

IBM Global Services



DataInterchange Programmer's Reference

Version 4 Release 1

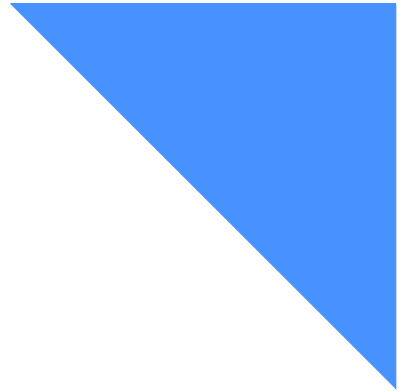
NOTE: Before using this information and the product it supports, be sure to read the general information under Notices on page 721.

Sixth Edition (January 2002)

This edition is a major revision and replaces document number SB34-2001-04.

© Copyright IBM Corporation 1998, 2002. All rights reserved.

Government Users Restricted Rights - Use, duplication, or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.



Contents

.....

To the reader	xv
Who should read this book	xv
How to use this book	xv
Syntax conventions used in this book	xv
Related books	xvi
Summary of changes	xvii
 Chapter 1. Using The Datainterchange Utility	1
Any-to-any data translation	2
Outbound processing using send maps	3
Fixed-to-fixed translation using send maps	3
Sending with fixed-to-fixed translation	4
Inbound processing using receive maps	5
Managing data	6
Removing and archiving event log entries	6
Reporting and extracting data	7
PRINT commands	8
Producing management reports from the Transaction Store	9
Creating management reporting reports	9
Creating transaction data or transaction envelope reports	10
Creating Transaction Store reports	11
Exporting and importing	12
Profile maintenance	12
Continuous receive	12
Reporting continuous receive status	13
Persistent environment	14
Using the DataInterchange Utility in the MVS environment	15

Chapter 2. DataInterchange commands and keywords	17
Command language syntax	17
Command language validation	19
Error filtering	21
Overriding utility condition codes	22
CLOSE MAILBOX command	23
DEENVELOPE command	24
DEENVELOPE AND TRANSLATE command	26
DELETE PROFILE command	28
ENVELOPE command	29
ENVELOPE AND SEND command	31
ENVELOPE DATA EXTRACT command	33
EXPORT command	36
GLB DUMP command	37
GLB TRACE command	38
HOLD command	39
IMPORT command	41
LOAD LOG ENTRIES command	42
NETWORK ACTIVITY DATA EXTRACT command	43
PRINT ACKNOWLEDGMENT IMAGE command	45
PRINT ACTIVITY SUMMARY command	47
PRINT EVENT LOG command	49
PRINT STATUS SUMMARY command	51
PRINT STATUS SUMMARY2 command	53
PRINT TRANSACTION DETAILS command	55
PRINT TRANSACTION IMAGE command	57
PROCESS NETWORK ACKS command	59
PURGE command	60
QUERY command	62
QUERY PROFILE command	64
RECEIVE command	65
RECEIVE AND DEENVELOPE command	66
RECEIVE AND SEND command	68
RECEIVE AND TRANSLATE command	69
RECONSTRUCT command	71
RECONSTRUCT AND SEND command	72
RCVFILE command	73
RCVFILE AND SEND command	74
REENVELOPE command	75
REENVELOPE AND SEND command	77
RELEASE command	79
REMOVE LOG ENTRIES command	81
REMOVE STATISTICS command	82
REMOVE TRANSACTIONS command	83
REPORT CONTINUOUS RECEIVE STATUS command	85
RESET STATISTICS command	86
RESTART RECEIVE command	87
RESTART SEND command	88
RETRANSLATE TO APPLICATION command	89
SAP STATUS EXTRACT command	91
SAP STATUS REMOVE command	92
SEND command	93

