WebSphere® WebSphere Data Interchange for MultiPlatforms

IBM

Version 3.3

Programmer's Reference Guide

SC34-6217-01

WebSphere® WebSphere Data Interchange for MultiPlatforms

IBM

Version 3.3

Programmer’s Reference Guide

SC34-6217-01

Note!

Before using this information and the product it supports, read the information in “Notices” on page 547.

March 2007

This edition applies to IBM WebSphere Data Interchange for MultiPlatforms, V3.3. and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this documentation, email doc-comments@us.ibm.com. We look forward to hearing from 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 3.3 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 *WebSphere Data Interchange for MultiPlatforms Messages and Codes Guide* 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
  ```

- 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 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 MultiPlatforms Quick Start Guide*, CF0YREN

  This document provides a brief overview of how to use WebSphere Data Interchange.
- *WebSphere Data Interchange for MultiPlatforms Administration and Security Guide*, SC34-6214-01

  This document provides information on administrative tasks you will use in WebSphere Data Interchange.
- *WebSphere Data Interchange for MultiPlatforms Messages and Codes Guide* , SC34-6216-01

  This book provides information to assist you in diagnosing errors.
- *WebSphere Data Interchange for MultiPlatforms User's Guide*, SC34-6215-01

  This book provides information on the WebSphere Data Interchange Client/Server user interface.
- *WebSphere Data Interchange for MultiPlatforms Programmer's Reference Guide*, SC34-6217-01

  This document provides detailed technical information about WebSphere Data Interchange.
- *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00

  This document provides instructions for your WebSphere Data Interchange mapper.
- *WebSphere Data Interchange for MultiPlatforms Utility Commands and File Formats Reference Guide*, SC23-5873-00

  This document provides the commands and file formats necessary to use WebSphere Data Interchange.
- *WebSphere Data Interchange for z/OS V3.3 Installation Guide*, SC34-6269-01

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

Chapter 1. WebSphere Data Interchange Utility file specifications

This chapter describes the files used by the WebSphere Data Interchange Utility and the specifications of each file. "Using sample JCL" on page 23 also contains information about the WebSphere Data Interchange Utility files, and "Required utility data sets" on page 37 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 creates a unique file name for the application file and assigns this unique physical name to the logical file name specified on the PERFORM command. The new file is 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.

Utility 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 Utility 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 is not case sensitive except for values associated with the following keywords:

- ACFIELD
- ADDRLN1
- ADDRLN2
- CMMTLN1
- CMMTLN2
- CMPYNM
- CNCTNM
- CNCTPH
- NETNAME
- STDDESC

For information about PERFORM commands, see *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

The WebSphere Data Interchange Utility first verifies that EDISYSIN is allocated. If so, the logical name EDISYSIN is used. If EDISYSIN does not exist, the logical name SYSIN is used. 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:

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 WebSphere MQ 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.

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 or detach DB2. For more information, see the Programming Guide. 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 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 contains the DB2 control information. If EXEC PGM=IKJEFT01 is used, SYSTSIN contains 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, the records are 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.
- 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 or translated.

Specified
Sending: TD queue in network profile or overridden by the `FILEID` specified on the command.

Receiving: receive file in mailbox (requestor) 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. See *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00 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 writes 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 writes 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 6.

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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

Specified

Does not apply.

logical name

FFSEXCP. FFSEXCP takes precedence and is used if allocated. You can override this logical name and use a WebSphere MQ queue instead. You can use the MQEXCP parameter to specify a WebSphere Data Interchange WebSphere MQ queue profile member to use instead of a sequential file.

In CICS, the exception file name and type are specified in the WebSphere Data Interchange Utility control information.

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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

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 a WebSphere MQ queue instead. You can specify a WebSphere Data Interchange WebSphere MQ 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. See*WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00 for more information.

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 a WebSphere MQ queue instead. You can specify a WebSphere Data Interchange WebSphere MQ 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 for MultiPlatforms Mapping Guide*, SC23-5874-00.

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.

ADF print file (ADFPRNT)

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
ADFPRNT. ADFPRNT takes precedence and is used if allocated. You can override this logical name and use a WebSphere MQ queue instead. You can specify a WebSphere Data Interchange WebSphere MQ 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 for MultiPlatforms Mapping Guide*, SC23-5874-00.

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 1 shows a sample PRTFILE.

```
1Audit Trail Report -DataInterchange Utility- Date: 05/04/07 Time:
10:48:19 Page: 00001
 FF0588 Command: PERFORM TRANSFORM WHERE INFILE(XMLFILE)
OUTFILE(OUTFILE) SYNTAX(X) CLEARFILE(Y) TRACELEVEL(A2)

      Message: RU0003 Severity: 00
         The best rule match for the document was: map name POXML5SR-EDI,sending TP nickname
ANY, receiving TP nickname ANY, usage indicator P, document POXML5SR, dictionary name TESTS,
syntax xml.
      Message: UT0008 Severity: 00
          Map name being processed: POXML5SR-EDI.

 FF0007 Data was written to OUTFILE. Message control number or document
id was 000000009.
 FF0585 The PERFORM TRANSFORM command completed successfully.
```

Figure 1. Sample PRTFILE

XML print file (XMLPRNT)

The XML 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
XMLPRNT. XMLPRNT takes precedence and is used if allocated. You can override this logical name and use a WebSphere MQ queue instead. You can specify a WebSphere Data Interchange WebSphere MQ 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 for MultiPlatforms Mapping Guide*, SC23-5874-00.

Suggested format
Record format: Variable. 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. A default schema is provided for XML Print Files

Figure 2 on page 10 shows a sample XMLPRNT.

```
<?xml version="1.0" encoding="UTF-8"?>
<WDILog>
   <Header>
      <Date>
         <Year>2005</Year>
         <Month>04</Month>
         <Day>07</Day>
      </Date>
      <Time>
         <Hour>10</Hour>
         <Minute>48</Minute>
         <Second>19</Second>
      </Time>
   </Header>
   <Message>
      <MsgId>FF0588</MsgId>
      <Severity>0</Severity>
      <Text>Command: PERFORM TRANSFORM WHERE INFILE(XMLFILE)
OUTFILE(OUTFILE) SYNTAX(X) CLEARFILE(Y) TRACELEVEL(A2)</Text>
      <InsertData>PERFORM TRANSFORM WHERE INFILE(XMLFILE)
OUTFILE(OUTFILE) SYNTAX(X) CLEARFILE(Y) TRACELEVEL(A2)</InsertData>
   </Message>
   <DocInfo>
      <DocId>1111111111</DocId>
      <SenderTPNick>ANY</SenderTPNick>
      <SenderId>ANY</SenderId>
      <SenderQual>ANY</SenderQual>
      <ReceiverTPNick>ANY</ReceiverTPNick>
      <ReceiverId>ANY</ReceiverId>
      <ReceiverQual>ANY</ReceiverQual>
      <Syntax>xml</Syntax>
      <Dictionary>TESTS</Dictionary>
      <Document>POXML5SR</Document>
   </DocInfo>
   <Message>
      <MsgId>RU0003</MsgId>
      <DocId>1111111111</DocId>
      <Severity>0</Severity>
      <Text>The best rule match for the document was: map name POXML5SREDI,
sending TP nickname ANY, receiving TP nickname ANY, usage
indicator P, document POXML5SR, dictionary name TESTS, syntax
xml.</Text>
      <InsertData>POXML5SR-EDI</InsertData>
      <InsertData>ANY</InsertData>
      <InsertData>ANY</InsertData>
      <InsertData>P</InsertData>
      <InsertData>POXML5SR</InsertData>
      <InsertData>TESTS</InsertData>
      <InsertData>xml</InsertData>
   </Message>
   <Message>
      <MsgId>UT0008</MsgId>
      <DocId>1111111111</DocId>
      <Severity>0</Severity>
      <Text>Map name being processed: POXML5SR-EDI.</Text>
      <InsertData>POXML5SR-EDI</InsertData>
   </Message>
   <DocInfo>
```

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 Document 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 a WebSphere MQ queue instead. You can specify a WebSphere Data Interchange WebSphere MQ 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 for MultiPlatforms Mapping Guide*, SC23-5874-00.

Suggested format
Record format: FBA or VBA. Record length: 132 bytes.

Remarks
Reports requested from the Document 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 following commands:

- QUERY
- ENVELOPE DATA EXTRACT
- NETWORK ACTIVITY DATA EXTRACT
- TRADING PARTNER CAPABILITY DATA EXTRACT
- TRADING PARTNER PROFILE DATA EXTRACT
- TRANSACTION ACTIVITY DATA EXTRACT
- TRANSACTION DATA EXTRACT

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 a WebSphere MQ queue instead. You can use the `MQQUERY` parameter specify a WebSphere Data Interchange WebSphere MQ 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 for MultiPlatforms Mapping Guide*, SC23-5874-00.

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.

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.

XML work file (XMLWORK)

This optional work file is used for the XML Split function which will take a single XML document and create multiple XML documents. The Data Transformation processing uses the work file to hold records to be used to split and reconstruct XML documents.

Note: This function is not available in CICS.

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
XMLWORK.

Suggested format
Record format: Variable blocked (VB). Record length: 32756 bytes.

Remarks
None.

WebSphere Data Interchange work file (EDIPARSE)

large source files containing multiple source documents. The Data Transformation processing will use the work file to hold one source document as opposed to using internally allocated buffers when the source document is greater than 64MB.

Note: This function is not available in CICS.

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
EDIPARSE.

Suggested format
Record format: Variable blocked (VB). Record length: 32756 bytes.

Remarks
None.

Pageable translation work 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 ensures 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.

Data transformation pageable work EDIWORK)

The EDIWORK file is used to reduce the amount of memory used during data transformation. By paging out (writing) output data that usually resides in memory to the EDIWORK file, memory usage is reduced. Using the PAGE(Y) keyword on PERFORM TRANSFORM command causes the output data to be paged to a work file (EDIWORK) once a size threshold is met. The work file offset is passed back to the Utility, where the data is read from the work file, and written to the specified output file.

There are two places in the Message Broker that check the output size threshold in order to possibly page the output. If a single output document reaches 10MB, then the document is released from memory and written to the work file. If a collection of smaller output documents reaches 10MB, then any remaining output documents are written to the work file.

The first case occurs in EDIAMMSG AppendToBuffer, and the second case occurs in EDIMBCNI AddToOutList.

If the PAGE(Y) keyword is specified on a PERFORM TRANSFORM command and file EDIWORK is not defined or allocated or if there is a problem opening the file, a severity-4 message is logged. In this case, processing would continue as if PAGE(N) was specified.

If an error occurs writing to the EDIWORK file, then a severity-8 message is logged and the translation terminates.

The type of output data that can be paged is not restricted. The PAGE(Y) keywords applies to ADF, EDI, and XML output data.

There is a size restriction on the amount of data that can be paged to EDIWORK. The current restriction is 2GB. If this size is exceeded, a severity-8 message is logged and the translation terminates. Because the EDIWORK file can be reused each time the Utility calls the Message Broker (which occurs multiple times during a PERFORM command), the EDIWORK file cannot be defined as MOD or as an append-to-end-of-file file. The base EDIWORK file offset is reset to zero each time the Utility calls the Message Broker.

Note: PAGE(Y) is not valid in CICS on PERFORM TRANSFORM commands.

EDI pageable work file (EDIPAGE)

The EDIPAGE is used to reduce the memory requirement for large EDI translations. By paging out (writing) data that normally resides in memory to the EDIPAGE file, memory usage is reduced. Using the Abstract Message Model (AMM), a WebSphere Data Interchange internal representation of data during translation, AMM nodes that are part of repeating data are paged out. When there is a repeating loop, element, segment, structure, or record, the AMM subtree containing the repeating data is paged out of memory when the number of repetitions exceeds a given threshold and the data is no longer immediately needed. The default threshold is 1000. This threshold can be overridden using the PAGETHRESHOLD keyword. Using the PAGETHRESHOLD(0) keyword turns paging off.

The EDIPAGE file must be allocated or the Pageable AMM feature is not used. If the EDIPAGE file is not defined, a severity-0 message AM0016 is written to the Print File and processing continues normally.

If an error occurs opening the page file, transformation continues normally and a severity-0 message is logged. If an error occurs writing to the page file, transformation continues normally and a severity-4 message is logged.

The EDIPAGE file contains the internal representation of both the input and the output data. Typically, you should allow several times the total of the input and (expected) output file sizes when determining how much space to allocate. This allows for additional information (name, data type, parent-child pointers, and so on) that is kept in the file. The actual ratio depends on the document structure and the PAGETHRESHOLD value. For relatively verbose formats such as XML, the ratio is usually lower, around 2 or 3 times the data size. For more condensed formats such as EDI, the ratio is likely to be higher, such as 3 to 5 times the data size.

If the PAGETHRESHOLD is higher, more of each document is kept in memory before starting to write data to the EDIPAGE file, so the EDIPAGE file is smaller.

Note: The Pageable AMM function is primarily intended for reducing the memory used to process large individual documents (EDI transactions, Data Format transactions, and XML documents). Although it slightly reduces memory usage when processing large EDI interchanges or other files with many medium sized transactions (2–10 MB), it may not be worth allocating the large page file produced in this case since the EDIPAGE file accumulates from one transaction to the next, until the end of the input file is reached. Increasing the PAGETHRESHOLD in this case can significantly reduce the size of the EDIPAGE file, and keep more of each transaction in memory.

EDIPAGE can be allocated with any technique the user prefers. Some of the options are:

- To execute WebSphere Data Interchange without the PAMM feature, issue the PERFORM TRANSFORM command with the PAGETHRESHOLD(0) keyword.
- To execute WebSphere Data Interchange and reduce the memory used during transformation, that is start using the PAMM sooner, issue the PERFORM TRANSFORM command with the PAGETHRESHOLD(500) keyword.
- To execute WebSphere Data Interchange and delay using the PAMM, use the PERFORM TRANSFORM command with the PAGETHRESHOLD(1500) command.
- To allocate the EDIPAGE dataset in z/OS, use the following DD statement:

  ```
  //EDIPAGE DD DSN=&&tempfile,UNIT=SYSDA,SPACE=(CYL,XXX)
  ```
- To allocate the EDIPAGE file in Windows, issue the following command file statement:

  ```
  set file(EDIPAGE,..\..\..\edipage.text);
  ```

Notes:

1.
   a. The Pageable AMM feature is not available when using WebSphere Data Interchange in CICS. The primary reason for this is the use of a sequential file as the EDIPAGE file and handling of sequential files within CICS. A design point is also that Very Large Transactions, as those suited for the Pageable AMM feature, are more than likely not acceptable for transaction oriented CICS processing.
   b. The Pageable AMM feature does not apply to EDI transactions using Hierarchical Loops (HL segments).

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 acknowledgement 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 acknowledgement 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 1 are optional.

Table 1. 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 acknowledgement envelope
IRIDFA	35	Interchange receiver ID override for outbound functional acknowledgement envelope
GSIDFA	35	Group sender ID override for outbound functional acknowledgement envelope
GRIDFA	35	Group receiver ID override for outbound functional acknowledgement 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 2.

Table 2. 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 2. 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 3.

Table 3. 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 4 on page 21

Table 4. 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.

Chapter 2. 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.V3R3M0.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**.

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 44.

In DB2, change the parameter PLAN=EDIENU33 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.V3R3M0.SEDILMD1,DISP=SHR       <--EDI LOAD LIBRARY
//         DD DSN=SYS1.SNA.LOADLIB,DISP=SHR          <--COMM. LOAD LIBRARY
//         DD DSN=DD DSN=SYS1.XML.SIXMMOD1,DISP=SHR  <--XML ToolKit Version 1 Release 9
```

Section 4 Print Files

There are three print files in WebSphere Data Interchange:

- PRTFILE
- XMLPRNT
- ADFPRNT

PRTFILE is required for all WebSphere Data Interchange Utility functions. XMLPRNT and ADFPRNT are optional. Error messages generated during function processing and a summary report are written to PRTFILE. Error messages are written to the optional files if specified. 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 GXS Expedite Base/MVS using IEBASE: This section contains JCL statements for using the GXS 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 GXS Expedite Base/z/OS. INMSG is the network input file and contains commands written by WebSphere Data Interchange for processing by GXS Expedite Base/z/OS. OUTMSG is the network output file and contains the responses from GXS Expedite Base/z/OS after processing. INB1STAT is used during status updating to hold network acknowledgments. For more information about these GXS Expedite Base/z/OS files, refer to the GXS Expedite Base/MVS Programming Guide.

```
//*********************************************************************
//* Specify the IGN GXS 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 GXS Expedite Base/z/OS using IEBASE and Comm-Press: this section contains JCL statements for using the GXS 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 GXS Expedite Base/z/OS with compression. For more information about these GXS Expedite Base/z/OS files, refer to the GXS Expedite Base/MVS Programming Guide.

```
//*******************************************************************
//* If you are using GXS 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

The record format should be fixed with a logical record length of 84.

```
//********************************************************************
//* Step 1 - Allocate user supplied control file
//********************************************************************
//*
//EIMCTL   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 acknowledgement 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

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 Work files

Section 18a 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

```
//************************************************************
//* 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)
//*
//*************************************************************************
//* Specify the Pageable Translation work file for Data Transformation *
//*************************************************************************
//*EDIPAGE  DD DISP=(NEW,DELETE,DELETE),UNIT=SYSALLDA,SPACE=(CYL,25)
//*EDIWORK  DD DISP=(NEW,DELETE,DELETE),UNIT=SYSALLDA,SPACE=(CYL,25)
//*EDIVAX   DD DISP=(NEW,DELETE,DELETE),UNIT=SYSDA,SPACE=(CYL,1500)//*
```

Section 18b Message parsing work file: The message parsing work file is used to parse out 1 logical message from a large source file that contains multiple logical messages.

```
//*******************************************************************
//* Specify the message parsing work file.  Uncomment the following        *
//* DD statement if you use this feature.                          *
//*                                                                *
//* Suggested: DCB=(RECFM=V,LRECL=32760)                           *
//*******************************************************************
//*
//*EDIPARSE DD DSN=EDI.EDIPARSE.FILE,DISP=SHR
```

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 24 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 (EDIENU33) 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=EDIENU33" 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 (EDIENU33) -
  PARM('LANGID=ENU SYSID=DIENU APPLID=EDIFFS
PLAN=EDIENU33 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 9, replacing SYS1 with the high level qualifier for your system.

