receive usage. An association between a receive map and a trading partner.

record. A logical grouping of related data structures and fields.

release character. The character that indicates that a separator or delimiter is to be used as text data instead of as a separator or delimiter. The release character must immediately precede the delimiter.

repository data. A group of data definitions, formats, and rules/usages, that WebSphere Data Interchange uses to process your data.

requestor. See *mailbox*.

reverse translation table. A user-defined table that translates data values that differ between trading partners. For example, if a manufacturer and supplier have different part numbers for the

same item, each company can use its own part number and have it converted to the other company's part number during translation. Reverse translation tables translate standard values to local values.

rule. See *map rule*.

runtime data. Data used by the WebSphere Data Interchange translator, such as control strings, code lists, translation tables and profiles.

S

security administrator. The person who controls access to business data and program functions.

segment. A group of related data elements. A segment is a single line in a transaction set, beginning with a function identifier and ending with a segment terminator delimiter. The data elements in the segment are separated by data element delimiters.

segment directory. A file containing the format of all segments in an EDI standard.

segment identifier. A unique identifier at the beginning of each segment consisting of two or three alphanumeric characters.

segment ID separator. The character that separates the segment identifier from the data elements in the segment.

segment terminator. The character that marks the end of a segment.

send map. On of three supported map types. A send map is a set of mapping instructions that describe how to translate a proprietary application data document into an EDI standard transaction.

send usage. An association between a send map and a trading partner.

simple element. An item in the source or target document that does not contain child items, only data. Examples are EDI data elements, data format fields, XML attributes, and PCDATA values.

single-occurrence mapping. A form of mapping in which each occurrence of a loop or repeating compound or simple data element in a document is mapped to a different compound or simple data element in another document.

source document definition. A description of the document layout that will be used to identify the format of the input document for a translation.

special literal. The send and receive Mapping Data Element Editors include the Literal or Mapping Command field. Literals are constant values you enter in this field, such as 123. Special literals are values you enter in this field that begin with an ampersand (&) and are command to WebSphere Data Interchange, rather than constant values. For example, to use today's date, you enter &DATE.

standards. See *EDI standard*.

structure. See *data structure* or *data format structure*.

subelement. In UN/EDIFACT standards, a data element that is part of a composite data element. For example, a data element and its qualifier are subelements of a composite data element.

subelement separator. A character that separates the subelements in a composite data element.

T

tag. In UN/EDIFACT standards, the segment identifier. In export/import, a code identifies each field in the export record. Such export/import files are known as "tagged" files.

target document definition. A description of the document layout that will be used to create an output document from a translation.

TD queue. See *transient data queue*.

TDCC. Transportation Data Coordinating Committee.

TDQ. Transient data queue.

temporary storage queue (TS). Storage locations reserved for immediate results in CICS. They are deleted after the task that created them is complete and they are no longer necessary.

TPT. Trading partner transaction. See *map*.

trading partner profile. The profile that defines your trading partners, including information about network account numbers, user IDs, who pays for network charges, etc.

trading partners. Business associates, such as a manufacturer and a supplier, who agree to exchange information using electronic data interchange.

trading partner transaction. See *map*.

trailer. A control structure that indicates the end of an electronic transmission.

transaction. A single business document, such as an invoice. See also *EDI transaction*.

transaction set. A group of standard data segments, in a predefined sequence, needed to provide all of the data required to define a complete transaction, such as an invoice or purchase order. See also *EDI transaction set*.

Transaction Store. The file that contains the results of translations and a history of translation activity.

transform. The process of converting a document from one format to another.

transient data queue (TD). A sequential data set used by the Folder Application Facility in CICS to log system messages.

translation. The process of converting a document from one format to another.

translation table. A user-defined table that translates data values that differ between trading partners. For example, if a manufacturer and supplier have different part numbers for the same item, each company can use its own part number and have it converted to the other company's part number during translation.

TSQ. See *temporary storage queue*.

U

UCS. Uniform Communication Standard.

unary operator. An operator that changes the sign of a numeric value.

UN/EDIFACT. United Nations Electronic Data Interchange for Administration Commerce and Transport.