SENDFILE command	94
START CONTINUOUS RECEIVE command	95
STOP CONTINUOUS RECEIVE command	96
TRADING PARTNER CAPABILITY DATA EXTRACT command	97
TRADING PARTNER PROFILE DATA EXTRACT command	99
TRANSACTION ACTIVITY DATA EXTRACT command	101
TRANSACTION DATA EXTRACT command	103
TRANSFORM command	105
TRANSLATE AND ENVELOPE command	106
TRANSLATE AND SEND command	108
TRANSLATE TO APPLICATION command	110
TRANSLATE TO STANDARD command	112
UNLOAD LOG ENTRIES command	113
UNPURGE command	114
UPDATE STATISTICS command	116
UPDATE STATUS command	117
Keyword descriptions	118
Chapter 3. File formats and DataInterchange Utility records	173
Transaction Store input and output files	173
Command file (EDISYSIN or SYSIN)	173
DataInterchange DB2 command file (EDITSIN)	174
Network commands file (NETOP)	175
Application file	175
Envelope file	176
Exception file (FFSEXCP)	177
Tracking file (FFSTRAK)	178
Print file (PRTFILE)	179
Report file (RPTFILE)	180
Query file (EDIQUERY)	180
Work file (FFSWORK)	181
Pageable translation work file (EDIVAX)	182
Enveloping options file for functional acknowledgments (FAENV)	182
Export/Import utility function	186
Export/Import control file (CTLFILE)	187
Export/Import control file label descriptions	187
Export/Import files	190
Export/Import common control record (0C1/0C2)	191
Export/Import common end of group record (000)	192
Export/Import file data area	192
Exporting and importing trading partner profiles	193
Importing new definitions to DataInterchange	194
Export/Import trading partner profile header record (7P1)	194
Export/Import trading partner profile record (7P2)	194
Trading partner profile member (TPPROF-P2)	195
Trading partner profile member field descriptions	197
Trading partner contact definition (7P3)	209
Trading partner control numbers (7P4)	209
Comments definition (7A1)	210
Contact definition (7Z1)	210
Export and importing EDI standard records	211
EDI standard dictionary record (1Y1)	212

EDI standard transaction header record (1Y2)	213
EDI standard transaction detail record (1Y3)	213
EDI standard segment header record (1Y4)	214
EDI standard segment detail record (1Y5)	214
EDI standard data element header record (1Y6)	214
EDI standard data element detail record (1Y7)	215
EDI standard segment notes record (1Y8)	215
EDI standard transaction notes record (1Y9)	216
EDI standard composite data element notes record (1YA)	216
Exporting and importing data formats.	217
Data format dictionary record (2W1)	218
Data format record ID record (2W2)	219
Data format header record (2W3)	219
Data format loop record (2W4)	220
Data format record record (2W5)	220
Data format structure record (2W6)	221
Data format field record (2W7)	221
Data format header details record (2W8)	221
Data format loop details record (2W9)	222
Data format record details record (2WA)	222
Data format structure details record (2WB)	223
Exporting and importing maps	224
Map header record (3V1)	226
Map segment record (3V2)	227
Map element record (3V3)	228
Map application control record (3V9)	229
Map syntax record (3VA)	229
Map local variables record (3VB)	230
Map reference record (3VC)	230
Map nodes record (3VD)	230
Map commands record (3VE)	231
Global Variables Details Record (BK2)	231
Send usage record (3T5)	232
Receive usage record (3T6)	233
Data transformation map rule record (3T7)	234
Exporting and importing table definitions.	237
Export/Import table definition (B1)	237
Export/Import table definition (B1) field descriptions	238
Export/Import table entry (B2)	240
Table entry (B2) field descriptions	240
Exporting and importing XML records	241
XML dictionary record (AJ1)	242
XML DTD header record (AJ2)	242
XML DTD details record (AJ3)	243
Additional profile layouts.	244
Mailbox (requestor) profile (REQPROF-P2)	244
Network security profile (SECUPROF-P2)	245
Network profile (NETPROF-P2)	246
Network commands profile (NETOP-P2)	246
Activity log profile (ACTLOGS-P2)	247
Application defaults profile (APPDEFS-P2)	247
User exit profile (ADAMCTL-P2)	248