```
//STEPLIB          DD DSN=EDI.V3R3M0.SEDILMDI,DISP=SHR
                   DD DSN=EDI.V3R3M0.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 5 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 5. 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	

Table 5. Required utility data sets (continued)

Command	JCL sections					
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				
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.V3R3M0.SEDIINS1). For additional information, see *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

```
//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.V3R3M0.ARCFILE,DISP=(NEW,CATLG),
// DCB=(RECFM=V,LRECL=4096,BLKSIZE=0),
// UNIT=SYSALLDA,SPACE=(CYL,(10,5))
//HLDFILE DD DSN=EDIENU.V3R3M0.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.V3R3M0.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.V3R3M0.ARCFILE,DISP=SHR
//HLDFILE DD DSN=EDIENU.V3R3M0.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 (EDIENU33) -
PARM('LANGID=ENU SYSID=DIENU APPLID=EDIFFS
PLAN=EDIENU33 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)
//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 (EDIENU33.EDIELOG)
UNLDDN (UNLD)
WORKDDN (UNLD)
REORG INDEX (EDIENU33.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 55
- "Translation services" on page 60
- "Enveloping services" on page 118
- "Data extraction services" on page 153
- "Communication services" on page 158
- "Update status services" on page 179
- "SYNCPOINT services" on page 186

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 42.

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 370.

- 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 366.
- 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 374.

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 3 on page 43.

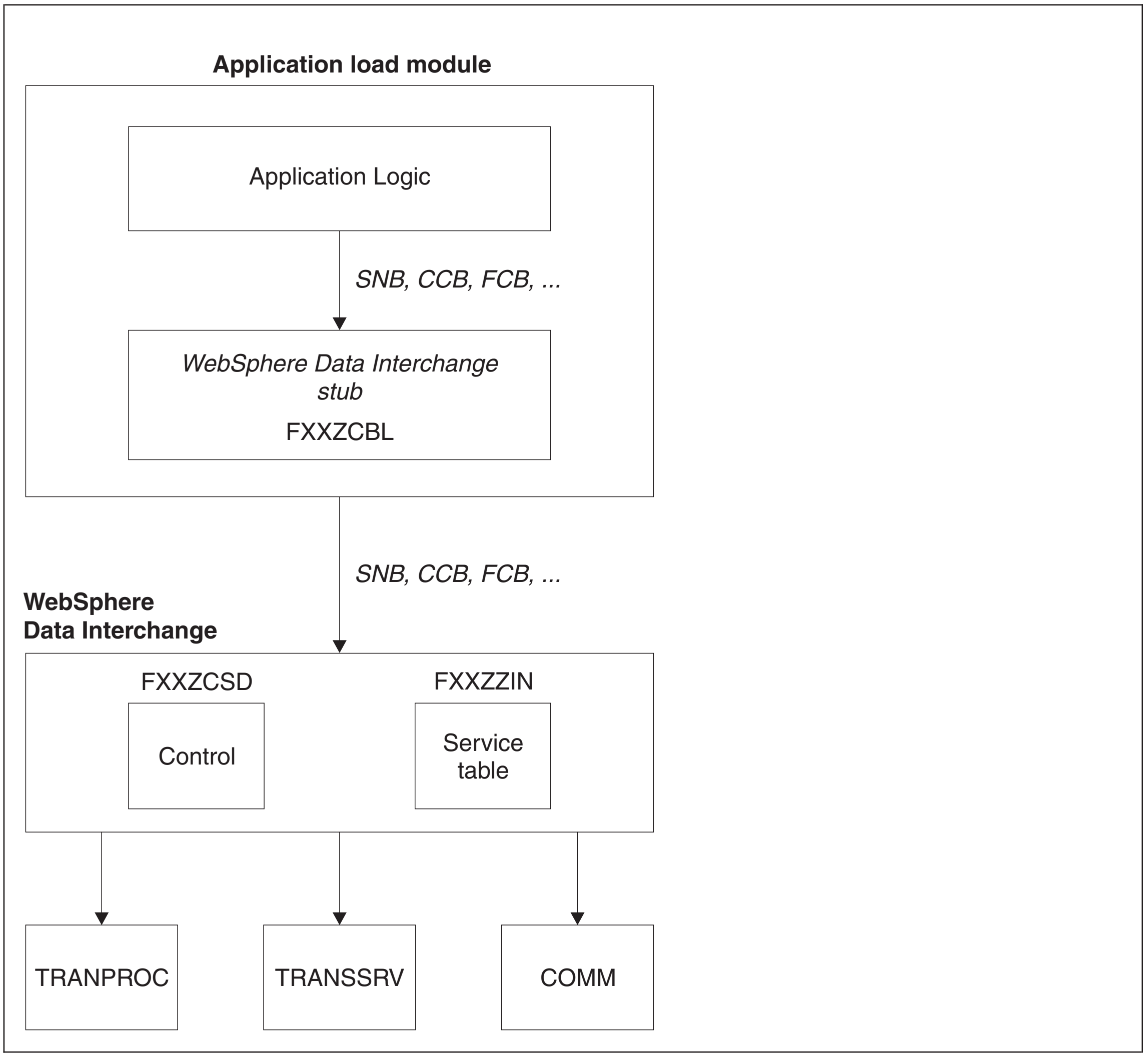


Figure 3. Load Module Relationships

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 value ″8″ (eight) causes WebSphere Data Interchange to use the Resource Recovery Services attachment facility rather than the call attachment facility. RRSAF is required whenever attaching to DB2 from a Workload Manager (WLM) administered address space. The default is **N**.

EDITSIN examples

Example 1

Your DB2 subsystem is named *DB93*, your DB2 plan is named *EDIENU33*, 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(EDIENU33)
```

Example 2

Your DB2 subsystem is named *DB93*, your DB2 plan is named *EDIENU33*, and you do not want WebSphere Data Interchange to attach or detach DB2. Include the following keywords and values in EDITSIN.

```
SYSTEM(DB93) PLAN(EDIENU33) OPEN(N) CLOSE(N)
```

Example 3

Your DB2 subsystem is named *DB93*, your DB2 plan is named *EDIENU33*, 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(EDIENU33) 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
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 55.
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 63.
3. Call the translation service to signal the end of translation. For detailed information, see “End translation/enveloping API” on page 135.
4. Call the communication service to send the transactions. For detailed information, see “Send transactions and restart send transactions API” on page 162.
5. Call the environmental service to terminate WebSphere Data Interchange. For detailed information, see “Terminating the API” on page 58.

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 3, "Using WebSphere Data Interchange in the CICS environment," on page 195.
- 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

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 6 lists the functions provided by the environmental service.

Table 6. 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). 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 7.

Table 7. 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. 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 8.

Table 8. Environmental services initialization return codes

RC	**Explanation**
0	Initialization was successful.

Table 8. 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 9.

Table 9. 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:

```
FXXZccc(SNB,CCB,FCB)
```

The unique parameters required for this function request are defined in Table 10

Table 10. 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 11.

Table 11. 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 Document 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 Document 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 Document Store is updated by all translation and enveloping operations, and is maintained and reported on by the Document Store Facility or the WebSphere Data Interchange Utility. Updating the Document 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 Document Store database consists of seven DB2 tables, described in Table 12 on page 61.

Table 12. Translation services DB2 tables

Table	Description
EDIVTSTH	The transaction handle table. This is the primary table in the Document 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 Document 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 acknowledgement 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 acknowledgement 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 12. *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 63 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 103 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 135 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 151 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 151 for more information.

- 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 152 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 152 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 55. 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	Envelope transaction
990	Close and QUEUE the current interchange
991	Issue 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 Document 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 63.
- The enveloping service (function code **215**) builds the proper service segments and retrieves the transaction image from the Document 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 121.
- 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 162.

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 66.

You can control whether enveloping is performed at the time of translation with the `ENVLDELAY` field in the TCB. You can only delay enveloping for translated transactions already stored in the Document 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 376.

Consider the following points when performing automatic enveloping:

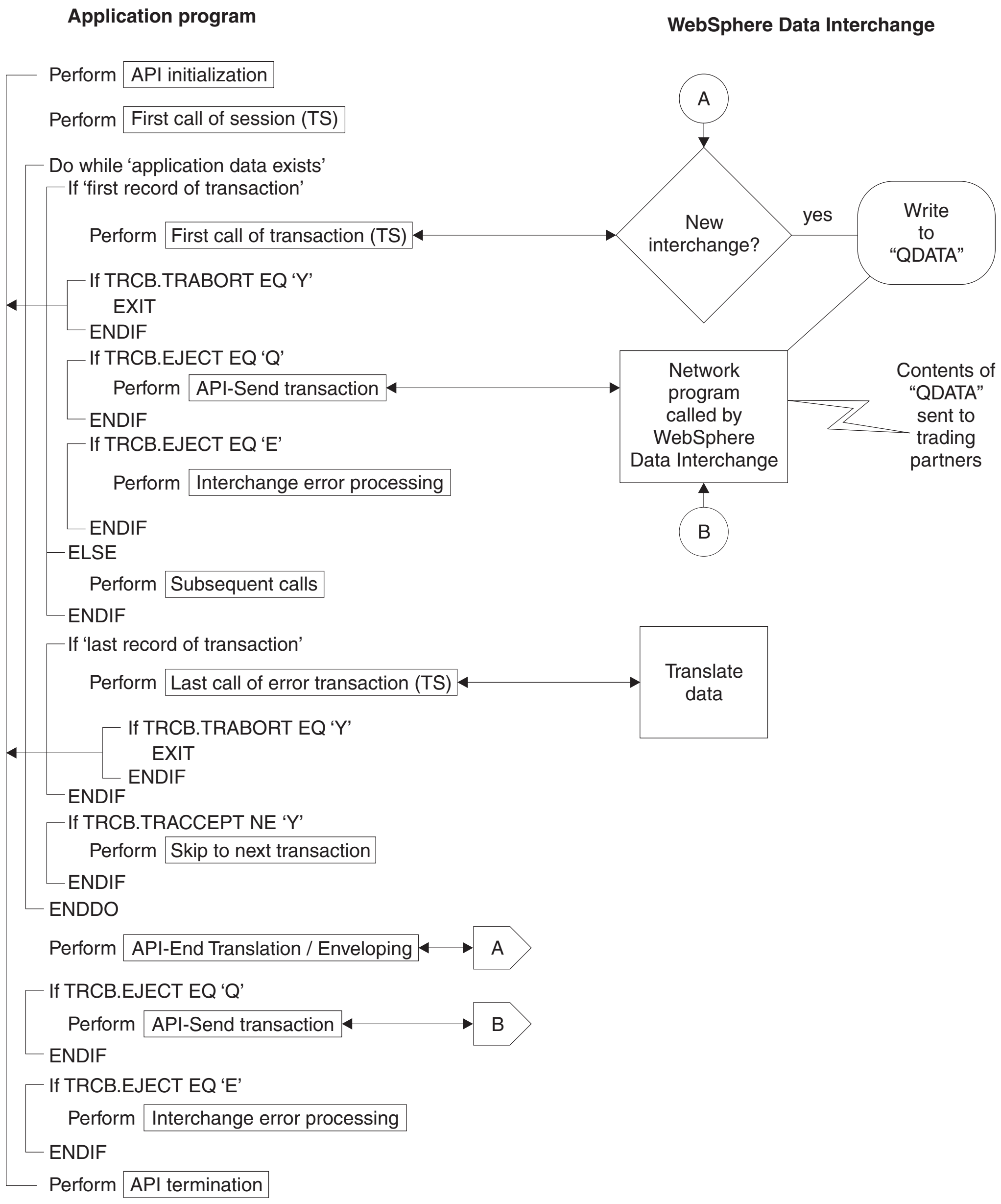
- All data involved in a translation, including the image of the EDI standard data produced, is saved in the Document Store. This occurs whether enveloping is part of the translation or is delayed. An EDI standard image is saved even when the translation is not successful, because the image might prove helpful in determining why the translation failed.
- If enveloping is performed 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 Document 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.

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

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)
     If TRCB.TRABORT EQ 'Y'
        EXIT
     ENDIF
     If TRCB.EJECT EQ 'Q'
        Perform API-Send transaction
     ENDIF
     If TRCB.EJECT EQ 'E'
        Perform Interchange error processing
     ENDIF
  ELSE
     Perform Subsequent calls
  ENDIF
  If 'last record of transaction'
     Perform Last call of error transaction (TS)
        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
If TRCB.EJECT EQ 'Q'
   Perform API-Send transaction
ENDIF
If TRCB.EJECT EQ 'E'
   Perform Interchange error processing
ENDIF
Perform API termination
```

WebSphere Data Interchange
A
New interchange?
yes
Write to "QDATA"
Network program called by WebSphere Data Interchange
Contents of "QDATA" sent to trading partners
B
Translate data
A
B

Figure 4. 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 Document Store. To post information to the Document 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 Document 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 mode

  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 83.

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 completes 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 Document Store, to prevent overcrowding 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 13.

Table 13. Transaction services translate-to-standard parameters

Parameter	Description
SNB	**ZSNBLL** 32 **ZSNBNAME** TRANPROC **ZSNBPC** 6
FCB	ZFCBFUNC **131** production **111** test

Table 13. Transaction services translate-to-standard parameters (continued)

Parameter	Description
TRCB	The translator control block. For more information about this block, see “Translator Control Block (TRCB)” on page 376.
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 14, Table 15, Table 16, and Table 17 on page 73.

Table 14. 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 15. Function code block (FCB) initialization for translate-to-standard

Field name	Initialization
ZFCBLL	**4**
ZFCBFUNC	**131** (production) **111** (test)

Table 16. 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).

Table 16. Translator control block (TRCB) initialization for translate-to-standard (continued)

Field name	Initialization
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 86.
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 397. **Note:** The value set in this field affects the recovery scope used during the session. For more information, see "Send Recovery Scope" on page 86.
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.
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.

Table 16. Translator control block (TRCB) initialization for translate-to-standard (continued)

Field name	Initialization
BATCHID	The batch ID for a group of transactions. The Document 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 Document 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 88. **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 Document 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 Document 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.
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
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 17. 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 18 on page 74.

Table 18. 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 19 must be set by the application on the first call for translation.

Table 19. 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 19. 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 83. **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 86. **Y** All application data has been provided and translation must begin. For more information on this field, see “Translate-to-standard API ” on page 63. For more information on the effects of the EJECT field settings, see “Last call for transaction (TS)” on page 80.
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 Document 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 88. **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 Document 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 (other)** 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 20 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 119.

Table 20. 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 20. 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 21. 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:

`FXXZ`*`ccc`*`(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 22 on page 78.

Table 22. 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 23.

Table 23. 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 133 for a description of the fields that cause a new interchange to be started.

The TRCB fields described in Table 24 on page 79 are returned when an interchange is written. The fields provide information about the interchange, the network, and the file receiving the interchange.

Table 24. 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 132 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 162 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 25 are updated during translation, based on translation type.

Table 25. 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 25. 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 packed 10-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 405.

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 26.

Table 26. 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 151. **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 26. 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 27.

Table 27. 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 151. **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 27. 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 28.

Table 28. 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 152. **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 Document 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 99.
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 29 are updated.

Table 29. 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 376 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 150.

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 186.

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 Document 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 133.

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 134.

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 Document 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 60.

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.

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 170.

Deenveloping data
Enveloping/Deenveloping services (function code **214**). See "Deenvelope API" on page 136

Translating data
Transaction services (function code **213**). See "Translate specific API" on page 95

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 Document Store
- Generates a functional acknowledgement, if one was requested
- Reconciles any received acknowledgments with the original transactions

The Translation service retrieves the transaction image from the Document 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 103.

Processing speed can be improved when you use the DEENVELOPE AND TRANSLATE combination command. This sequence is illustrated in Figure 5 on page 93.

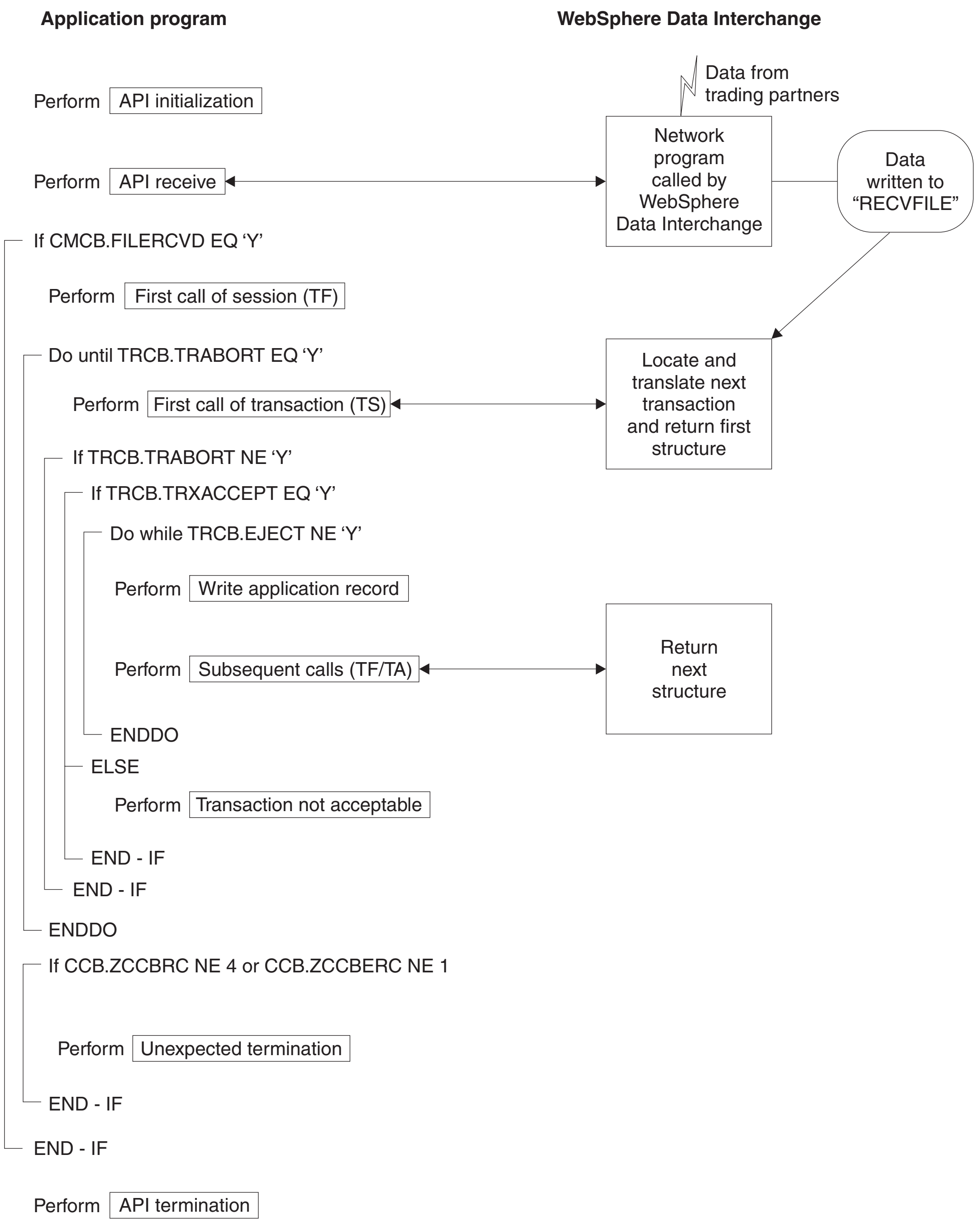

Application program
WebSphere Data Interchange
Data from trading partners
Perform API initialization
Perform API receive
Network program called by WebSphere Data Interchange
Data written to “RECVFILE”
If CMCB.FILERCVD EQ ‘Y’
Perform First call of session (TF)
Do until TRCB.TRABORT EQ ‘Y’
Perform First call of transaction (TS)
Locate and translate next transaction and return first structure
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)
Return next structure
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

Figure 5. 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 Document Store.
- The deenvelope function generates the functional acknowledgement 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 Document 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 Document 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 Document Store, and no functional acknowledgments are generated or reconciled. If you want information written to the Document 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 acknowledgement processing occurs as usual (including updating the Document Store).

- The interchange that contains the functional acknowledgement 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 Document 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 Document Store.

Translate specific API

The basic format of the API request to translate a specific transaction from EDI standard format to application format is:

`FXXZ`*`ccc`*`(SNB,CCB,FCB,TRCB,TRIDB,TRODB)`

The unique parameters for the Translate specific API request are defined in Table 30.

Table 30. 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 376 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 407 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 409 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 Document 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 Document 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

MAPCHAIN
: Indicates whether map chaining 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 Document 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 147.

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 148.

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 31 on page 99 provide some basic attributes of the transaction just processed, and relate to the transaction side of the processing rather than the application side.

Table 31. 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 Document 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 Document 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 116. **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 405.

The TRCB fields described in Table 32 are application-related and provide information about the application side of the processing rather than the transaction side.

Table 32. 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 32. 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 WebSphere MQ, (which are supported in both z/OS and CICS). Valid values are: **MQ** WebSphere MQ 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 116 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 33 are concerned with the interchange for the current transaction.

Table 33. 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**.

Table 33. Interchange header/trailer fields in the translator control block (TRCB) (continued)

Field name	Description
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 on page 99.

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 Document 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 135.

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 Document 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:

`FXXZ`*`ccc`*`(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 376.

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 407 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 409 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 86.

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 376.

Note: The value set in this field affects the recovery scope during the current session. For more information, see "Send Recovery Scope" on page 86.

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 Document 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 Document 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 Document 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 Document 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

MAPCHAIN
Indicates whether map chaining 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 initialize 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 Document 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 147 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 148 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 34 provide information on basic attributes of the transaction. This information is about the transaction side of the processing rather than the application side.

Table 34. 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 Document 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 Document 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 116 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**.

Table 34. TRCB fields providing information on basic transaction attributes (continued)

Field name	Description
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 405.

The TRCB fields described in Table 35 are involved in the application side of the processing rather than the transaction side.

Table 35. 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** WebSphere MQ 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 116 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 acknowledgement. Valid values are:

E	UNB/UNZ
I	ICS/ICE
T	STX/END
U	BG/EG
X	ISA/IEA
G	The acknowledgement does not have an interchange header.
S	The acknowledgement was written to the Document Store, but was not enveloped.

FARC The return code from the translator for the functional acknowledgement translation performed during deenveloping. For more information about return codes, refer to *WebSphere Data Interchange for MultiPlatforms User's Guide*.

FAERC The extended return code from the translator for the functional acknowledgement translation performed during deenveloping. For more information about return codes, refer to *WebSphere Data Interchange for MultiPlatforms User's Guide*.

The TRCB fields described in Table 36 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 36. 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.

Table 36. Interchange header/trailer fields in TRCB (continued)

Field Name	Description
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**.
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 151. **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.

Table 36. Interchange header/trailer fields in TRCB (continued)

Field Name	Description
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.
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 37 are associated with the group to which the current transaction belongs.

Table 37. 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.

Table 37. Group header/trailer fields in the TRCB (continued)

Field name	Description
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.
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 151. **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 38 are associated with the transaction header and trailer for the current transaction.

Table 38. 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 152. **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.

Table 38. Transaction header/trailer fields in the TRCB (continued)

Field name	Description
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.
TTC	The current transaction or message ID value. See also `TRNID` on 99.
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 Document 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 148. 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 Document 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 Document Store database.
- A deenveloping function that locates interchanges in a file, parses the interchange into groups and transactions, and adds the transactions to the Document Store. The deenveloping function also adds the EDIVTSEV, EDIVTSGP and EDIVTSTU records to the Document 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 Document 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 63 and “Translate-to-standard API” on page 69.

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 133 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 133 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 Document 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 39.

Table 39. 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 381.
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 408.
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 410.

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 40. 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 381. **Note:** This value affects the recovery scope during the session.

Table 40. 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 8, “Exit routines,” on page 327.
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 131 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.
RAWDATAOUT	Indicates whether RAWDATA output is used for fixed-to-fixed mappings. **Y** Uses RAWDATA output **(other)** Uses C & D record output

Table 40. 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 envelope 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 Document 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 41 provide some basic attributes of the transaction just processed. These fields are related to the transaction side of the processing.

Table 41. 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 376.

Application-related fields (EV): The TRCB fields described in Table 42 are related to the application side of the processing.

Table 42. 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 42. 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 map chaining 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 43 are related to the interchange header or trailer.

Table 43. 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 151. **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 43. 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 44 are related to the group header or trailer.

Table 44. 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 151. **Y** Started a new group **(other)** Did not start a new group

Table 44. 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 45 are related to the transaction header or trailer

Table 45. 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 152. **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 Document Stored as a 4-byte binary value. See `TST` for the character representation of this value.

Table 45. TRCB fields related to transaction header or trailer (continued)

Field name	Description
TTC	The current transaction or message ID value. See also `TRNID` on 99.
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 46 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 46. 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 132 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 135.

Envelope processing considerations (EV)

The following sections list special considerations for envelope processing.

Envelope versus reenvelope

Both the WebSphere Data Interchange Utility and Document Store Facility provide a means of enveloping and reenveloping transactions. When the ENVELOPE function is used, only transactions in the Document 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 Document 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 75.

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 162) 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 47.

Table 47. 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 381 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 410 for a general description of this block.

Queueing - TRCB Fields:

The TRCB fields described in Table 48 on page 135 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 48. 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 132 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 49.

Table 49. 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 410 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 134 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 Document 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 91 and “Translate-file-to-application API” on page 103.

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 Document 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 50.

Table 50. 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 376.
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 407 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 409 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," 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 51.

Table 51. 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 86.

Table 51. 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 376. **Note:** This field contains values that affect the recovery scope during the session. For more information, see "Send Recovery Scope" on page 86.
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 8, "Exit routines," on page 327.
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 51. 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 Document 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 Document 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 Document 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.
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 147.

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 acknowledgement 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 148. 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 52 are related to the transaction side of the processing rather than the application side.

Table 52. 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 Document 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 Document 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 405 for a list of errors.

The TRCB fields described in Table 53 on page 143 are related to the application side of the processing rather than the transaction side.

Table 53. 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 54 are related to the functional acknowledgments.

Table 54. Fields related to functional acknowledgements

Field name	Description
FABUILT	The envelope type for the functional acknowledgement. Valid values are: **E** UNB/UNZ **I** ICS/ICE **T** STX/END **U** BG/EG **X** ISA/IEA **G** The acknowledgement does not have an interchange header. **S** The acknowledgement was written to the Document Store, but was not enveloped.
FARC	The return code from the translator for the functional acknowledgement translation done during deenvelope. For more information, refer to *WebSphere Data Interchange for MultiPlatforms User's Guide*.
FAERC	The extended return code from the translator for the functional acknowledgement translation done during deenvelope. For more information, refer to *WebSphere Data Interchange for MultiPlatforms User's Guide*.

The TRCB service segment fields (TF) described in Table 55 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 55. 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 55. 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 56 on page 145 change as transactions are processed to indicate how much of the interchange has been processed.

Table 56. 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 151. **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 57, Table 58 on page 146, and Table 59 on page 146 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 57. 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 57. 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 58 change as transactions are processed to indicate how much of the group has been processed.

Table 58. 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 151. **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 59 are associated with the transaction header and trailer for the current transaction.

Table 59. 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 152. **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 Document 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 on page 99.
TVER	The transaction version (transaction header for data type **VR**).
TREL	The transaction release (transaction header for data type **LV**).

Table 59. 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 60. 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 61. 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 186.

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 62.

Table 62. 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 63.

Table 63. 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 64.

Table 64. Parameters for the Retrieve group header API

Parameter	Description
SNB	The same SNB used in previous requests. No further initialization is required.

Table 64. Parameters 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:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

The unique parameters for this API request are defined in Table 65.

Table 65. 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 66.

Table 66. 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 acknowledgement image **219** Functional acknowledgement image
TRCB	The translator control block. See "Translator Control Block (TRCB)" on page 376 for details on this control block.
TRIDB	The translator input data block. See "Translator Input Data Block (TRIDB)" on page 407 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 409.

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 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 Document Store and returns it as formatted records in the TRODB. For more information, see *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

The TRCB fields described in Table 67 on page 155 are required to initialize transactions.

Table 67. 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 Document Store and formatted into the data extract record formats. For more information, see *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00. 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 acknowledgement interchange record
   - Functional acknowledgement group record
   - Functional acknowledgement transaction record
3. Transaction record, which can be followed by:
   - Transaction acknowledgement interchange record
   - Transaction acknowledgement group record
   - Transaction acknowledgement 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 acknowledgement for the group, the interchange, group, and transaction records for the functional acknowledgement are returned with the group record even if concatenation has not been requested.

If there is a transaction acknowledgement for the transaction, the interchange, group, and transaction records for the transaction acknowledgement 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 68 are returned in response to the first request for detailed information about a transaction.

Table 68. 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

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:

```
FXXZccc(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 acknowledgement image API

This API function allows your application to retrieve the transaction acknowledgement 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:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

A transaction acknowledgement 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 acknowledgement image API

This API function allows your application to retrieve the functional acknowledgement image for the current group returned from the Retrieve detailed data API. You must request the current group (function code **216)** before requesting an acknowledgement image (function code **219)**. Once the group detail record is returned, make the following API request:

```
FXXZccc(SNB,CCB,FCB,TRCB,TRIDB,TRODB)
```

A functional acknowledgement 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 121 and “Translate-to-standard API ” on page 63.
- Using the ENVELOPE, REENVELOPE, or combination commands provided by the WebSphere Data Interchange Utility.
- Using the enveloping and reenveloping functions provided by the Document 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 141 and “Translate-file-to-application API” on page 103.
- Using the DEENVELOPE or combination commands provided by the WebSphere Data Interchange Utility.
- Using the deenvelope functions provided by the Document 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 69 contains an overview of the functions provided by the communications service. The logical name for the communications service is COMM.

Table 69. Overview of functions provided by Communication services

Function	**Code**	**Sample Call Statement for Function**
Queue standard data	**110**	`FXXZ`*`ccc`*`(SNB,CCB,FCB,CMCB,TPPDB,DATABLK)`
Send transactions	**211**	`FXXZ`*`ccc`*`(SNB,CCB,FCB,CMCB,TPPDB)`
Send files	**221**	`FXXZ`*`ccc`*`(SNB,CCB,FCB,CMCB,TPPDB)`
Receive	**232**	`FXXZ`*`ccc`*`(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 70.

Table 70. 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 70. 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 71. 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 71. 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 GXS 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 179). 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 72.

Table 72. 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 412.
TPPDB	The trading partner profile data block. For more information, see "Trading partner profile data block (TPPDB)" on page 160.

CMCB initialization for sending transaction data

Before issuing an send API request, you must initialize the CMCB fields described in Table 73.

Table 73. 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 73. 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 165.
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 73. 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 acknowledgement wanted. See the definition of this field in "Communication Control Block (CMCB)" on page 412 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 167 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 167 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 74 are returned on a request to send transactions.

Table 74. 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 acknowledgement` field of the mailbox (requestor) profile or the `Net acknowledgement` 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 74. 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:

```
FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)
```

The TPPDB is described in Table 70 on page 160. The CMCB settings for this request are described in Table 75. 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 75 describes the fields and the initialization requirements.

Table 75. 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 75. 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 acknowledgement wanted. See “Communication Control Block (CMCB)” on page 412 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 76.

Table 76. 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 acknowledgement` 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 76. 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 GXS 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 GXS 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:

`FXXZ`*`ccc`*`(SNB,CCB,FCB,CMCB,TPPDB)`

The TPPDB is described in Table 70 on page 160. The CMCB settings for this request are described in “CMCB initialization for receive requests” on page 171. 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 77 describes the fields and initialization requirements.

Table 77. 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 172.
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 77. 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 78.

Table 78. 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 78. 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 70 on page 160. 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 79 describes the fields and initialization requirements.

Table 79. 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 acknowledgement wanted regarding the cancellation. See “Communication Control Block (CMCB)” on page 412 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 79. 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 80are 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 80. 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:

```
FXXZccc(SNB,CCB,FCB,CMCB,TPPDB)
```

The TPPDB is described in Table 70 on page 160. The CMCB settings for this request are described in “CMCB initialization for Return filename request” on page 176. 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 81 describes the fields and initialization requirements.

Table 81. 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 82.

Table 82. 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 132

132. 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 103 and “Envelope API” on page 121.

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 acknowledgement 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 70 on page 160. 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 83.

Table 83. 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 acknowledgement data.

Table 83. 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 184 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

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 84.

Table 84. 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 181.

Table 84. 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 184).
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 85.

Table 85. 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 86 on page 182 describes the format for a full key using a trading partner nickname. Use this update status key block when `ZFCBFUNC` contains **215**.

Table 86. 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 87 describes the format for a full key using an interchange qualifier and ID. Use this update status key block when ZFCBFUNC contains **213**.

Table 87. 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 88 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 88. 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 89. 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 184) 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 90 uses the account number and user ID. Use this update status key block when `ZFCBFUNC` contains **212**.

Table 90. 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 90. 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 91 uses the interchange qualifier and ID. Use this update status key block when `ZFCBFUNC` contains **214**.

Table 91. 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 92 describes the fields used to build an alternate key that can be used later to retrieve and update the status of this interchange.

Table 92. 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 Document 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 acknowledgement expected for this interchange. See "Trading Partner Profile Block (TPPDB)" on page 423 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 86). 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 Document Store is compromised.

Table 93 lists the functions provided by the SYNCPOINT service. The logical name for the SYNCPOINT service is SYNCSERV.

Table 93. 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 86.

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 94 describes the DB2 recovery conditions and actions.

Table 94. 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 94. 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 95.

Table 95. 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 fifth (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:

```
FXXZccc(SNB,CCB,FCB)
```

The unique parameters for the COMMIT work function request are defined in Table 96.

Table 96. 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:

```
FXXZccc(SNB,CCB,FCB)
```

The unique parameters for the ROLLBACK work function request are defined in Table 97.

Table 97. Parameters for the ROLLBACK work function request

Parameter	Description
SNB	**ZSNBNAME** SYNCSERV **ZSNBPC** 3

Table 97. 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 482 and “Get/Put envelope exit and service” on page 340.

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 98.

Table 98. 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 482 and “Get/Put envelope exit and service” on page 340.

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 99.

Table 99. 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.

Chapter 3. 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 5, "Interfacing to other networks and applications," on page 275.

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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

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 207.
- 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 229.
- 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

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 229 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 216 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 216.

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 229 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 WebSphere MQ 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 199.

  – 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.

- WebSphere MQ queues

  WebSphere MQ queues are specified using WebSphere Data Interchange WebSphere MQ Queue profile member names and a `File type` of **MQ**. WebSphere MQ 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. WebSphere MQ allows cross-platform communications between applications. In order to use WebSphere Data Interchange WebSphere MQ support, you must have WebSphere MQ 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 482 and “Put Envelope service example” on page 482.

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 482 and “Outbound envelope program example” on page 483.

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 100. 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 216.

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 GXS Expedite/CICS are not deleted by WebSphere Data Interchange or by GXS 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.

Document 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

- 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 WebSphere MQ 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 216.

Table 101 lists all WebSphere Data Interchange Utility commands and the point at which a CICS SYNCPOINT command can be issued.

Table 101. Unit of work for each 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 101. Unit of work for each 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 101. Unit of work for each 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, Document 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 Document 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 Document 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 216.

The following guidelines are suggested if the WebSphere Data Interchange Utility is to be part of your application's unit of work:

- 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 Document 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 229.
- Use the CICS SYNCPOINT command in continuous receive response programs and utility termination response programs. For more information, see “Response applications” on page 229.

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 ten translations or other GXS 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 GXS Expedite/CICS transactions and can be
*  removed if GXS Expedite/CICS is not used.
*-----------------------------------------------------------------*
```

```
DSNCRCT TYPE=ENTRY,AUTH=(SIGNID,*,*),THRDM=10,THRDA=10, X
    TXID=(EDIB,EDID,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, 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 Document 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 GXS 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)
```

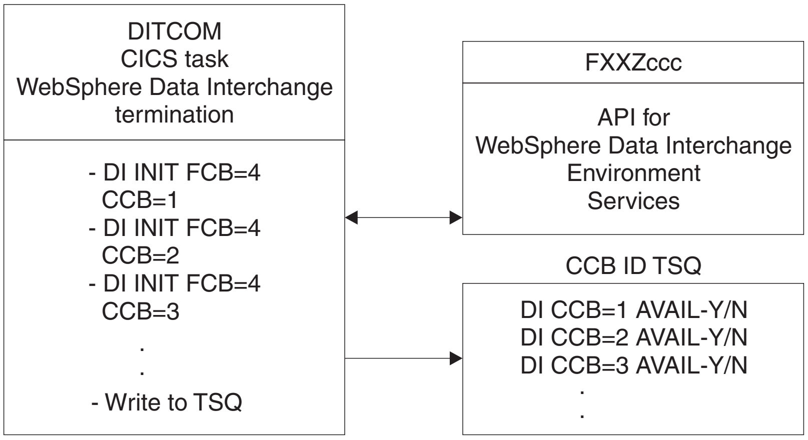


Figure 6. HOT-DI initialization diagram

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 55.

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 537.

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**.

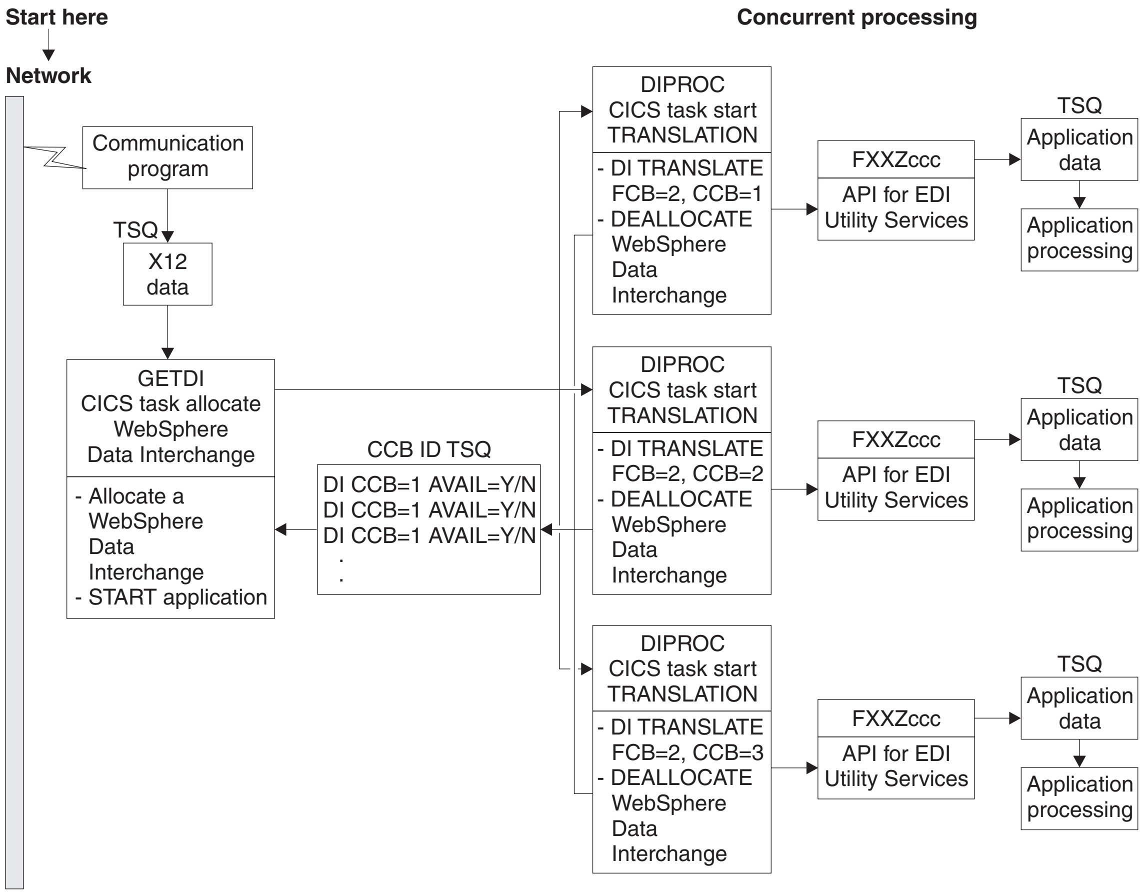

Start here
Network
Concurrent processing
Communication program
TSQ
X12 data
GETDI
CICS task allocate WebSphere Data Interchange
- Allocate a WebSphere Data Interchange
- START application
CCB ID TSQ
DI CCB=1 AVAIL=Y/N
DI CCB=1 AVAIL=Y/N
DI CCB=1 AVAIL=Y/N
DIPROC
CICS task start TRANSLATION
- DI TRANSLATE FCB=2, CCB=1
- DEALLOCATE WebSphere Data Interchange
- DI TRANSLATE FCB=2, CCB=2
- DI TRANSLATE FCB=2, CCB=3
FXXZccc
API for EDI Utility Services
TSQ
Application data
Application processing

Figure 7. 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 57.

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 216.

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 58.

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)
```

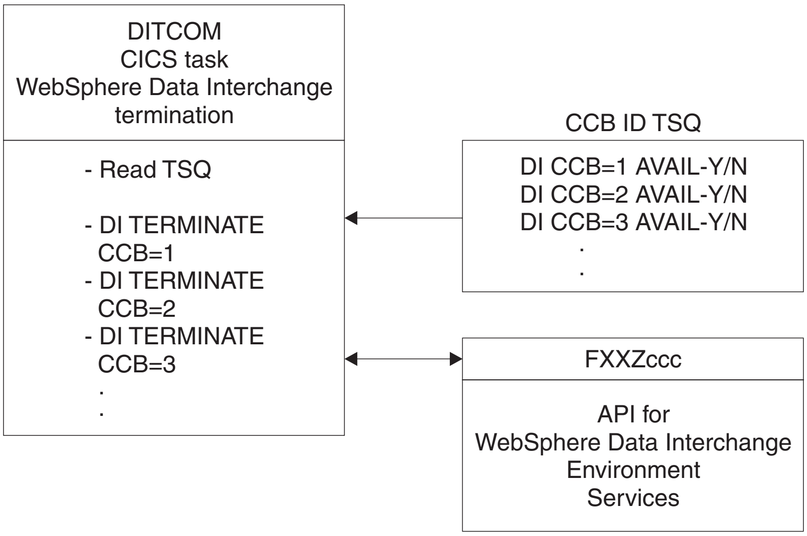


Figure 8. HOT-DI termination diagram

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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00 or "Communication services" on page 158.