Uniform Communication Standard (UCS). The EDI standard used in the grocery industry.

UN/TDI. United Nations Trade Data Interchange.

Usage. An association between a send or receive map and a trading partner.

V

validation table. A table, supplied by WebSphere Data Interchange or defined by the user, which contains all acceptable values for a single data field.

variable. The entity in which a value may be stored based on data received; as opposed to a constant value.

W

WebSphere Data Interchange. A generic term for the WebSphere Data Interchange products, WebSphere Data Interchange for z/OS and WebSphere Data Interchange for Multiplatforms. WebSphere Data Interchange is a translator of data from one document format to another; the pieces of this product include a TSO parameter entry mechanism, a CICS parameter entry mechanism, a Windows-based parameter entry mechanism (WebSphere Data Interchange Client), and a translator.

WebSphere Data Interchange Client. A Windows-based product for entry of parameters needed by the WebSphere Data Interchange translator.

Glossary

WebSphere MQ. An IBM product that is used to implement messaging and queueing of data groups. Earlier releases of this product were known as MQSeries.

WebSphere MQ queue profile. Represents a relationship between a logical name and a physical WebSphere MQ queue name.

WINS. Warehouse Information Network Standard.

Windows®. Microsoft's graphical operating system under which WebSphere Data Interchange Client runs.

X

X12. A common EDI standard approved by the American National Standards Institute.

XML. See *Extensible Markup Language*.

Bibliography

This section describes the documentation available for the WebSphere Data Interchange product.

WebSphere Data Interchange publications

The WebSphere Data Interchange publications are:

- WebSphere Data Interchange for z/OS Administration Guide SC34-6214
- WebSphere Data Interchange User's Guide SC34-6215
- WebSphere Data Interchange Messages and Codes SC34-6216
- WebSphere Data Interchange Programmer's Reference SC34-6217
- WebSphere Data Interchange for z/OS Installation Guide SC34-6999

You can order publications from the IBMLink™ Web site at:

`http://www.ibm.com/ibmlink`

In the United States, you can also order publications by dialing 1-800-879-2755.

In Canada, you can order publications by dialing 1-800-IBM-4YOU (1-800-426-4968).

For further information about ordering publications contact your IBM authorized dealer or marketing representative.

Softcopy books

All the WebSphere Data Interchange books are available in softcopy formats.

Portable Document Format (PDF)

The library is supplied as stand-alone PDFs in US English in the DOC directory on the product CD. The contents of the DOC directory can be viewed without installing the product.

PDF files can be viewed and printed using the Adobe Acrobat Reader. You will need Adobe Acrobat Reader with Search Version 4.05 on Windows NT, or Adobe Acrobat Reader with Search Version 4.5 on UNIX® systems.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. web site at:

`http://www.adobe.com/`

If you cut and paste examples of commands from PDF files to a command line for execution, you must check that the content is correct before you press Enter. Some characters might be corrupted by local system and font settings.

WebSphere Data Interchange information available on the Internet

The WebSphere Data Interchange product Web site is at:

`http://www.ibm.com/websphere/datainterchange/`

By following links from this Web site you can:

- Obtain latest information about the WebSphere Data Interchange products.
- Access the WebSphere Data Interchange books in PDF format.

WebSphere Data Interchange on the Internet

Index

Special characters

& 133, 472

+ 589

Numerics

000 record 215
0C1/0C2 record 214
1Y2 record 234
1Y3 record 234
1Y4 record 235
1Y5 record 235
1Y6 record 236
1Y7 record 236
1Y8 record 237
1Y9 record 237
1YA record 238
2W1 record 240
2W2 record 241
2W3 record 241
2W4 record 242
2W5 record 243
2W6 record 243
2W7 record 243
2W8 record 244
2W9 record 244
2WA record 245
2WB record 245
3T5 record 254
3T6 record 255
3T7 record 256
3V1 record 248
3V2 record 249
3V3 record 250
3V9 record 251
3VA record 251
3VB record 252
3VC record 252
3VD record 252
3VE record 253
7A1 record 231
7P1 record 217
7P2 record 218
7P3 record 231
7P4 record 231
7Z1 record 232