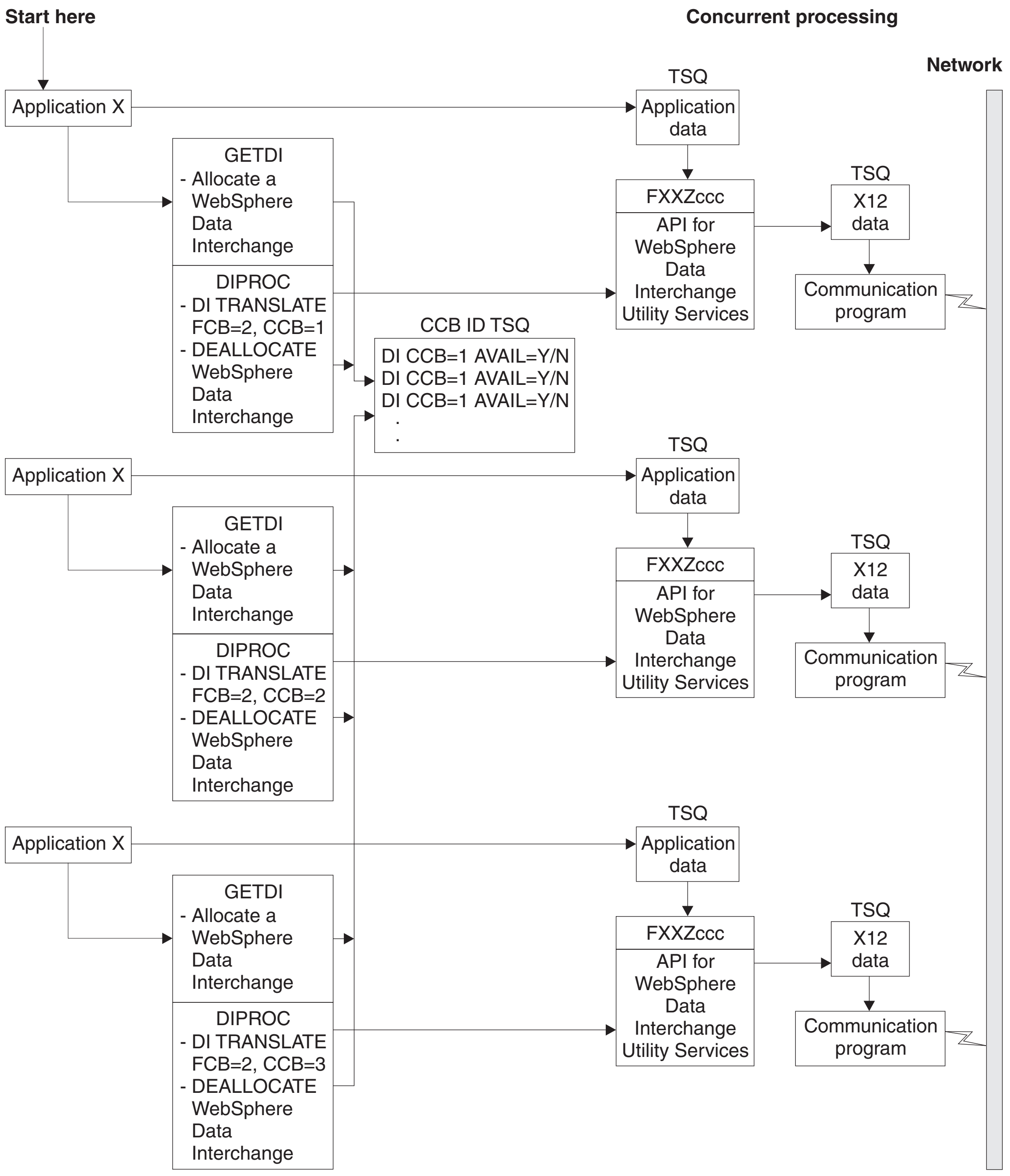

Start here
Concurrent processing
Network
Application X
TSQ
Application data
GETDI
- Allocate a WebSphere Data Interchange
DIPROC
- DI TRANSLATE FCB=2, CCB=1
- DEALLOCATE WebSphere Data Interchange
FXXZccc
API for WebSphere Data Interchange Utility Services
TSQ
X12 data
Communication program
CCB ID TSQ
DI CCB=1 AVAIL=Y/N
DI CCB=1 AVAIL=Y/N
DI CCB=1 AVAIL=Y/N
Application X
TSQ
Application data
GETDI
- Allocate a WebSphere Data Interchange
DIPROC
- DI TRANSLATE FCB=2, CCB=2
- DEALLOCATE WebSphere Data Interchange
FXXZccc
API for WebSphere Data Interchange Utility Services
TSQ
X12 data
Communication program
Application X
TSQ
Application data
GETDI
- Allocate a WebSphere Data Interchange
DIPROC
- DI TRANSLATE FCB=2, CCB=3
- DEALLOCATE WebSphere Data Interchange
FXXZccc
API for WebSphere Data Interchange Utility Services
TSQ
X12 data
Communication program

Figure 9. 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 102. 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 102. 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	N	CICS system ID override
RESPFLAG	Char	125	1	Y	Response indicator
FILETYP	Char	126	2	Y	Network acknowledgement 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 acknowledgement 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 acknowledgement return code
FAERC	Bin	188	4	Y	Functional acknowledgement extended return code
FABUILT	Char	192	1	Y	Functional acknowledgement 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	N	User condition code pointer
APMTCBP	Bin	200	4		Application multi-TSQ pointer
RES1	Char	204	10	N	Reserved for future use
FFNOCNV	Char	214	1		No return code conversion flag
FANAME	Char	215	8		Functional acknowledgement file name
RESPACTV	Char	223	1	N	Response program active flag

Table 102. Format of WebSphere Data Interchange utility control information (continued)

Name	Type	Offset	Length	Result	Description
WORKNAME	Char	224	8	N	Name of work file
BATFLG	Char	232	1		Batch or UTILSRV API (HOTDI) flag
NOEXCP	Char	233	1	N	No exception file
INVPARM	Char	234	1	N	Network program invalid parameter
LOGACTV	Char	235	1	N	Logging CE0050 active
FFUSADDR	Bin	236	4	N	Pointer to this block
CCBP	Bin	240	4	N	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 202. 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. This field has a 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00. 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00. 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00. 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00. 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00. 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 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 229 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 229 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 acknowledgement, the `ENVFILE` field contains the name of the TS queue holding the network acknowledgement.

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 acknowledgement 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 *WebSphere Data Interchange for MultiPlatforms User's Guide*.

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 acknowledgement processed. If the TS queue contains a network acknowledgement, 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 acknowledgement. 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 Document 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 GXS Expedite/CICS during a continuous receive. This field is only passed when the utility is invoked due to a continuous receive. 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 acknowledgement processing.

FAERC

The highest extended return code encountered during functional acknowledgement 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 199.

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 199.

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 199.

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 199.

Continuous receive considerations

The continuous receive Facility is a WebSphere Data Interchange service that works in conjunction with GXS Expedite/CICS and Information Exchange. It also can used with WebSphere MQ 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

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 Document Store is kept up to date without initiating the UPDATE STATUS command at different points during the session.

Continuous receive using WebSphere MQ

You can use WebSphere MQ to trigger continuous receive processing. Like sending to an Information Exchange mailbox, you can have your trading partners deliver envelopes to WebSphere MQ queues. To use WebSphere MQ for continuous receives, do the following:

1. Your CICS and WebSphere MQ administrators must enable the CICS region for WebSphere MQ support. As part of this process, they should define a WebSphere MQ trigger event queue, monitored by the WebSphere MQ transaction CKTI in the region. In this example, the name of this queue is **CICS1.TRIGGER**.
2. Using the DEFINE PROCESS command, your WebSphere MQ administrator must define one WebSphere MQ process to provide WebSphere MQ 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 WebSphere MQ administrator must define at least one WebSphere MQ 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 WebSphere MQ profile member to this specific WebSphere MQ queue. The order in which you specify the two keyword-value combinations is not important.

   In this example, only one WebSphere MQ queue is used for event-driven EDI but you can define multiple queues. As long as the WebSphere Data Interchange and WebSphere MQ information is set up correctly, WebSphere Data Interchange will process any number of queues.
4. In WebSphere Data Interchange, you must define a WebSphere MQ 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 227 to tell WebSphere Data Interchange what processing is required once data is received from WebSphere MQ.

When the above steps have been completed successfully, WebSphere Data Interchange automatically processes all data written to the WebSphere MQ queue as soon as WebSphere MQ 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 WebSphere MQ 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 a WebSphere MQ trigger event has occurred. So, if you are using WebSphere MQ for continuous receive processing, skip to the next section, "WebSphere Data Interchange processing after data is received" on page 227.

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 103 defines the settings in the profile fields that combine to make the selection criteria unique.

Table 103. 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.

Table 103. Continuous receive selection criteria (continued)

Profile field	Description
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 GXS 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 104.

Table 104. 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 acknowledgement 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 acknowledgement. If a response program is supplied in the continuous receive profile, it is given control and the name of the TS queue containing the acknowledgement 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 GXS 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 GXS 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 GXS Expedite/CICS are processed by WebSphere Data Interchange. If EDI1 is not defined, there is 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. By defining EDI1 as non-recoverable, you eliminate the possibility of lost interchanges, and GXS 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 GXS 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 GXS 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 GXS Expedite/CICS typically send interchanges asynchronously. When WebSphere Data Interchange requests a send, GXS 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, GXS Expedite/CICS (transaction IST1) will LINK back to WebSphere Data Interchange to update the status of the transaction appropriately.

WebSphere Data Interchange and GXS Expedite/CICS can send interchanges synchronously. The GXS 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 Document 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 acknowledgement processing.

Note: WebSphere Data Interchange supports compress and delivery priority for GXS Expedite/CICS.

Using continuous receive outside GXS Expedite/CICS

Thus far, the continuous receive facility has been described as it works with GXS 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 293 for details.

You can process multiple incoming TS queues when continuous receive is used without GXS Expedite/CICS. For more information, see “Processing multiple incoming TS queues” on page 199.

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 216 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 220.

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 216. 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 216 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 232.

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 216.

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 232.

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 acknowledgement and updated the status in the Document Store. The name of the TS queue containing the acknowledgement 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 acknowledgement itself. The TS queue holding the network acknowledgement is deleted before termination. If you have no reason to save the network acknowledgement, 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)" 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 PERFORM 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 Document 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 216.

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 Document 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 *WebSphere Data Interchange for MultiPlatforms User's Guide*. 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.

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, see the *WebSphere 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 the *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

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 228.

EDI2

If TD queue EDI2 is defined as an intrapartition TD queue, WebSphere Data Interchange will use this queue as the GXS Expedite/CICS administrative response file.

GXS 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 GXS 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, GXS Expedite/CICS passes WebSphere Data Interchange one network acknowledgement 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, GXS Expedite/CICS, and Information Exchange

WebSphere Data Interchange provides communication routines that interface to GXS 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 acknowledgement 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. GXS 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 GXS Expedite Base/MVS, this means that you cannot have concurrent Information Exchange sessions for the same mailbox on two different GXS 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 GXS 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 GXS Expedite/CICS. GXS 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 245.

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 GXS 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, GXS 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 241 for details.

Attention: Expedite/CICS provides a facility to start and end Information Exchange sessions directly, but you should not use GXS 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 GXS Expedite/CICS functions. The GXS 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 GXS 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, GXS 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 GXS Expedite acknowledges that there is an active Information Exchange session. You can also determine if GXS 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 GXS Expedite/CICS and WebSphere Data Interchange. If you confirm the request to end the session, GXS 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 241.

When GXS 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 GXS 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 GXS Expedite/CICS Display Application User's Guide.

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 GXS Expedite/CICS. It causes an out-of-sync condition between the components.

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 GXS 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 GXS Expedite/CICS IDLT CICS terminal transaction to delete the Information Exchange session control information from the GXS 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 GXS Expedite/CICS and passes the EDI envelope. GXS 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 GXS Expedite/CICS. GXS Expedite/CICS is then free to process the next envelope for this continuous receive session.

Exactly when WebSphere Data Interchange returns control to GXS Expedite/CICS depends on whether the TD queue EDI1 is defined as intrapartition. If EDI1 is defined, WebSphere Data Interchange and GXS Expedite run synchronously. In other words, GXS Expedite/CICS will not process the next envelope until WebSphere Data Interchange returns control. If the EDI1 TD queue is not defined, GXS Expedite/CICS and WebSphere Data Interchange run asynchronously where GXS 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 228. 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

Note: This information applies only to continuous receive processing associated with GXS 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 293.

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 GXS Expedite/CICS, passing it the continuous receive selection criteria. For criteria options, see "Continuous receive selection criteria" on page 226. GXS 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, GXS 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 GXS Expedite/CICS passing the key for the particular continuous receive session. GXS 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. GXS Expedite/CICS returns control to WebSphere Data Interchange where WebSphere Data Interchange and GXS Expedite/CICS can complete asynchronously. However, WebSphere Data Interchange checks GXS 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 GXS Expedite/CICS provides a facility to start and stop continuous receive sessions directly, using these GXS 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 GXS Expedite/CICS. The GXS 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00. 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 GXS Expedite/CICS or Information Exchange. Therefore, it is essential that GXS Expedite/CICS (using LG01) be used to terminate all continuous receive sessions before executing EDIZ. If GXS 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 GXS 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 105 describes some methods to help you to identify an unrecoverable continuous receive session.

Table 105. 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 GXS 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 GXS 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. GXS 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 GXS Expedite/CICS IDLT CICS terminal transaction to delete the Information Exchange session control information from the GXS Expedite/CICS control file. The format of the IDLT transaction is:

   ```
   IDLTacct [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 GXS 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 GXS 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 GXS 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             *
* GXS Expedite/CICS PLT program EXP0STRT.  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 entries.            *
*-------------------------------------------------------------------*
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 GXS 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 106. 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.
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.
EDIM	A transaction that executes the WebSphere Data Interchange Message Broker to transform data.
EDIQ	The transaction that gains control from WebSphere MQ transaction CKTI when data is received into a WebSphere MQ 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.

Table 106. WebSphere Data Interchange supplied CICS transactions (continued)

Transaction	Description
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 EDIX, 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.
EDIW	The WebSphere Data Interchange Utility invocation transaction. For more information, see "Using EDIW to invoke the WebSphere Data Interchange Utility" on page 251.
EDIX	A background transaction used to perform some Document 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.
EDI7	Program name EDIJSTRT. This starts one or more XML parser long-running transactions (EDIJ) and a long-running monitor transaction (EDI9). These transactions must be started in order to do XML processing. The number of parser transactions to be started is controlled by the parser.number property in the EDIParser.properties file.
EDI8	Program name EDIJSTOP. This stops all active XML parser transactions (EDIJ) and the monitor transaction (EDI9).
EDI9	Program name EDIJMNTR. This is a monitor program for the long-running XML parser transactions. If a parser ends abnormally, the monitor will restart it. This transaction is normally started by transaction EDI7.

Table 106. WebSphere Data Interchange supplied CICS transactions (continued)

Transaction	Description
EDIJ	Program name EDIJPXML. This is the long-running XML parser transaction. It waits for data to be put on a queue by another WebSphere Data Interchange component, parses the data, and passes the results on another queue. One or more of these transactions are normally started by transaction EDI7.

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 GXS Expedite/CICS. Typically, WebSphere Data Interchange and GXS 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 GXS 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 250 describes which fields are filled in by WebSphere Data Interchange and which fields are filled in by GXS Expedite/CICS.

On the send side, WebSphere Data Interchange could invoke the user exit during enveloping and, subsequently, GXS 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, GXS 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 107. 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
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 216.

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
```

Chapter 4. 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(ediec33e);
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

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 a WebSphere MQ queue

The WebSphere Data Interchange adapter program is installed as part of WebSphere Data Interchange for Multiplatforms V3.3. The configuration scripts provided set up the necessary queues and definition objects. The adapter uses WebSphere MQ Triggering to know when messages need processing.

When a message is put to an application queue, a trigger message is created. The WebSphere MQ 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 WebSphere MQ support architecture uses six WebSphere MQ 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 263.

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)/`WebSphere MQ 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 263, 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 WebSphere MQ. 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 a WebSphere MQ 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 WebSphere MQ through the use of an adapter program which can listen for trigger message generated by WebSphere MQ 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 WebSphere MQ. 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 a WebSphere MQ 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).

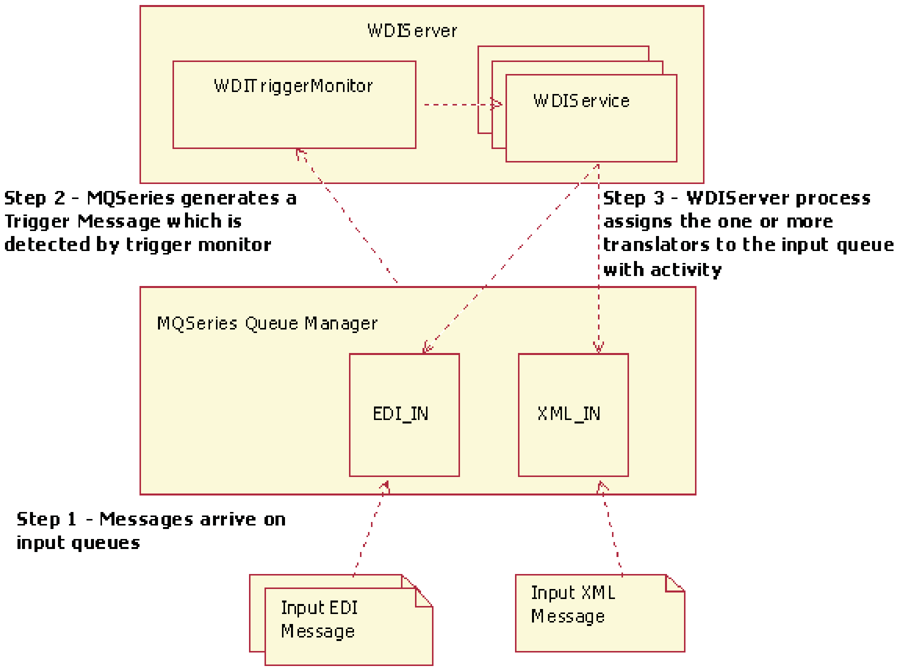

WDIServer
WDITriggerMonitor
WDIService
Step 2 - MQSeries generates a Trigger Message which is detected by trigger monitor
Step 3 - WDIServer process assigns the one or more translators to the input queue with activity
MQSeries Queue Manager
EDI_IN
XML_IN
Step 1 - Messages arrive on input queues
Input EDI Message
Input XML Message

Figure 10. Overview of advanced adapter

The diagram in Figure 10 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 WebSphere Data Interchange 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.

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 WebSphere Data Interchange 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 (include the periods after the equal sign):

```
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 WebSphere MQ. 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 WebSphere Data Interchange 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 and output. 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 EDIEC33E which is the default plan name created when WebSphere Data Interchange is installed. Only change this value if you have created an alias to the database or are using more than one WebSphere Data Interchange 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 WebSphere Data Interchange software has been installed and configured the next step would be to setup the input queues which will be feeding data to WebSphere Data Interchange. When the WebSphere Data Interchange software is installed a script called AdvAdapterMQ.txt which contains WebSphere MQ commands to create several new WebSphere MQ 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 through the Trigger Data field. These four options are:

1. NumThreads()

   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()

   Specifies 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()

   Specifies 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 WebSphere Data Interchange is triggered.
5. Convert

Indicates that WebSphere Data Interchange should convert the message code page as it is received. Valid values are Y for yes and N for no. The default is Y to convert all incoming data.

6. 6. CCSID

   Indicates the Coded Character Set ID used when converting data read from this queue. Valid values are determined by Websphere MQ, and the default value is zero to use the CCSID defined for the local Queue Manager.

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 WebSphere Data Interchange, WebSphere MQ 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 subdivided 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)

  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 WebSphere Data Interchange bin directory. The installation default values for Windows are listed below. The default values for AIX are similar.

```
runtimedirectory=C:\WDIServer33\bin
datadirectory=C:\WDIServer33\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=EDIEC33E
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 host name 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 instantiated 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. 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 host name 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 271 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/DIv33/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

```
#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("EDIEC33E");
   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;

   // 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);
}
```

Chapter 5. 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 158. 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 2, “Using WebSphere Data Interchange in the z/OS environment,” on page 23.

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 11 on page 279 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 288.
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 228.
- VANICICS is for CICS only, fills in several predefined 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 289.
- PTTOPT identifies the point-to-point network. For more information, see “Point-to-point networks” on page 282.
- VANIMQ is for the exclusive use of WebSphere MQ 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 412.
- 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 284. 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.

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 288.