A

abends, CICS return codes 429
account key
    alternate 565
    full 564

acknowledgment images data extracts, record
  layout 310
activity log profile 266
adapter user exits 587
administrative data
    exporting 12
    importing 12
AJ1 record 262
AJ2 record 262
AJ3 record 262
API 589
    business tasks 436
    cancel send 555
        parameters 556
    close and queue interchange
        parameters 516
    commit work parameters 572
    data extraction parameters 535
    deenveloping 518
        parameters 519
    end translation/enveloping 517
        parameters 518
    ENVELOPE AND SEND command 437
    enveloping 503, 504
        parameters 504
    environmental services
        initializing 438
        initializing parameters 439
        initializing request results 439
        terminating 441
        terminating parameters 442
        terminating request results 442
    example 594
    get envelope parameters 574
    initialize SYNC parameters 570
    internal calls 558
    issue commit 532
    issue commit parameters 532
    languages 426
    link edit 426
    process network acknowledgments parameters 559
    put envelope parameters 575
    queue standard data 558
    receive parameters 552
    receive transactions 552
    restart receive parameters 552
    restart receive transactions 552
    restart send transactions 544
        parameters 545
    retrieve detailed data 536
    retrieve group header parameters 533
    retrieve interchange header parameters 533

internal
   calls   558
   translation, outbound   472
invoking WebSphere Data Interchange Utility using
  EDIW   404
ISPF functions   213
issue commit API   532
issue commit parameters   532

J

JCL
   samples   407
JCL DISP option   196
JCL parameter example   17
JCL samples   407, 408, 409, 419, 420
JMS implementation   807

K

keys
   account number and user ID
      alternate keys   565
      full key   564
   full envelope   563
   interchange   561
      alternate   562, 565
   interchange ID and qualifier
      alternate key   566
      full key   564
   trading partner nickname   563
   transaction ID   565
   transaction store, common data extract   302
keywords
   APPLID, z/OS environment   15
   case sensitive   193
   DATE   21
   DLM   16
   HANDLE   21
   LANGID   16
   MQEXCP   16
   MQPRT   16
   MQQUERY   16
   MQRPT   16
   MQSYSIN   16
   MQTRAK   16
   overview   19
   PLAN   17
   RECOVERY   356
   SYSID   16
   SYSTEM   17
   TIME   21
   USERPGM   11

L

LANGID keyword   16
language profile layout   268

languages
   exit   312
last call for transaction   463, 486, 498
last call of session   466, 486, 498, 513, 529
layers
   group   502
   interchange   501
   transaction   502
libraries   631
link edit requirements   426
linkage editor   312
literals
   &   472
load libraries   631
   members   631
LOAD LOG ENTRIES command   44
   example   44
load modules diagram   426
locating trading partner profiles   529, 530
logs, activity layout   266
loops
   data format   242
   details record data format   244

M

mailbox (requestor) profile block (REQDB)   706
   field descriptions   707
mailbox profiles   263
maintenance
   file   804
   profiles   13
management reports
   categories   8
   creating   10, 11, 12
   data extract layouts   297
managing data   6
maps
   associated objects   246
   export/import   245
   exporting   12
   importing   12
   receive   3
   records
      application control   251
      commands   253
      data transformation map rule   256
      element   250
      global variables   253
      header   248
      local variables   252
      nodes   252
      receive usage   255
      reference   252
      segment   249
      send usage   254

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Burlingame Laboratory Director

IBM Burlingame Laboratory
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not necessarily tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE

This information may contain sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Websphere Business Integration Connect contains code named ICU4J which is licensed to you by IBM under the terms of the International Program License

Agreement, subject to its Excluded Components terms. However, IBM is required to provide the following language to you as a notice:

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2003 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the ″Software″), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED ″AS IS″, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

IBM
the IBM logo
AIX
CICS
DB2
DB2 Universal Database
IBMLink
IMS
MQSeries
MVS
OS/390
WebSphere
z/OS

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

IBM ®

®