Application-to-network flow diagram

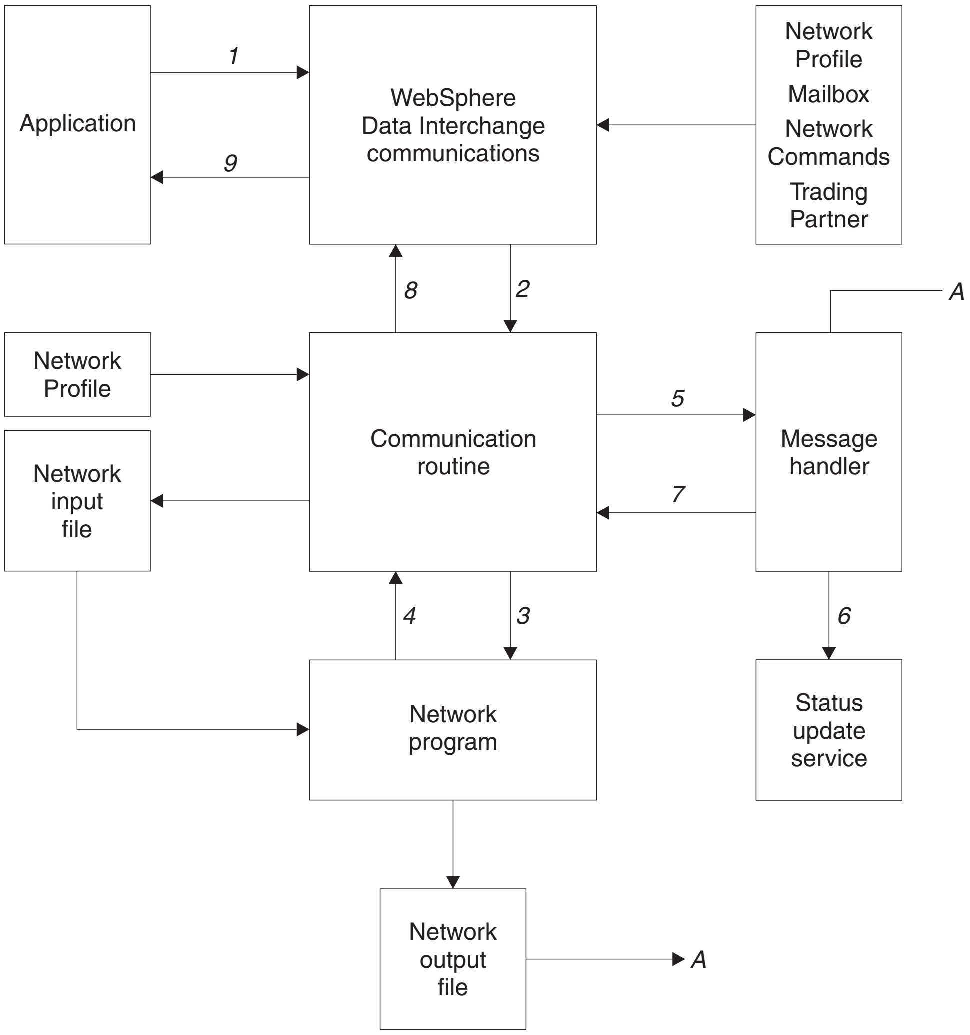


Figure 11. Application-to-network flow diagram

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 158. 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 acknowledgement 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 Document 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 GXS Expedite network program and when GXS 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 GXS 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 288.

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 Document Store. For more information, see "Update status services" on page 179.
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 283.

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 287.

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 279) 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 8, “Exit routines,” on page 327 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 158.

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 365.

- Service Name Block (SNB)
- Common Control Block (CCB)
- Function Control Block (FCB)

- 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 108 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 108. 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:

- 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 109 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 109. 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 110 describes how to use the FSUPPORT bytes.

Table 110. 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 8, “Exit routines,” on page 327

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 Document Store, if status update messages are contained in network output file. For more information, see “Update status services” on page 179.
  - 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 GXS 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 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 Document 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 111.

Table 111. 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 112.

Table 112. 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 113.

Table 113. 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 Document Store. You do this by using the update envelope status API documented in "Update status services" on page 179. 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 114 on page 293.

Table 114. 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 GXS 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.

There are three major differences between continuous receive processing which interacts with GXS Expedite/CICS and processing that interacts with user-written receive applications as follows:

- Only continuous receives associated with GXS 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 GXS Expedite/CICS. The continuous receive profile member ID must be a maximum of 8 characters long. Continuous receive profile member IDs associated with GXS 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 115.

Table 115. 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 199.

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 218.

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.

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 source document property THANDLE for Data Transformation maps are provided to enable mapping of the WebSphere Data Interchange archive key to the SAP IDOC for inbound processing. The special WebSphere Data Interchange variable name DISAPSEQ is not supported for data transformation processing. WebSphere Data Interchange 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 Document Store is active, functional acknowledgement 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, acknowledgement 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 send and receive maps, 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 WebSphere MQ

You can exchange data with your trading partners using WebSphere MQ queues in network profiles. WebSphere Data Interchange prohibits the use of WebSphere MQ queues as the target of the envelope process or as input to the deenvelope process. WebSphere MQ queues can be used as part of a logical network where enveloped data is sent to and received from WebSphere MQ 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 WebSphere MQ Administrator to define the WebSphere MQ queue you plan on using.
2. Define WebSphere Data Interchange WebSphere MQ Queue profile members for the queues your WebSphere MQ 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 WebSphere MQ 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 WebSphere MQ queue profile member names created in Step 2 above. The **SENDMQ** keyword should be followed by the WebSphere Data Interchange WebSphere MQ queue profile member name where you send data. The **RECEIVEMQ** keyword should be followed by the WebSphere Data Interchange WebSphere MQ 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 365.
5. If you are using continuous receive, you also perform the following tasks:
   - When using WebSphere MQ queues with an application file, you must identify the associated WebSphere MQ 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 WebSphere MQ queues (for print, report, exception, and so on).
   - When using a WebSphere MQ queue for your application data file, you must enter the WebSphere MQ 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

   For more information, see “Continuous receive using WebSphere MQ” on page 225.

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 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 encoding, 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 code page and will treat any newline characters as a carriage-return character (x’0A’), which is a valid XML white space character.

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 created as EBCDIC unless overridden with the Encodetarget mapping property. 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 6. Data Transformation Document Store implementation

Data Management and reporting for Data Transformation processing uses the existing Document Store interfaces. The WebSphere Data Interchange Utility interface is supported for all platforms and includes the primary Document Store functions.

The primary Document 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 Document Store interface (WebSphere Data Interchange Client and Utility PERFORM commands) will be used to report the status of transactions in the Document Store. Documentation on WebSphere Data Interchange Document Store may be found in the *WebSphere Data Interchange for MultiPlatforms User's Guide*, SC34-6215-01.
- 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 Document Store

The WebSphere Data Interchange 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 Document Store database.

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 WebSphere Data Interchange 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 WebSphere Data Interchange C++ API.

There are three settings in the application defaults profile that control Document Store activity:

- The Document Store radio buttons control whether information is saved to the document store. You can choose to:
  - Always save information to the Document Store, but only for EDI transactions

  - Always save information to the Document Store, including information about Data Format and XML documents. Information about XML and Data Format documents is only saved to the Document Store for Data Transformation processing.
  - Only save EDI information for transactions that are successfully translated
  - Only save EDI information for transactions that have errors
  - Do not save any information to the Document Store. If this value is set, the Document Store is inactive, and the image flags are ignored.
- The Document Image setting controls whether inbound and outbound transaction images are to be written to the Document Store. This includes EDI transactions, XML and Data Format Documents, and inbound functional acknowledgement images.
- The Functional Acknowledgement Image setting controls whether outbound functional acknowledgement images are to be written to the store. This flag does not control Document Store activity for inbound functional acknowledgement images. Inbound functional acknowledgement images are controlled by the Transaction Image flag.

Chapter 7. Common Events Handler

The common events handler:

- Enables you to improve the performance of the WebSphere Data Interchange Event Log and Print File. See “Improving performance by using the Common Events Handler” for more information.
- Enables you to develop custom Java plug-in programs to process WebSphere Data Interchange generated Events. See “Using the Common Events Handler Plug In” on page 320 for more information.

Improving performance by using the Common Events Handler

The Common Events Handler adds the following enhancements to WebSphere Data Interchange:

- An XML version of the Print File. This allows applications to easily parse the error messages and generate appropriate alerts or other notifications. For samples of the Print File and the XML Print File, see “Samples print files” on page 317.
- An ADF (fixed record format) version of the Print File.
- The ability to route print files to an MQ queue, file, directory, or CICS transient data or temporary storage queue for post-WebSphere Data Interchange processing. By creating Event Destination profiles, you can route print files to different destinations based on certain criteria, such as the maximum severity level or whether a particular error message occurred. A WebSphere Data Interchange supplied handler or a user-written handler can then read these files from each destination and process them.
- The filtering messages to the Event Log, the Print File, the XML Print File, and the ADF Print File. Through filtering the you have the flexibility to eliminate messages that you consider unnecessary.
- The inclusion of more information about the documents that cause a particular error allowing you to more easily determine which input file or document caused a particular error. During DT translations the Document Store handle is included in Event Log entries
- The option to write Print File messages during Data Transformation processing, instead of holding messages in memory until processing is complete. This reduces memory usage when many messages are logged, and enables you to see progress during large translations.

Filtering messages

The Event Handler enhances the filtering of messages to the Event Log, the Print File, the XML Print File, and the ADF Print File. This enhancement enables users to eliminate messages that they consider unnecessary.

To filter messages WebSphere Data Interchange uses the Application Defaults profile fields, Event Destination profile, Error Message ID table, and Destination Properties table. The Error Message ID table contains entries associated with either an Application

Defaults profile or an Event Destination profile. The Destination Properties table contains entries associated with an Event Destination profile.

Within the Application Defaults profile there are error severity threshold fields used for message filtering. For the Event Log, the Print File, the XML Print File, and the ADF Print File, you can select to log all messages, messages with severity code 4 and higher, messages with severity code 8 and higher, messages with severity code 12, or no messages. You can also specify whether Event Logging should occur asynchronously or synchronously.

The Application Defaults profile contains an error message ID list. Selecting these error message IDs from EDIMSGS overrides the severity thresholds of the profile. Message IDs can be marked as follows:

- Include in Event Log
- Exclude from Event Log
- Neither include or exclude from Event log
- Include in Print File
- Exclude from Print File
- Neither include or exclude from Print file
- Include in XML Print File
- Exclude from XML Print File
- Neither include or exclude from XML Print file
- Include in ADF Print File
- Exclude from ADF Print File
- Neither include or exclude from ADF Print file

Ordering precedence

Filtering messages to the Event Log occurs in the following order of precedence:

1. If the Event Log Active box is not checked in the Application Defaults profile, no logging occurs.
2. Messages are written to the Event Log, if marked as Include in Event Log in the message ID list in the Application Defaults profile.
3. Messages are not written to the Event Log, if marked as Exclude from Event Log in the message ID list in the Application Defaults profile.
4. Messages meeting the Event Log Severity Threshold in the Application Defaults profile are written to the Event Log.

Filtering messages to the Print File will occur in the following order of precedence:

1. Utility keywords IGNOREINFO and IGNOREWARN filter severity level 0 and 4 messages respectively.
2. Utility keyword FILTERMSGS filters specified messages with severity less than 8.
3. Messages are written to the Print File, if marked as Include in Print File in the message ID list in the Application Defaults profile.

4. Messages are not written to the Print File, if marked as Exclude from Print File in the message ID list in the Application Defaults profile.
5. Messages meeting the Print File Severity Threshold in the Application Defaults profile are written to the Print File.

Filtering messages to the XML Print File will occur in the following order of precedence:

1. Utility keywords IGNOREINFO and IGNOREWARN filter severity level 0 and 4 messages respectively.
2. Utility keyword FILTERMSGS filters specified messages with severity less than 8.
3. Messages are written to the XML Print File, if marked as Include in XML Print File in the message ID list in the Application Defaults profile.
4. Messages are written to the XML Print File, if marked as Exclude from XML Print File in the message ID list in the Application Defaults profile.
5. Messages meeting the XML Print File Severity Threshold in the Application Defaults profile are written to the XML Print File.

Filtering messages to the ADF Print File will occur in the following order of precedence:

1. Utility keywords IGNOREINFO and IGNOREWARN filter severity level 0 and 4 messages respectively.
2. Utility keyword FILTERMSGS filters specified messages with severity less than 8.
3. Messages are written to the ADF Print File, if marked as Include in ADF Print File in the message ID list in the Application Defaults profile.
4. Messages are not written to the ADF Print File, if marked as Exclude from ADF Print File in the message ID list in the Application Defaults profile.
5. Messages meeting the ADF Print File Severity Threshold in the Application Defaults profile are written to the ADF Print File.

Filtering samples

The following are samples of how to accomplish message filtering in WebSphere Data Interchange.

- To view errors with severity 8 and higher in the Event Log, you would set the Event Log Severity Threshold to 8 in the Application Defaults profile.
- To write both errors with a severity 8 or higher and the map name being processed to an XML Print file, you would set a two fields in the Application Defaults profile:
  - Set the XML Print File Severity Threshold to 8
  - Select UT0008 from the message ID list and marked as Include in XML Print File.
- To write errors with severity 8 or higher to the Print File but not Profile member not found messages to a print file:
  - Set the Print File Severity Threshold to 8
  - Select PS0301 from the message ID list and mark as Exclude from Print File

Routing the print files

Using WebSphere Data Interchange, you can specify that the completed Print File, XML Print File, or ADF Print File is routed to a destination for further processing. This facilitates a notification process that might be used, for example, when errors are encountered during translation. Any number of destinations can be specified. For notification samples see, “Examples of notification based on severity” on page 315.

The destinations and criteria for routing the print files are defined in the Event Destination profile. This profile is part of the WebSphere Data Interchange client. There is one member per destination/routing criteria. Routing print files to their destinations occurs at the end of Utility invocations.

An Event Destination profile specifies the type of file to be routed (Print File, XML Print File, or ADF Print File), the severity threshold that causes the file to be routed, and the destination (MQ queue, file, directory, CICS TS queue, CICS TD queue, or CICS entry sequenced VSAM file).

The Event Destination profile can be tied to a particular APPLID (Application Defaults profile). If an APPLID is specified, the Event Destination profile would only apply to invocations of the Utility that use that APPLID. You can also specify that the end-of-job condition code is overridden. Here the highest condition code encountered within a triggered Event Destination or the job itself would be returned.

You can choose whether a header record should precede the Print File, XML Print File, or ADF Print File when routed to the destination. The header record is a predefined structure containing information from the Event Destination profile, including the profile’s properties. The event infrastructure will parse this record, and pass the information to the e-mail handler or other handler as properties. To see a sample of the header file for XML see“Samples print files” on page 317.

The Event Destination profile contains an error message ID list. Messages in this list override the severity threshold of the profile, and pertain only to messages actually written to the Print File, XML Print File, or ADF Print File (whichever is specified). Message IDs are marked as Force Routing to Destination or Ignore. If a message ID is marked as Force Routing to Destination, then that error causes a flag to be set so that the file type gets routed to the destination at the end of the Utility, regardless of the severity of the error. If a message ID is marked as Ignore, then that error does not cause the flag to be set, even if the severity of the error met the threshold.

During execution of the Utility, once an error message passes through Print File filtering and is written to the Print File, that message is used to determine whether the Print File is routed to a destination at the end of the Utility invocation. The order of precedence for whether a Print File gets routed is:

- If a message is marked as Force Routing to Destination in the message ID list of an applicable Event Destination profile, the Print File is flagged for routing to the destination.
- If a message is marked as Ignore in the message ID list of an applicable Event Destination profile, no routing evaluation takes place.
- If the severity of the message meets the Routing Threshold of an applicable Event Destination profile, the Print File is flagged for routing to the destination.

Flagging XML Print Files and ADF Print Files for routing to a destinations occurs similarly as above.

Printing and logging services

When the WebSphere Data Interchange Service Director initializes, the Application Defaults profile, all the Event Destination profiles, and the Error Message ID table are read. Relevant data is assembled into a control block (called the Event Filter Control Block), and hung off the CAB. The Event Logging service (EDIEL) and the Print File service (EDIFFPR) interrogates this control block as error messages are processed. The control block is used to filter messages and to determine, in the case of the Print File, whether the Print File gets routed to a destination for post-WebSphere Data Interchange processing. If it is determined that routing of the Print File should occur, that information is saved in the control block. When the Service Director terminates, the control block is freed. The CCB is passed to the Message Broker as errors are encountered, instead of ediLog building a linked-list. ediLog calls an EDIDTUTL function that calls Common Error Services, using the Service Director, to write the error message.

Common Error services (EDICEML)

Common Error services (EDICEML) is the common event handler for messages written to the Event Log and to the Print File. Common Error services calls the Event Logging service (EDIEL) to write to the Event Log, and calls the Print File service (EDIFFPR) to write to the Print File. Filtering occurs and destination trigger conditions are evaluated in the Event Logging service (for the Event Log) and in the Print File service (for the Print File), because there are instances where these services are called without going through Common Error services.

XML Print service

The XML Print File service is equivalent to the Print File service. The XML Print File service is called at the beginning of the Print File service. Just like the Print File service, messages are filtered and destination trigger conditions are evaluated in the XML Print File service. There can be messages in the regular Print File that are not in the XML Print File, and vice versa.

ADF Print service

The ADF Print File service is equivalent to the Print File service. The ADF Print File service is called at the beginning of the Print File service. Just like in the Print File service, messages are filtered and destination trigger conditions are evaluated in the ADF Print File service. There can be messages in the regular Print File that are not in the ADF Print File, and vice versa.

QSAM services

The Utility uses QSAM services to write Print Files, XML Print Files, and ADF Print Files to their destinations at the end of the Utility invocation. These destinations are any of the supported QSAM file types. A default schema is provided for XML Print Files, and the record format of ADF Print Files is provided.

At the end of the Utility session (in EDIFFUS) the Event Filter Control Block is interrogated and the Print File, the XML Print File, and the ADF Print File are routed to appropriate destinations. From these destinations a CEI handler, a user-defined handler, or an email notification program can be triggered.

Using the Common Events Handler

The Common Events Handler uses the information from the Application Defaults profile and the Events Destination profile. “Application Defaults profile” and “Events Destination profile” on page 313give a brief description of the fields that the Common Events Handler use in these profiles. See the *WebSphere Data Interchange for MultiPlatforms User's Guide* for more detailed information on these profiles.

The Error Message ID table is associated with the Application Defaults profile and the Events Destination profile that contain error messages. When the table is associated with the Application Defaults profile, message IDs can be selected for either inclusion or exclusion for the Event Log or any of the print files. When the table is associated with the Events Destination profile, message IDs can be selected for triggering routing or to be ignored for triggering. “Error Message ID table (EDIPSMS)” on page 314 gives a brief overview of the table. See *WebSphere Data Interchange for MultiPlatforms Mapping Guide* for more information on the Error Message ID table (EDIPSMS).

The Destination Properties table contains the destination properties. At the end of the Utility processing when the Event Filter Control Block is processed, the Destination Properties tables reads each destination triggered. The name and value pairs are included in the header record that is sent to the destination. “Destination Properties table (EDIPSDP)” on page 315 gives a brief overview of the table. See *WebSphere Data Interchange for MultiPlatforms Mapping Guide* for more information on the Destination Properties table (EDIPSDP).

Application Defaults profile

The fields associated with the Common Events Handler in the Application Default profile are:

1. Event Log Severity Threshold (ELOGSEV)
   a. Error messages with severity codes that meet this threshold are logged to the Event Log.
   b. Values: All Messages (0), 4, 8, 12, No Logging (-1)
   c. Default: 8
2. Print File Severity Threshold (PFILSEV)
   a. Error messages with severity codes that meet this threshold are logged to the Print File.
   b. Values: All Messages (0), 4, 8, 12, No Logging (-1)
   c. Default: All Messages (0)
3. XML Print File Severity Threshold (XFILSEV)
   a. Error messages with severity codes that meet this threshold are logged to the XML Print File.
   b. Values: All Messages (0), 4, 8, 12, No Logging (-1)
   c. Default: No Logging (-1)
4. ADF Print File Severity Threshold (DFILSEV)

   a. Error messages with severity codes that meet this threshold are logged to the ADF Print File.
   b. Values: All Messages (0), 4, 8, 12, No Logging (-1)
   c. Default: No Logging (-1)
5. Write Event Log Messages Asynchronously (LOGASYNC)
   a. This flag indicates whether Event Logging should occur asynchronously or synchronously. When logging is synchronous, the translation process waits each time a message is logged for the message to complete before proceeding.
   b. Values: asynchronous (Y), synchronous (N).
   c. Default: synchronous (N)

Events Destination profile

The fields associated with the Common Events Handler in the Events Destination profile are:

1. Event Destination Name (DESTID)

   A unique name that is the key for this profile.
2. Description and other common profile fields (DESCRIPT, LOGLOCK, LASTUID, and LASTUDT)
3. Profile Active flag (ACTIVE)
   a. This field indicates whether the profile is active or not.
   b. Values: activate (Y) or deactivate (N).
   c. Default: not active (N)

   Note: The Client's Activate/Deactivate function will use this field.
4. Type of File to be Routed (FILETYPE)
   - This field indicates whether the type of file to route is a Print File, an XML Print File, or an ADF Print File.
   - Values: Print File (P), XML Print File (X), ADF Print File (D)
   - Default: XML Print File (X)
5. Routing Threshold (ROUTESEV)
   a. Print Files, XML Print Files, or ADF Print Files containing messages with severity codes that meet this threshold are routed to the destination.
   b. Values: All Messages (0), 4, 8, 12
   c. Default: 8
6. Related Application Defaults Profile Name (APPLID)
   a. The name of the Application Defaults profile, if this Event Destination profile is to be associated with a specific Application Defaults profile (the APPLID when invoking the Utility). If a name is chosen, then this Event Destination profile is only applicable when the Utility is invoked with that APPLID. If a name is not chosen, then this Event Destination profile applies to all invocations of the Utility.
   b. Values: All Application Defaults profiles in the system or blank
   c. Default: Blank

7. Condition Code Override (CONDCODE)
    a. Overrides the WebSphere Data Interchange end-of-job condition code if routing a print file to this destination is triggered. The highest condition code encountered within a triggered Event Destination or the job itself is returned.
    b. Value: 2-digit, key-in field
    c. Default: Blank
8. Include Header Record? (HEADER)
    a. This indicates whether the print file should be preceded by a header record.
    b. Values: include header record (Y), do not include header record (N).
    c. Default: include header records (Y)
9. Destination Type (DESTTYPE)
    a. This indicates the type of the destination (blank, MQ, TD, TS, TM, VS).
    b. Values: WebSphere MQ Queue (MQ), Transient Data Queue (TD), Temporary Storage Queue (TS), Temporary Storage Queue Main (TM), Entry Sequenced VSAM File (VS)
    c. Default: WebSphere MQ Queue (MQ)
10. Destination Name (DESTNAME)
    a. This indicates the name of the destination.
    b. Value: 8-character string
    c. Default: Blank

Error Message ID table (EDIPSMS)

The information used by the Common Events Handler in the Error Message ID table are:

1. Profile Name (PROFID)

   The name of the associated profile (For example: APPDEFS or EVENTDST)
2. Profile Member Name (PROFKEY)

   The name of the associated profile member (For example: EDIFFS)
3. Error Message ID (MSGID)

   The six-character error message ID from EDIMSGS
4. Action flags (FLAGS)

   Array of Y/N/blank flags indicating whether the message should cause an action.
   - Position 1 (Event Log).

     APPDEFS:
     – Y = Write message to Event Log
     – N = Do not write message to Event Log
     – Blank = No special consideration.

     EVENTDST: Not applicable (always Blank).
   - Position 2 (Print File).

     APPDEFS:
     – Y = Write message to Print File

  - N = Do not write message to Print File
  - Blank = No special consideration.

  EVENTDST:
  - Y = Force routing to destination
  - N = Ignore from routing consideration
  - Blank = No special consideration.
- Position 3 (XML Print File).

  APPDEFS:
  - Y = Write message to XML Print File
  - N = Do not write message to XML Print File
  - Blank = No special consideration.

  EVENTDST:
  - Y = Force routing to destination
  - N = Ignore from routing consideration
  - Blank = No special consideration.
- Position 4 (ADF Print File).

  APPDEFS:
  - Y = Write message to ADF Print File
  - N = Do not write message to ADF Print File
  - Blank = No special consideration.

  EVENTDST:
  - Y = Force routing to destination
  - N = Ignore from routing consideration
  - Blank = No special consideration.

Destination Properties table (EDIPSDP)

The information used by the Common Events Handler in the Destination Properties table are:

1. Event Destination Profile Name (DESTID)

   The name of the Event Destination profile.
2. Name (PROPNAM)

   The property name to which a value will be associated.
3. Value (PROPVAL)

   The property value associated with the name.

Examples of notification based on severity

An operator wants to receive an email when a severity 8 or higher error occurs during WebSphere Data Interchange translation.

Notification in an MQ environment

You are running WebSphere Data Interchange in an MQ environment (z/OS batch, CICS, or multi-platform). To configure the e-mail alert, you set the XML Print File Severity Threshold to 8 or lower in the Application Defaults profile. Then you create an Event Destination profile with the following data:

```
Event Destination Name = Name of profile
Active Flag = Checked
Type of File to be Routed = XML
Print File Routing Threshold = 8
Destination Type = WebSphere MQ
Queue Destination Name = Name of a WebSphere Data Interchange WebSphere MQ
Queues profile.
```

The MQ queue must be defined with a trigger program. The trigger program reads the XML Print File data off the queue, parses it, and creates and sends an the appropriate e-mail notification.

Notification in a non-MQ z/OS environment

You are running WebSphere Data Interchange in a z/OS batch, non-MQ environment and want to set up an e-mail alert. To configure the notification, you set the XML Print File Severity Threshold to 8 or lower in the Application Defaults profile. Then you create an Event Destination profile with the following data:

```
Event Destination Name = Name of profile
Active Flag = Checked
Type of File to be Routed = XML
Print File Routing Threshold = 8
Destination Type = blank (defaults to DD name)
Destination Name = Name of destination file (DD name).
```

During translation, if a severity 8 or higher error occurs, an internal flag is set indicating that the XML Print File should be is to be copied to the destination file. At the end of processing, the Utility checks this flag. If it is set, the XML Print File (XMLPRNT) is copied to the destination file. A subsequent JOBSTEP following the Utility executes a program that attempts to read the destination file. If the file is empty, no action is taken. If the file is not empty, the program reads the XML Print File data, parses it, and creates and sends an appropriate e-mail notification.

Notification in a non-MQ CICS environment

You are running WebSphere Data Interchange in a CICS, non-MQ environment and want to set up an e-mail alert. To configure the notification, you set the XML Print File Severity Threshold to 8 or lower in the Application Defaults profile. Then you create an Event Destination profile with the following data:

```
Event Destination Name = Name of profile
Active Flag = Checked
Type of File to be Routed = XML
Print File Routing Threshold = 8
```

Destination Type = CICS Intra-partition TD Queue
Destination Name = A CICS TD queue defined with a trigger level and transaction ID.

This transaction reads the XML Print File data off the queue, parses it, and creates and sends the appropriate e-mail notification.

Notification in a non-MQ MultiPlatform environment

You are running WebSphere Data Interchange in a multi-platform, non-MQ environment. To configure the e-mail alert, you set the XML Print File Severity Threshold to 8 or lower in the Application Defaults profile. Then you create an Event Destination profile with the following data:

Event Destination Name = Name of profile
Active Flag = Checked
Type of File to be Routed = XML
Print File Routing Threshold = 8
Destination Type = blank
Destination Name = A file directory name.

A directory watcher program must be running that polls the directory. This program reads the XML Print File data from the directory when it arrives, parses it, and creates and sends the appropriate e-mail notification.

Samples print files

The following sections contain sample print files.

Sample Print File

```
1Audit Trail Report  -DataInterchange Utility-  Date: 05/04/07   Time: 10:48:19 Page: 00001

 FF0588 Command: PERFORM TRANSFORM WHERE INFILE(XMLFILE) OUTFILE(OUTFILE) SYNTAX(X) CLEARFILE(Y) TRACELEVEL(A2)

     Message: RU0003 Severity: 00
       The best rule match for the document was: map name POXML5SR-EDI, sending TP nickname ANY, receiving TP nickname ANY,
       usage indicator P, document POXML5SR, dictionary name TESTS, syntax xml.
     Message: UT0008 Severity: 00
       Map name being processed: POXML5SR-EDI.

 FF0007 Data was written to OUTFILE. Message control number or document id was 000000009.
 FF0585 The PERFORM TRANSFORM command completed successfully.
```

Sample XML Header Record

```
<?xml version="1.0" encoding="UTF-8"?>
<DestHeader>
   <DestProfile>DESTPROFILE1</DestProfile>
   <DestProperty>
      <PropertyName>email.SMTPSender</PropertyName>
      <PropertyValue>edi-system@xyz.com</PropertyValue>
   </DestProperty>
   <DestProperty>
      <PropertyName>email.SMTPReceiver</PropertyName>
      <PropertyValue>123-456-7890@mycellphone.com</PropertyValue>
   </DestProperty>
   <DestProperty>
```

```
      <PropertyName>email.SMTPServer</PropertyName>
      <PropertyValue>mysmtp-server.xyz.com</PropertyValue>
   </DestProperty>
</DestHeader>
```

Sample XML Print File

```
<?xml version="1.0" encoding="UTF-8"?>
<WDISessionLog>
   <Header>
      <Date>
         <Year>2005</Year>
         <Month>04</Month>
         <Day>07</Day>
      </Date>
      <Time>
         <Hour>10</Hour>
         <Minute>48</Minute>
         <Second>19</Second>
      </Time>
      <SessionOrigin>
         <host>RALVSK</host>
         <tpe>DSN</type>
         <handle>EDI.TESTIN(20)</handle>
      </SessionOrigin>
      <SessionConclusion>
         <UserCC>922</UserCC>
         <DataTarget>
            <TargetID>1</TargetID>
            <TargetType>MQ></TargetType>
            <TargetHangle=EDI_OUT>
            <TargetID>MQ_MQ22 5551212</TargetID>
         </DataTarget>
      </SessionConclusion>
   </Header>
   <Message>
      <MsgId>FF0588</MsgId>
      <Severity>0</Severity>
      <Text>Command: PERFORM TRANSFORM WHERE INFILE(XMLFILE)
OUTFILE(OUTFILE) SYNTAX(X) CLEARFILE(Y) TRACELEVEL(A2)</Text>
   <InsertData>PERFORM TRANSFORM WHERE INFILE(XMLFILE)
OUTFILE(OUTFILE) SYNTAX(X) CLEARFILE(Y) TRACELEVEL(A2)</InsertData>
   </Message>
   <DocInfo>
      <DocId>1111111111</DocId>
      <TargetID>1</TargetID>
      <SenderTPNick>ANY</SenderTPNick>
      <SenderId>ANY</SenderId>
      <SenderQual>ANY</SenderQual>
      <ReceiverTPNick>ANY</ReceiverTPNick>
      <ReceiverId>ANY</ReceiverId>
      <ReceiverQual>ANY</ReceiverQual>
      <Syntax>xml</Syntax>
      <Dictionary>TESTS</Dictionary>
      <Document>POXML5SR</Document>
   </DocInfo>
   <Message>
      <MsgId>RU0003</MsgId>
      <DocId>1111111111</DocId>
      <Severity>0</Severity>
      <Text>The best rule match for the document was: map name
POXML5SR-EDI, sending TP nickname ANY, receiving TP nickname ANY, usageindicator P, document POXML5SR, dictionary name TESTS, syntaxxml.</Text>
      <InsertData>POXML5SR-EDI</InsertData>
      <InsertData>ANY</InsertData>
      <InsertData>ANY</InsertData>
      <InsertData>P</InsertData>
      <InsertData>POXML5SR</InsertData>
      <InsertData>TESTS</InsertData>
      <InsertData>xml</InsertData>
   </Message>
   <Message>
      <MsgId>UT0008</MsgId>
      <DocId>1111111111</DocId>
      <Severity>0</Severity>
      <Text>Map name being processed: POXML5SR-EDI.</Text>
      <InsertData>POXML5SR-EDI</InsertData>
   </Message>
   <DocInfo>
      <DocId>2222222222</DocId>
      <DerivedFromDocId>1111111111</DerivedFromDocId>
      <SenderTPNick>ANY</SenderTPNick>
      <SenderId>ANY</SenderId>
      <SenderQual>ANY</SenderQual>
      <ReceiverTPNick>ANY</ReceiverTPNick>
      <ReceiverId>ANY</ReceiverId>
      <ReceiverQual>ANY</ReceiverQual>
      <Syntax>edi</Syntax>
      <Dictionary>X12V4R1</Dictionary>
      <Document>850</Document>
   </DocInfo>
   <Message>
      <MsgId>FF0007</MsgId>
      <DocId>2222222222</DocId>
      <Severity>0</Severity>
      <Text>Data was written to OUTFILE. Message control number or
document id was 000000009.</Text>
      <InsertData>OUTFILE</InsertData>
      <InsertData>000000009</InsertData>
   </Message>
   <Message>
      <MsgId>FF0585</MsgId>
      <Severity>0</Severity>
      <Text>The PERFORM TRANSFORM command completed successfully.</Text>
   </Message>
   <Trailer>
      <MaxSeverity>0</MaxSeverity>
   </Trailer>
</WDISessionLog>
.
```

Using the Common Events Handler Plug In

The Common Event Handler enables you to develop custom Java plug-in programs to process WebSphere Data Interchange generated Events. The Common Events Handler is a collection of Java classes designed to facilitate developing customized routines. These classes mediate between the Event Handler and the runtime environment to provide a consistent set of tooling on z/OS, CICS, UNIX and Windows. The classes provide:

- Abstraction of the physical source.
- Read-only Access to persistent application properties
- Access to LogEventSession, Events and Document Information objects

The Java classes are packaged in com.ibm.edi.wdievents.

The Common Event Handler improves the usability and performance of the WebSphere Data Interchange Event Log and Print File. It includes the following capabilities:

1. Creating an XML version of the Print File. This allows applications to easily parse the error messages and generate appropriate alerts or other notifications.
2. Creating an ADF (fixed record format) version of the Print File.
3. Routing print files to an MQ queue, file, directory, or CICS transient data or temporary storage queue for post-WebSphere Data Interchange processing. You can create Event Destination profiles to route print files to different destinations based on certain criteria, such as the maximum severity level or whether a particular error message occurred. A WebSphere Data Interchange supplied handler or a user-written handler can then read these files from each destination and process them.
4. Filtering messages to the Event Log, the Print File, the XML Print File, and the ADF Print File. This allows you to eliminate messages you might consider unnecessary.
5. Including more information about the documents that caused a particular error. This allows you to easily determine which input file or document caused a particular error. During Data Transformation translations the Document Store handle is included in Event Log entries. This is already done in Send and Receive translations.
6. Writing Print File messages during Data Transformation processing, instead of holding them in memory until Data Transformation processing is complete. This reduces memory usage when many messages are logged, and helps users see progress during large translations.

The Common Even Handler uses an interface called WDIEventHandler that works with the WebSphere Data Interchange Event Broker to provide specific processing of WDILogSession objects. When you implement the interface you must implement the handLogSession method of this class. During startup the Broker loads the WDIEventHandler implementation class specified in the properties file and then calls the event handler constructor.

```
WDIEventHandler=name of fully qualified user implementation class
such as com.user.MyEventHandler
WDISessionSource=WDIFile2Event
```

The broker creates the configured WDISessionSource implementation.

The plug-in Event handler is called and the WDILogSession is passed. WDILogSession contains the following objects which can be accessed using the appropriate methods:

WDILogDocument
WDILogEvent
WDILogDocInformation
WDIEventHandlerProperties

The broker checks the enumerated return value from the WDIEventHandler. If the value is OK, the broker will commit the event source, if the value is ERROR_FAIL the event source will be failed and the next WDILogSession object created from SessionSource.

The instance of the Event Handler is deleted when the Broker is shut down or the WDISessionSource is rolled-back, or returns a FATAL.

WDILogSession class

The WDILogSession class contains the methods and data necessary to parse and manipulate information from the common event service. For construction, the class requires an input stream in the local encoding. It pulls bytes off of the stream until it recognizes the preamble generated by WebSphere Data Interchange Common Event Services. Next, it pulls either a single WDISessionLog XML document, or an arbitrary buffer of unrecognized characters preceding either another preamble or the end of the input stream.

getDocument public
: com.ibm.edi.wdievents.WDILogDocument getDocument(java.lang.String docID)

getFirstDocument public
: com.ibm.edi.wdievents.WDILogDocument getFirstDocument()

getFirstEvent public
: com.ibm.edi.wdievents.WDILogEvent getFirstEvent()

getNextDocument public
: com.ibm.edi.wdievents.WDILogDocument getNextDocument()

getNextEvent public
: com.ibm.edi.wdievents.WDILogEvent getNextEvent()

getProperties public
: com.ibm.edi.wdievents.WDIEventHandlerProperties getProperties()

getSourceBuffer public
: java.lang.StringBuffer getSourceBuffer()

hasProperties public
: boolean hasProperties()

setProperties public
: void setProperties(java.util.Properties p)

getHeader public
: com.ibm.edi.wdievents.Header getHeader()

WDILogDocument class

The WDILogDocument class implements the encapsulation of the WDIDocInformation and WDILogEvents. WDILogDocument has a default constructor that creates an empty object. Getters can then be used to access the objects.

getDocInformation public
: com.ibm.edi.wdievents.WDIDocInformation getDocInformation()

getFirstEvent public
: com.ibm.edi.wdievents.WDILogEvent getFirstEvent()

getNextEvent public
: com.ibm.edi.wdievents.WDILogEvent getNextEvent()

Accessing the Destination information

The **WDILogEventHandlerProperties** class contains the destination information that the Event Handler can use for routing information.

The **WDILogEventHandler** class extends java.util.Properties. It provides layered overriding properties extracted from the properties file and set by the various classes that participate in event handling. The constructor searches for the default wdi.properties in the file system using the current directory. These properties take lower precedence than properties set by WDILogSessionSource or WDILogSession objects. Any properties set by these objects are removed when the objects are destroyed.

The Print File can contain an optional XML header, the Preamble ("Sample Event Preamble" on page 325). The preamble is the destination information from the Event Destination Profile and Associated Destination Properties. The Associated Destination Properties are included in the WDIEventLogHandlerProperties and override those from the properties resource file.

Accessing the Event Document

The WDILogDocument contains the events associated with the document.

```
WDILogDocument document = session.getFirstDocument();
document = session.getNextDocument()
```

Accessing Events

WDILogEvents can be accessed sequentially through the WDILogSession

```
WDILogEvent event = session.getFirstEvent()
event = session.getNextEvent()
```

The Events can be sequentially accessed through the associated WDILogDocument

```
WDILogEvent event = document.getFirstEvent()
event = document.getNextEvent()
```

Accessing Event elements

WDILogEvent elements can be accessed by their corresponding method names. All elements return type String. For example the event message text is accessed

```
String text = event.getText()
```

WDILogEvent class

The **WDIEvent** class encapsulates the Event message information.

getDocId public
: java.lang.String getDocId()

  Returns: Returns the docId.

getDocument public
: com.ibm.edi.wdievents.WDILogDocument getDocument()

  Returns: Returns the document.

getInsertData public
: java.lang.String[] getInsertData()

  Returns: Returns the insertData.

getMsgId public
: java.lang.String getMsgId()

  Returns: Returns the msgId.

getSeverity public
: java.lang.String getSeverity()

  Returns: Returns the severity.

getText public
: java.lang.String getText()

  Returns: Returns the text.

Accessing the Document containing an event

Some events are associated with a WDILogDocument.

```
WDILogDocument document = event.getDocument()
```

If there is no document associated with the event then null is returned. The event method getDocID will return an empty string "" if there is no associated document.

Configuring email notification

You can configure email notification through the properties file, the WDILogSession Preamble, and the WebSphere Data Interchange server database. Typical destination properties for SMTP are:

```
smtpServerAddress=127.0.0.1
userId=Administrator
```

Property names and values depend on which protocol provider is used.

Note: Properties set from the WDILogSession override properties from all other sources.

Prerequisites for configuring e-mail notification are:
- JavaMail Version 1.2 or later (http://java.sun.com/)
- JavaBeans Activation Framework Version 1.0.1 or later (http://java.sun.com/)

E-mail notification examples

Error notification example: You are assigned the responsibility to monitor the WebSphere Data Interchange system and want a notification email when an error occurs. You need the XML error log to be included in the email to aid in troubleshooting. One alert notification is to be sent for the LogSession.

Solution 1: You specify the error conditions, type of file to be routed, and destination of the file for the desired event using the destination profile. You then set the e-mail specific parameters in the Associated Destination Properties of the Destination Profile as follows:
- smtpServerAddress = SMTP server address
- toAdresses = comma separated address values
- userid = User ID for mail server authentication if applicable
- password = Password for mail server authentication if applicable
- subject = Text that goes into the subject of the email
- header = Text the goes in the header before the actual mail content
- trailer = Text that goes in the trailer after the actual mail content
- contentType = The content type either attachment or inline
- mailingFrequency = The number of times this notification can be sent within the configured time interval
- timeInetrval = The time interval (NotificationInterval) between resets of the Frequency counter

Solution 2: You carry a mobile phone and require a brief message along with reference to the output files.

You specifies error conditions, type of file to be routed, and destination of the file for the event using the destination profile. You then set the e-mail specific parameters in the database using the client as follows:
- SMTP server address
- Domain address
- To Addresses (comma separated address values)
- Userid (ID for mail server authentication if applicable)
- Password (Password for mail server authentication if applicable)
- Subject (Text that goes into the subject of the e-mail)

- Header (Text the goes before the actual mail content)
- Trailer (Text that goes after the actual mail content)
- saveLogDirectory=The directory to save the Session Log data file
- sessionLogFile=Name of the file to hold the Session Log data.
- Frequency number (Number of times this notification can be sent in the NotificationInterval)
- NotificationInterval (The time interval between reset of the Frequency counter)

Overdue functional acknowledgement notifications: You are an EDI administrator and need to keep track of any overdue functional acknowledgements. Whenever there are any overdue functional acknowledgements, you want to be notified by e-mail if any exist.

To accomplish this, create an event destination profile for overdue FA messages. Enter all other information required information, such as file type, in this destination profile. You then enter the e-mail specific parameters in the database using the client.

A job is scheduled to run the PERFORM command to report any overdue functional acknowledgements. This can either be made a manual process or automated process. To make it an automated process use job a schedule. If any overdue FA exists, the event occurs and the e-mail notification gets generated.

Sample Event Preamble

```
 <DestHeader>
   <DestProfile>DESTPROFILE1</DestProfile>
   <DestProperty>
      <PropertyName>email.SMTPSender</PropertyName>
      <PropertyValue>edi-system@xyz.com</PropertyValue>
   </DestProperty>
   <DestProperty>
      <PropertyName>email.SMTPReceiver</PropertyName>
      <PropertyValue>123-456-7890@mycellphone.com</PropertyValue>
   </DestProperty>
   <DestProperty>
      <PropertyName>email.SMTPServer</PropertyName>
      <PropertyValue>mysmtp-server.xyz.com</PropertyValue>
   </DestProperty>
</DestHeader>
```

Chapter 8. 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 5, “Interfacing to other networks and applications,” on page 275.

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 *WebSphere Data Interchange for MultiPlatforms Messages and Codes Guide*

Exit Function

The any-to-any mapping function 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:

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 will 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 332, and “Receive parameters” on page 333.
- 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 335 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 366 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 370 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 335 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 366 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 370 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 acknowledgement 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 acknowledgement 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 Document 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 366 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 370 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 192 and “Put envelope service” on page 193.

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 363 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 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 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.

FXXZ*ccc* 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 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 116.

Table 116. 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 117.

Table 117. 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.
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 Document 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 Document 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 Document 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 *WebSphere Data Interchange for MultiPlatforms Messages and Codes Guide* 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.
   - 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 361.

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 360. 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 360.

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 359.

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 471.

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 335, "C definition" on page 335, and "COBOL definition" on page 336.

Service name block (SNB): See "Service Name Block (SNB)" on page 366 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 370 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 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 374 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 359) and to put results ("Put data routine" on page 360). 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 359). 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 359. A sample authentication routine is provided in “Authentication examples” on page 466.

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 335, “C definition” on page 335, and “COBOL definition” on page 336. The parameter list consists of the following pointers:

Service name block (SNB): See “Service Name Block (SNB)” on page 366 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 370 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 374 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 359). 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 359). 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    */
    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 360.

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 359.

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 360.

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 359.

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 351, "C definition" on page 351, and "COBOL definition" on page 349

Service name block (SNB): See "Service Name Block (SNB)" on page 366 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 370 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 374 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 359) and to put results (“Put data routine” on page 360). 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 359). 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    */
```

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 118.

Table 118. 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 118. 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 119.

Table 119. 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 119. 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 120.

Table 120. 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 3, “Using WebSphere Data Interchange in the CICS environment,” on page 195). 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00, 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 370 for 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 121 on page 363.

Table 121. 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 340.

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
```

Appendix A. WebSphere Data Interchange control blocks

This appendix describes WebSphere Data Interchange control blocks, including block layouts and field descriptions. The control blocks are 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.V3R3M0.SEDIASM1
Assembler and COPY

EDI.V3R3M0.SEDICBL1
COBOL and CBLCPY

EDI.V3R3M0.SEDICCC1
C and H

EDI.V3R3M0.SEDIPLI1
PL/I and INCLUDE

The names and descriptions of the files are listed in Table 122. Use these files as a starting point for copy books that are tailored for use by your installation.

Table 122. 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 122. 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 123 on page 368 describes the layout of the SNB.

Table 123. 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 124 describes the layout of the CCB.

Table 124. 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 eye-catcher
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 124. 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 125 describes the layout of the FCB.

Table 125. 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 acknowledgement image
219	Retrieve transaction acknowledgement image

- **Communications services (COMM)**

211	Send transactions
221	Send files
232	Receive
233	Cancel

252	Process network acknowledgement
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 60, "Enveloping services" on page 118, and "Data extraction services" on page 153

Table 126 describes the layout of the TRCB.

Table 126. 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 126. 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 126. 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 acknowledgement 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 acknowledgement 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 126. 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 acknowledgement 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 126. 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 acknowledgement 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	Map chain 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 acknowledgement return code
FAERC	1152	4	Bin	Functional acknowledgement 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 acknowledgement 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 acknowledgement. You can control when functional acknowledgments are deenveloped. For more information, see the `ENVLDELAY` field on 395. 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 acknowledgement.

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 Document 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 Document 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 Document 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 Document 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 Document 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 Document 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 acknowledgement) and its image are saved in the Document 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 396.

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 acknowledgement 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 405.

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 “SCOPE” on page 398. 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 405.

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 acknowledgement 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 acknowledgement translation done during deenveloping. For more information about return codes, refer to *WebSphere Data Interchange for MultiPlatforms User's Guide*.

FAERC

The extended return code from the translator for the functional acknowledgement translation done during deenveloping. For more information about extended return codes, refer to *WebSphere Data Interchange for MultiPlatforms User's Guide*.

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 acknowledgement file is required. Valid values are:

Y	A functional acknowledgement is required. An error is produced if the file is not available.
(other)	A functional acknowledgement 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 89.

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. 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 *WebSphere Data Interchange for MultiPlatforms Mapping Guide*, SC23-5874-00.

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 127. 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 128. 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 129. 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 130. 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 130. 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 131. 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 132. 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 60, “Enveloping services” on page 118, and “Data extraction services” on page 153

Table 133 describes how the TRIDB is used for all functions and the initialization requirements for each.

Table 133. 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 151..
2	A BLKLEN of 16 is sufficient. For more information, see “Retrieve group header API” on page 151.
3	A BLKLEN of 16 is sufficient. For more information, see “Retrieve transaction header API” on page 152.
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 63.
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 91.
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 95.
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 136.
215	A BLKLEN of 16 is sufficient. For more information, see “Envelope API” on page 121.
216	A BLKLEN of 16 is sufficient. For more information, see “Retrieve detailed data API” on page 154.
217	A BLKLEN of 16 is sufficient. For more information, see “Retrieve transaction image API” on page 157.
218	A BLKLEN of 16 is sufficient. For more information, see “Retrieve transaction acknowledgement image API” on page 157.
219	A BLKLEN of 16 is sufficient. For more information, see “Retrieve functional acknowledgement image API” on page 157.
990	A BLKLEN of 16 is sufficient. For more information, see “Close and queue interchange API” on page 134.
991	A BLKLEN of 16 is sufficient. For more information, see “Issue commit API” on page 150

Table 133. 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 135.

Table 134 describes how to define the TRIDB with 4-byte lengths.

Table 134. 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 60, “Enveloping services” on page 118, and “Data extraction services” on page 153

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 135 describes shows how the TRODB is used for all functions and the initialization requirements for each.

Table 135. 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 151.
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 151.
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 152
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 69.
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 69.
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 91.
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 103.
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 95.
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 136.

Table 135. 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 121.
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 154.
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 157.
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 acknowledgement image API" on page 157.
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 acknowledgement image API" on page 157.
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 134.
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 150.
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 135.

Table 136 describes how to define the TRODB with 4-byte lengths.

Table 136. 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 137 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 137. 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 137. 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 acknowledgement 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 415.

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 acknowledgement 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 414.

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	WebSphere MQ 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 GXS Expedite product interface. GXS Expedite will set the data type to A or E, depending on which GXS 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 GXS 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 GXS 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 GXS Expedite parameter for network profiles is PRIORTY.

ACKIND

The type of acknowledgement 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 GXS 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 GXS 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 GXS 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 GXS 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 GXS Expedite parameter for network profiles is FILEID.

CANSD

The cancellation start date in `YYMMDD` format. Applies only to canceling files and messages.

The GXS Expedite parameter for network profiles is STARTDATE.

CANST

The cancellation start time in `HHMMSS` format. Applies only to canceling files and messages.

The GXS Expedite parameter for network profiles is STARTTIME.

CANED

The cancellation end date in `YYMMDD` format. Applies only to canceling files and messages.

The GXS Expedite parameter is ENDDATE.

CANET

The cancellation end time in `HHMMSS` format. Applies only to canceling files and messages.

The GXS Expedite parameter is ENDTIME.

TMZONE

The time zone for cancellation requests. Valid values are:

L	Local time
G	Greenwich Mean Time

The GXS 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 WebSphere MQ 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 138 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 138. 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 acknowledgement 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

Table 138. Definition of the Trading Partner Profile Block (continued)

Name	Offset	Length	Type	Description
MACHTYPE	369	1	Char	Machine type
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

Table 138. Definition of the Trading Partner Profile Block (continued)

Name	Offset	Length	Type	Description
TPSUPU8	888	40	Char	Comment line 8
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 GXS 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 GXS 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 GXS Expedite parameter for network profiles is CHARGE.

NETACK

Indicates which network acknowledgments are requested. Applies to send requests when the acknowledgement indicator in the CMCB contains binary zeros. For related information, see `ACKIND` on 417. 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 GXS 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 GXS 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 GXS Expedite Base/z/OS, the valid range is **001**-**180**. For GXS 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 GXS 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 GXS 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 GXS 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 GXS 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 GXS 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 GXS 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 GXS 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 GXS 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 139. 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 140. 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 141 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 141. 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 GXS Expedite parameter for network profiles is VANIINB1.

TMZONE

The time zone for your location. The network specifies the allowable codes.

The GXS 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 GXS 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 142 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 142. 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 acknowledgement
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 142. 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 GXS 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 GXS 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 GXS 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 GXS 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 GXS Expedite Base/z/OS, the valid range is **001-180**. For GXS 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 GXS 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 GXS 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 GXS 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 GXS 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 GXS 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 GXS 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 acknowledgement 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 GXS 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 GXS Expedite Base/z/OS compression table to decide whether to compress the data.

PRIORITY

Indicates to GXS Expedite Base/z/OS and GXS 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.V3R3M0.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.V3R3M0.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                                 */
/* ----------------------------------------------------------------- */
```

```
#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 Document Store

Sample programs for querying the WebSphere Data Interchange Document Store are provided for COBOL and PL/I.

Querying the Document Store using COBOL

A sample COBOL CICS program shows how to invoke the Utility to query the Document Store. The program library member name is `EDI.V3R3M0.SEDICBL2(FXXCOBU)`

Querying the Document Store using PL/I

A sample PL/I CICS program shows how to invoke the Utility to query the Document Store. The program library member name is `EDI.V3R3M0.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 457).

```
/* ------------------------------------------------------------------ */
/* 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       */

/* ----------------------------------------------------------------- */
/* 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   */
           /* ------------------------------------------------------ */
```

```
          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.

```
/* ------------------------------------------------------------------ */
/* 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);
   /* ------------------------------------------------------------------- */
   /* 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");
   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                               */
/* ---------------------------------------------------------------- */
```

```
   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                                   */
```

```
    /* 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                                 */
           /* ---------------------------------------------------- */
```

```
            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 Document 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.V3R3M0.SEDICBL1(EDISAMR1)` and `EDI.V3R3M0.SEDICBL1(EDISAMS1)`.

Generating a network activity report

A COBOL program that illustrates how to generate a Document 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.V3R3M0.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.V3R3M0.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.V3R3M0.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.V3R3M0.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 229.

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.V3R3M0.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.V3R3M0.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.V3R3M0.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.V3R3M0.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.V3R3M0.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 356.

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.V3R3M0.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.V3R3M0.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.V3R3M0.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 352.

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.V3R3M0.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        */
```

```
#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 348.

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.V3R3M0.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.

```
/* ---------------------------------------------------------------- */
/* $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   */
    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:
         /* ---------------------------------------------------------- */
         /* 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            */
```

```
         /* 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                   */
/*                                                                  */
/* 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.V3R3M0.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.V3R3M0.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.V3R3M0.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.V3R3M0.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.V3R3M0.SEDICBL2(FXXNETP)`.

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 143 on page 485
- Table 144 on page 486
- Table 145 on page 488
- Table 146 on page 492
- Table 147 on page 496
- Table 148 on page 500
- Table 149 on page 502
- Table 150 on page 506
- Table 151 on page 510

The table headings have the following meanings:

Record name
Identifies the DB2 table names in the WebSphere Data Interchange database records that you have installed.

Product
Specifies if the record applies to DB2.

Type
Specifies whether the database record is defined as a DB2 table or DB2 index.

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 143. Remote Command Execution tables

Record name	Product/ Type	Description
EDICMD	ALL/DB2 table	Command Table. Contains one row for each command entered in each command dictionary (for example each PERFORM). CMDNNUM = your estimate
EDICMDDICT	ALL/DB2 table	Command Dictionary Table. Contains one row for each command dictionary entry. CMDDNUM = your estimate
EDICMDREF	ALL/DB2 table	Command Reference Table. Contains one row for each DD statement defined for a Command CMDRNUM = your estimate
EDIDATAREF	ALL/DB2 table	Data Reference Profile Table. Contains one row for each DD statement defined as a Data Reference Service Profile DREFNUM = your estimate

Table 144. Database records (general)

Record name	Product/ Type	Description
EDIAUDITHDR	ALL/DB2 table	Audit Trail Table. Contains one row for each object updated by the Client or Server Import AUDRNUM = your estimate
EDIDBFILE1	ALL/DB2 table	Data Base Request Print Table. Contains one row for each print line that is the result of a DB Request, for example server command execution DBPRNUM = your estimate
EDIDBFILE2	ALL/DB2 table	Not currently used.
EDIDBREQ	ALL/DB2 table	Data Base Request Table. Contains one row for each remote submission of a Command object. DBRQNUM = your estimate
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
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

Table 144. Database records (general) (continued)

Record name	Product/ Type	Description
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 145. 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

Table 145. 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 145. 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

Table 145. 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

Table 145. Standards database records (continued)

Record name	Product/ Type	Description
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
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 146. 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

Table 146. 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 146. 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 Index. 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

Table 146. 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 146. 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 147. 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 147. 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

Table 147. Data formats database records (continued)

Record name	Product/ Type	Description
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
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

Table 147. Data formats database records (continued)

Record name	Product/ Type	Description
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
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

Table 147. Data formats database records (continued)

Record name	Product/ Type	Description
EDIDTDHDRY	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. XMLDTDY= XMLDTD
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
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
EDIXMLNS	All/DB2 table	XML Name Space table. Contains one row for XML name space imported as part of the XML schema. XMLNSUM = your estimate

Table 148. 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
EDIPSTP4	ALL/DB2 index	Trading Partner Profile Unique Index. There is one index entry for each EDIPSTP. PSTP4NUM = PSTPNUM
EDIPSTPX	ALL/DB2 index	Trading Partner Profile Unique Index. There is one index entry for each EDIPSTP entry. PSTPXNUM = PSTPNUM

Table 148. Trading Partner database records (continued)

Record name	Product/ Type	Description
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
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 148. Trading Partner database records (continued)

Record name	Product/ Type	Description
EDIPSBI	ALL/DB2 table	Trading Partner Business ID Table. Contains one row for each Business ID defined for each Trading Partner. PSBINUM = your estimate

Table 149. Set up database records

Record name	Product/ Type	Description
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
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.
EDISDP	ALL/DB2 table	Destination Profile Table. Contains one row for each Destination Profile defined. PSDPNUM = your estimate
EDIPSDIX	ALL/DB2 index	WebSphere Data Interchange Control File Unique Index. This table is static.
EDIPSED	ALL/DB2 table	Event Destination Profile Table. Contains one row for each Event Destination Profile defibedr. PSEDNUM = your estimate

Table 149. Set up database records (continued)

Record name	Product/ Type	Description
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
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
EDIPSMCD	ALL/DB2 table	Message Content Descriptor Profile Table. Contains one row MCD Profile defined. PSMCDNUM = your estimate
EDIPSMS	ALL/DB2 table	Client Message Table. Contains one row for each system message. PSMSNUM = MSGNUM
EDIPSMQ	ALL/DB2 table	WebSphere MQ Profile Table. There is one index entry for each WebSphere MQ profile member. PSMQNUM = number of WebSphere MQ profile members
EDIPSMQX	ALL/DB2 index	WebSphere MQ Profile Table Index. There is one index entry for each WebSphere MQ 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

Table 149. Set up database records (continued)

Record name	Product/ Type	Description
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
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
EDIPSSL	ALL/DB2 table	Service Profile Table. Contains one row for each Service Profile defined. PSSLNUM = your estimate
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. Not currently used.
EDIPSSYX	ALL/DB2 index	System Profile Unique Index. Not currently used.
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

Table 149. Set up database records (continued)

Record name	Product/ Type	Description
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
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

Table 149. Set up database records (continued)

Record name	Product/ Type	Description
EDIPSXEX	ALL/DB2 index	X Envelope Profile Table Index. There is one index entry for each type X envelope profile member (X12). PSXEXNUM = PSXENUM

Document Store tables

The tables listed in Table 150 are all part of the Document Store. The number of entries in the Document 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 Document Store. The default is 30 days.
2. The elapsed number of days between runnings of the Document 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 150. Document Store tables

Record name	Product/ Type	Description
EDIDSACT	ALL/DB2 table	Document Store Activity Table. Not currently used. DSACTNUM = 0
EDIDSACTATTR	ALL/DB2 table	Document Store Activity Attribute Table. Not currently used. DSAATNUM = 0
EDIDSATTR	ALL/DB2 table	Document Store Attribute Table. Contains one row each attribute of a document not specified as a field in the DS Document table, for example *Mapname* DSATTNUM = your estimate
EDIDSDOC	ALL/DB2 table	Document Store Document Table. Contains one row for each document in the store, for example, *XML Document* or *Data Format Document*. DSDOCNUM = your estimate
EDIDSDTL	ALL/DB2 table	Document Store Detail Table. Not currently used DSDTLNUM = 0

Table 150. Document Store tables (continued)

Record name	Product/ Type	Description
EDIDSEXTN	ALL/DB2 table	Document Store Extension Table. Contains one row for each row in DSDOC to house specific attributes. DSEXTNUM = DSDOCNUM
EDIDSIMG	ALL/DB2 table	Document Store Image Table. Contains one row for each 4K segment of a document image, for example *XML document*. DBRQNUM = your estimate
EDIDSREL	ALL/DB2 table	Document Store Relation Table. Not used at this time
EDIDSRELATTR	ALL/DB2 table	Document Store Relation Attribute Table. Not used at this time. DSRLANUM = 0
EDIDSSTATUS	ALL/DB2 table	Document Store Status Table. Contains one row for each document in the DSDOC to store status information. DSSTANUM = DSDOCNUM
EDITSTH	ALL/DB2 Table	Document 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	Document 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	Document Store Table Unique Index. TSTHONUM = TSTHNUM
EDITSTI	ALL/DB2 Table	Document 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	Document 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

Table 150. Document Store tables (continued)

Record name	Product/ Type	Description
EDITSTO	ALL/DB2 Table	For Send/Receive Translation, this only applies to send transactions and only for those transactions that supply override values in the C record or through the API. Overrides may not be supplied with non-C Record input and Non-API processing. 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. If the override values change, one record is created each time an override value changes from one transaction to the next. Override values provided through the API or on the C record will be used to override values found in the Trading Partner and Envelope profiles. For Data Transformation processing, envelope overrides are not limited to the C Record and API. Envelope values can be provided in the Trading Partner profile, Envelope Profile, using the C record, and mapping commands. Each PERFORM TRANSFORM command that results in an EDI target document being created will create a record in the EDITSTO table. Enveloping information from the Trading Partner Profile, Envelope profile, C record, and mapping overrides for each EDI target Document are stored in the EDITSTO table to enable Multilevel overrides for data transformation processing for immediate enveloping, for delayed enveloping, and reprocessing. Mapping override values can be provided in a Data Transformation Map using the SetProperty mapping command. TSTONUM = 0
EDITSTOX	ALL/DB2 Index	Document 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	Document 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
EDITSAUX	ALL/DB2 Index	Document 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	Document 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	Document 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

Table 150. Document Store tables (continued)

Record name	Product/ Type	Description
EDITSEVX	ALL/DB2 Index	Document 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	Document Store Envelope Unique Index. TSEV1NUM = TSEVNUM
EDITSGP	ALL/DB2 Table	Document 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	Document 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	Document 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	Document 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
EDITSTUY	ALL/DB2 Index	Document 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 each EDITSTU entry unless a large amount of REENVELOPE activity occurs. TSTUYNUM = TSTUNUM
EDITSLT	ALL/DB2 table	Document Store Lock Table. Contains a single record that is used as a lock table when update activity involves the Document Store. TSLTNUM = 1
EDITSIC	ALL/DB2 table	Tranaction Store Insert Control Table. Contains one row for each Transaction Handle during interchange processing. Rows are managed automatically. TSICNUM = 10

Security Tables

This section describes the security tables that are shipped with WebSphere Data Interchange for MultiPlatforms Version 3.3.

Table 151. Security tables

Record name	Product/ Type	Description
EDISECROLE	ALL/DB2 table	Security Role Table. Contains one row for each role defined in the Client RBAC. SECRNUM = your estimate
EDISECROLEDETROLE	ALL/DB2 table	Security Role Table. Contains one row for each role detail (role within role) defined in the Client RBAC. SECRRNUM = your estimate
EDISECROLEDETOB	ALL/DB2 table	Security Role Detail Object Table. Contains one row for each permission granted to a Role in the Client RBAC. SECRONUM = your estimate
EDISECUID	ALL/DB2 table	Security Userid Table. Contains one row for each userid defined in the Client RBAC. SECUNUM = your estimate
EDISECUIDDETROLE	ALL/DB2 table	Security Userid Detail Table. Contains one row for each role assigned to a Userid defined in the Client RBAC. SECURNUM = your estimate
EDISECGRP	ALL/DB2 table	Security Access Group Table. Contains one row for each access group defined in the Client RBAC. SECGNUM = your estimate
EDISECUIDDETGRP	ALL/DB2 table	Security Userid Detail Group Table. Contains one row for each access group assigned to a userid in the Client RBAC. SECUGNUM = your estimate
EDISECUIDDETOB	ALL/DB2 table	Security Userid Detail Object Table. Contains one row for each permission assigned to a userid in the Client RBAC. SECUONUM = your estimate

Allocation tables

This section describes the DB2 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 152 on page 511 and also provide information used in the “Space calculation scenario” on page 518.

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 152 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 152. DB2 database allocation

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
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

Table 152. DB2 database allocation (continued)

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
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
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
EDIAUDITHDR	500	50	3	1200	410
EDICMD	300	50	0	10200	29
EDICMDDICT	50	10	2	2258	19
EDICMDREF	100	20	12	323	272
EDICSTX	4800	2400	1	4032	1198
EDICSTXX	32	16	143	22	858
EDIDATAREF	50	20	8	468	87
EDIDBFILE1	100	50	0	8195	11
EDIDBFILE2	0	0	0	0	0
EDIDBREQ	500	100	0	8288	60
EDIDSACT	0	0	0	0	0
EDIDSACTATTR	0	0	0	0	0
EDIDSATTR	100	50	2	2368	39
EDIDSDOC	600	200	4	1069	552
EDIDSDTL	0	0	0	0	0
EDIDSEXTN	600	200	2	2566	233

Table 152. DB2 database allocation (continued)

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
EDIDSIMG	1000	400	2	2000	499
EDIDSREL	0	0	0	0	0
EDIDSRELATTR	0	0	0	0	0
EDIDSSTATUS	600	200	15	250	2221
EDIDTDHDR	5000	2500	2	1398	2496
EDIDTDHDRX	2000	800	53	60	26394
EDIDTD	50	25	12	320	126
EDIDTDX	50	20	50	64	515
EDIELOG	1000	500	9	440	2232
EDIELOGX	1000	500	44	72	10664
EDIELOG1	1000	500	39	80	9424
EDIELOG2	1000	500	35	88	8432
EDIENVP	10	5	29	140	15
EDIENVPX	10	5	320	8	160
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
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

Table 152. DB2 database allocation (continued)

Table name	Primary Allocation (Kilobytes)	Secondary Allocation (Kilobytes)	Records per page	Record size	Records in Primary Allocation
EDIMAPNODESX	2000	800	171	18	84660
EDIMAPCMDS	5000	2500	2	2072	2496
EDIMAPCMDSX	2000	800	171	18	53286
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
EDIPSDP	50	10	11	336	120
EDIPSED	50	10	19	195	198
EDIPSMS	1000	10	45	70	11084
EDIPSMCD	50	5	2	2528	17
EDIPSSL	50	10	1	3935	11
EDIPSBI	250	10	1	2894	84
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
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
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

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
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
EDISECROLE	10	2	17	212	9
EDISECROLEDETROLE	10	2	50	60	25
EDISECROLEDETOB	100	20	74	34	1701
EDISECUID	50	10	19	196	197
EDISECUIDDETROLE	10	5	60	47	30
EDISECGRP	10	2	17	212	9
EDISECUIDDETGRP	10	5	60	47	30
EDISECUIDDETOB	100	20	97	21	2227
EDITSIC	100	10	29	118	673
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
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
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
EDITSLT	10	5	4074	1	2037
EDITSTU	600	120	16	241	2368
EDITSTUX	150	30	32	94	1136
EDITSTUY	150	30	272	10	9656
EDIXMLDICT	50	25	34	118	357
EDIXMLDICTX	25	10	108	30	455
EDIXMLNS	50	10	6	688	60

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 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 Document Store, and the amount of DASD required for the Document 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 153 on page 520 describes the allocation required given the set of assumptions explained in “Space calculation scenario” on page 518.

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 152 on page 511.)

```
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 153. 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 153. 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

Table 153. 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 153. 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

Table 153. 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 153. 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

Table 153. 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

Space calculation worksheet

Use the worksheet in Table 154 to help you to calculating your space estimates for DB2 database records supplied.

Table 154. DB2 database allocation worksheet

Table name	Number of records required	DB2 primary allocation
EDIADFDICT		
EDIADFDICTX		
EDIADFFIELD		
EDIADFFIELDX		
EDIADFFIELDY		
EDIADFHDRMEM		
EDIADFHDRMEMX		
EDIADFHDRMEMY		
EDIADFHEADER		
EDIADFHEADERX		
EDIADFHEADERY		
EDIADFHEADERZ		
EDIADFLOOP		
EDIADFLOOPMEM		
EDIADFLOOPMEMX		
EDIADFLOOPMEMY		
EDIADFLOOPX		
EDIADFLOOPY		
EDIADFLOOPZ		

Table 154. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDIADFRECMEM		
EDIADFRECMEMX		
EDIADFRECMEMY		
EDIADFRECORD		
EDIADFRECORDX		
EDIADFRECORDY		
EDIADFRECORDZ		
EDIADFSTRUCT		
EDIADFSTRUCTMEM		
EDIADFSTRUCTMEMX		
EDIADFSTRUCTMEMY		
EDIADFSTRUCTX		
EDIADFSTRUCTY		
EDIAUDITHDR		
EDICMCD		
EDICMDDICT		
EDICMDREF		
EDICSTX		
EDICSTXX		
EDIDATAREF		
EDIDBFILE1		
EDIDBFILE2		
EDIDBREQ		
EDIDSACT		
EDIDSACTATTR		
EDIDSATTR		
EDIDSDOC		
EDIDSDTL		
EDIDSEXTN		
EDIDSIMG		
EDIDSREL		
EDIDSRELATTR		
EDIDSSTATUS		
EDIDTD		
EDIDTDHDR		
EDIDTDHDRX		

Table 154. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDIDTDX		
EDIELOG		
EDIELOG1		
EDIELOG2		
EDIELOGX		
EDIENVP		
EDIENVPX		
EDIMAPAPPLCNTL		
EDIMAPAPPLCNTLX		
EDIMAPAPPLCNTLY		
EDIMAPCMDS		
EDIMAPCMDSX		
EDIMAPELE		
EDIMAPELEX		
EDIMAPELEY		
EDIMAPGBLVAR		
EDIMAPGBLVARX		
EDIMAPHEAD		
EDIMAPHEADX		
EDIMAPLCLVAR		
EDIMAPLCLVARX		
EDIMAPNODES		
EDIMAPNODESX		
EDIMAPREF		
EDIMAPREFX		
EDIMAPSEG		
EDIMAPSEGX		
EDIMAPSEGY		
EDIMAPSEGZ		
EDIMAPSYNTAX		
EDIMAPSYNTAXX		
EDIMRCM		
EDIMRCMX		
EDIMRPC		
EDIMRPCX		
EDIMRPR		

Table 154. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDIMRPRX		
EDIMRPS		
EDIMRPSX		
EDIMRRT		
EDIMRRTX		
EDIMRST		
EDIMRSTX		
EDIMSGS		
EDIMSGSX		
EDIOWNR		
EDIPROF		
EDIPROFX		
EDIPSACX		
EDIPSAD		
EDIPSADX		
EDIPSAP		
EDIPSAPX		
EDIPSBI		
EDIPSCR		
EDIPSCRX		
EDIPSDI		
EDIPSDIX		
EDIPSDP		
EDIPSED		
EDIPSED		
EDIPSEE		
EDIPSEEX		
EDIPSEI		
EDIPSEIX		
EDIPSIE		
EDIPSIEX		
EDIPSLP		
EDIPSLPX		
EDIPSMCD		
EDIPSMQ		
EDIPSMQX		

Table 154. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDIPSMS		
EDIPSNO		
EDIPSNOX		
EDIPSNP		
EDIPSNPX		
EDIPSRQ		
EDIPSRQX		
EDIPSSL		
EDIPSSP		
EDIPSSPX		
EDIPSSY		
EDIPSSYX		
EDIPSTD		
EDIPSTDX		
EDIPSTE		
EDIPSTEX		
EDIPSTP		
EDIPSTP1		
EDIPSTP2		
EDIPSTPX		
EDIPSTT		
EDIPSTT1		
EDIPSTTX		
EDIPSTV		
EDIPSTVX		
EDIPSUE		
EDIPSUEX		
EDIPSXE		
EDIPSXEX		
EDIRECIDINFO		
EDIRECIDINFOX		
EDIRULE		
EDIRULE1		
EDIRULE2		
EDIRULE3		
EDIRULEX		

Table 154. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDISECGRP		
EDISECROLE		
EDISECROLEDETOB		
EDISECROLEDETROLE		
EDISECUID		
EDISECUIDDETGRP		
EDISECUIDDETOB		
EDISECUIDDETROLE		
EDISPTP3		
EDISSTK		
EDISSTKX		
EDISSTXX		
EDISTDCDN		
EDISTDCDNX		
EDISTDDED		
EDISTDDEDU		
EDISTDDEDX		
EDISTDDEDY		
EDISTDDEDZ		
EDISTDDEH		
EDISTDDEHX		
EDISTDDEHY		
EDISTDDSGD		
EDISTDENV		
EDISTDENVX		
EDISTDSGDU		
EDISTDSGDX		
EDISTDSGDY		
EDISTDSGDZ		
EDISTDSGH		
EDISTDSGHX		
EDISTDSGN		
EDISTDSGNX		
EDISTDSTH		
EDISTDSTHX		
EDISTDTXD		

Table 154. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDISTDTXDX		
EDISTDTXDY		
EDISTDTXDZ		
EDISTDTXH		
EDISTDTXHX		
EDISTDTXHY		
EDISTDTXN		
EDISTDTXNX		
EDITPCM		
EDITPCMX		
EDITPCN		
EDITPCNX		
EDITPCT		
EDITPCTX		
EDITPRT		
EDITPRTX		
EDITPRTY		
EDITPRTZ		
EDITPST		
EDITPSTX		
EDITPSTY		
EDITPSTZ		
EDITSAU		
EDITSAUX		
EDITSAUY		
EDITSEV		
EDITSEV1		
EDITSEVX		
EDITSGP		
EDITSGPX		
EDITSIC		
EDITSLT		
EDITSTH		
EDITSTHO		
EDITSTHX		
EDITSTI		

Table 154. DB2 database allocation worksheet (continued)

Table name	Number of records required	DB2 primary allocation
EDITSTIX		
EDITSTO		
EDITSTOX		
EDITSTU		
EDITSTUX		
EDITSTUY		
EDIXMLDICT		
EDIXMLDICTX		
EDIXMLNS		

Space calculation worksheet

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 536
- "WebSphere Data Interchange CICS considerations" on page 537
- "File maintenance techniques" on page 538
- "Document Store query techniques" on page 539

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 Document 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. 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.

- 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

- 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.

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 is 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 Document 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 485.
- 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 Document 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 Document 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 23 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.

Document 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 Document Store utility functions require a search and retrieval of information contained in the Document Store.

The following techniques apply to all PERFORM commands that require a database selection/query. These same techniques can be utilized with the interactive Document 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)
```

```
        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 WebSphere MQ 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.

Additional performance features for Data Transformation processing: best practices

The following information is provided as a set of best practices for Data Transformation performance processing.

- Running with Pageable AMM on at all times. The dynamic initiation of PAMM logic based on message size means smaller messages will be processed as usual, but large messages will invoke the special handling techniques in a ″dark room″ environment. Select a default PAGETHRESHOLD value that is consistent with company resources and imperatives. Increasing the PAGETHRESHOLD may be particularly useful if processing large EDI interchanges that contain many small to medium sized transactions.
- Only using TRACELEVEL in problem determination situations.
- Filter warning message using printed output limiting keyword options after evaluating the impact of messages on staff problem determination abilities and needs and frequency of message need.
- Using XMLSPLIT on a case by case basis; while XMLSPLIT is on by default, the feature is activated with the ″split tags″ specified with the XML DTD or schema.
- Using PARSEFILE on demand when large files are input; while WebSphere Data Interchange can process the file, traditional delivery mechanisms choke when a 1 or 2 GB input file enters the environment; it is anticipated that a message of this size will need special handling to make the file available to WebSphere Data Interchange.
- Running with the EDIWORK feature on at all times; again, this feature has a threshold trigger that indicates a potential memory resource issue; using the feature as a default prevents unforeseen issues.
- If possible, do not save transaction images in the transaction store–at least when processing large EDI Interchanges. Saving the images to the transaction store causes an additional copy of each transaction image to be held in memory during the DT processing.

Appendix F. Example Configurations of Java API and Email Alert for Common Event Handling

The following files for Java API to WebSphere Data Interchange Common Event Handling are required:

- edicevh.jar - WebSphere Data Interchange Common Event Handling Java API
- edicevhsample.jar - WebSphere Data Interchange Sample email notification event handler with java source
- env.profile - Java runtime environment variables for executing Java in z/OS batch JCL
- wdi.properties - default WebSphere Data Interchange Common Event Handling Java API properties

The sample email alert plug-in handler requires the additional jars from Sun JavaMail and Sun Activation Framework:

- mail.jar
- activation.jar

You will need to update the CLASSPATH value to point to the Common Event Handling API jars and the supporting java for the different wdiSource options such as WebSphere MQ Queues, z/OS datasets, or CICS TDQs.

For z/OS CICS, the basic CLASSPATH should look something like this:

```
CLASSPATH=/usr/lpp/cicsts/cicsts13/classes/dfjwrap.jar:
/usr/lpp/cicsts/cicsts13/classes/dfjcics.jar:
/usr/lpp/cicsts/cicsts13/classes/dfjcorb.jar:
/usr/lpp/java/current/J1.1/lib/classes.zip:
/u/edi/cicshome/mail.jar:
/u/edi/cicshome/activation.jar:
/u/edi/cicshome/edicevh.jar:
/u/edi/cicshome/edicevhsample.jar:
/u/edi/cicshome/wdi.properties:.
```

For z/OS JCL execution of the Java API and Email alert:

```
CLASSPATH=/u/edi/java:
/u/edi/java/activation.jar:
/u/edi/java/recjava.jar:
/u/edi/java/recordio.jar:
/u/edi/java/mail.jar:
/u/edi/java/edicevh.jar:
/u/edi/java/edicevhsample.jar:
/u/edi/java/wdi.properties:.
```

Note: *recjava.jar* and *recordio.jar* are supplied with IBM Software Development Kit for z/OS, Java 2 Technology Edition, Version 1.4. These support using DataSet as the session source.

For Windows and AIX, the CLASSPATH would need the following:

```
CLASSPATH=.\samples\activation.jar;.\samples\mail.jar;.\samples\edicevh.jar;.\samples\e
```

Substitute the .\ with the full WDIServer install directory path to fully qualify the jars.

The run time configuration of the Java API and Email is done through wdi.properties. This file is the default, but the property file used can be overridden at startup by specifying the Java system property *wdi.properties.file={alternate properties file name}*.

For example, instead of wdi.properties, we want to use a different properties file named *alternate.properties*. Then, the Java command to use this alternate properties file is *java -Dwdi.properties.file=alternate.properties com.ibm.edi.wdievents.WDIEventBroker .*

The CLASSPATH must include the *alternate.properties* file, otherwise the JAVA API will not be able to find it.

WebSphere Message Queues

If WebSphere MQ is to be used as the session source then the following jars must also be included in the CLASSPATH:

- com.ibm.mq.jar
- com.ibm.mqjms.jar
- connector.jar

These jar files are located in the IBM WebSphere MQ product Java lib directory. The wdi.properties file must be configured to use the queue wdiSource. Only one wdiSource is supported at run time in the WebSphere Data Interchange property file, but multiple alternate property files can be configured, each with a different wdiSource.

```
wdiSource=com.ibm.edi.wdievents.WDIQueue2Session
```

and the associated `Q` and `Qmgr` properties information for your installation must be be set. For example:

```
#WDIQueue2Session properties
queuename=wdinotifyq
dlqueuename=wdinotifybackoutq
qmanagername=QM_wdiMgr
hostname=localhost
channel=S_wdichan
```

Logging and Tracing

Logging and tracing information defaults to STDOUT and STRERR, in the wdi.properties file the logging and tracing output for the email plug-in can be redirected

```
#Logging and Tracing logFile=wdiemail.log saveLogDirectory=./ debug=true
```

The property *debug=true* is for the email alert to output SMTP trace information as a problem determination aid for connection problems with the server.

logFile=STDOUT results in the email feature outputting logging and tracing to the system *stdout*.

Destination Profile Overrides

The Destination Profile property values can override all values in wdi.properties except the wdiSource, the source's associated properties, and the plug-in handler. This is because the CEH Java API needs to know how to connect to the source of routed WebSphere Data Interchange print files, XML event files, etc prior to receiving the Destination Profile data that arrives though the connection to the wdiSource.

Email Alert Event Handler Plug-in

The Email alert Event Handler Plugin class is specified in the wdi.properties file by setting

```
wdiHandler=com.ibm.edi.wdievents.email.WDIEmailNotify
```

The URL or IP address of your smtp server must be set

```
smtpServerAddress=nnn.nnn.nnn.nnn
```

The from address of the email notification set

```
fromAddress=host@myaddress.com
```

The destination email address to receive the notification

```
toAddresses=admin@address.com
```

The SMTP server may require user ID and password authentication. These values can be entered here or in the Event Destination Profile.

```
userId=my_id password=my_password
```

If authentication is not required, then leave these values commented out. Using WebSphere Data Interchange Client, create a Destination Profile to output the *wdi prtfile* to the configured *wdiSource*.

Notices

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

IMS
WebSphere MQ
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.

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.

acknowledgement. See *functional acknowledgement*, *network acknowledgement*.

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.

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 acknowledgement. An electronic acknowledgement 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.

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 acknowledgement. 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*.

Document 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.

WebSphere MQ. An IBM product that is used to implement messaging and queueing of data groups. Earlier releases of this product were known as WebSphere MQ®.

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 V3.3 publications are:

- *WebSphere Data Interchange for MultiPlatforms Quick Start Guide* CF0YREN
- *WebSphere Data Interchange for MultiPlatforms Administration and Security Guide* SC34-6214-01
- *WebSphere Data Interchange for MultiPlatforms Messages and Codes Guide* SC34-6216-01
- *WebSphere Data Interchange for MultiPlatforms User's Guide* SC34-6215-01
- *WebSphere Data Interchange for MultiPlatforms Programmer's Reference Guide* SC34-6217-01
- *WebSphere Data Interchange for MultiPlatforms Mapping Guide* SC23-5874-00
- *WebSphere Data Interchange for MultiPlatforms Utility Commands and File Formats Reference Guide* SC23-5873-00
- *WebSphere Data Interchange for z/OS V3.3 Program Directory* GI10-2561-01
- *WebSphere Data Interchange for z/OS V3.3 Installation Guide* SC34-6269-01
- *WebSphere Data Interchange for z/OS V3.3 License File* GC34-6270-02

Softcopy books

All the WebSphere Data Interchange books are available in softcopy format.

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.

Index

Printed in USA

SC34-6217-01