Language profile (LANGPROF-P2)	249
EDIFACT (E envelope) profile (E-P2)	249
ICS (I envelope) profile (I-P2)	251
UN/TDI (T envelope) profile (T-P2)	252
UCS (U envelope) profile (U-P2)	253
X12 (X envelope) profile (X-P2)	254
Continuous receive profile (CONTRECV-P2 for CICS only)	255
CICS performance profile (SYSPROF-P2 for CICS only)	256
MQSeries queue profile (MQSERIES-P2)	256
DataInterchange Utility records format	257
Control (C) records	257
Control record label descriptions	259
Data (D) records	268
Data record format - single structure	269
Data record label descriptions (single structure)	269
Data record format - multiple structures	269
Data record label descriptions (multiple structures)	269
End transaction and interchange (Z) records	270
Z record format	270
Z record label descriptions	270
Raw data records	271
Optional records	271
Information (I) records	273
Interchange header (E) records	274
Group header (G) records	275
Transaction set header (T) records	275
Queuing totals (Q) records	275
Management reporting	276
Trading partner profile data extract	276
Trading partner capability data extract	277
Network activity data extract	278
Transaction activity data extract	279
Transaction Store data extract information categories	280
Transaction Store data extract common key	281
Common key format	281
Transaction Store data extract record formats	282
Interchange data extract record layout	282
Group data extract record layout	283
Transaction data extract record layout	284
Application data extract record layout	287
Transaction/Acknowledgment image data extract record layout	288
Chapter 4. Using the DataInterchange application program interface	291
API languages	292
API link edit	292
DataInterchange/MVS and DB2 attachment	294
DataInterchange/MVS- CICS abend return codes	295
COBOL calls	295
PL/I calls	296
C calls	298
Assembler calls	299
API business tasks	302

Environmental services	304
Initializing the environmental API	304
Utility service API	306
Terminating the API	306
Translation services	308
Translation service functions	310
Translate-to-standard API	311
Translate, envelope, and send process diagram	314
Test translate-to-standard	315
Translation special considerations	316
Pageable translation	337
Translate-to-application API	338
Receive, deenvelope, and translate diagram	340
Test Translate-to-application	341
Translate-to-application API	341
Translate specific API	341
Translate-file-to-application API	350
Translate-to-application processing considerations (TA)	363
Enveloping services	365
Interchange layer	367
Group layer	367
Transaction layer	367
Enveloping service	368
Envelope API	368
Envelope processing considerations (EV)	379
Close and queue interchange API	382
End translation/enveloping API	384
Deenvelope API	384
Envelope processing and profile location considerations (E)	396
Deenvelope processing and profile location considerations (DE)	397
Issue commit API	399
Retrieve interchange header API	399
Retrieve group header API	400
Retrieve transaction header API	400
Data extraction services	402
Initialization for data extraction	402
Retrieve detailed data API	404
Retrieve transaction image API	406
Retrieve transaction acknowledgment image API	406
Retrieve functional acknowledgment image API	406
Communication services	407
Trading partner profile data block (TPPDB)	409
Common CMCB output fields	411
Return codes from communications	411
Send transactions and restart send transactions API	412
Send files API	417
Receive and restart receive API	420
Cancel API	424
Return filename API	426
Internal calls	427
Update status services	429
Update status API	430

Full envelope key	432
Alternate keys	433
SYNCPOINT services	436
DB2 TIMEOUT/DEADLOCK processing	437
Initialize SYNC function	438
COMMIT work function	439
ROLLBACK work	439
Get envelope service	441
Put envelope service	441
Chapter 5. Exit routines	443
Return codes	444
Exit languages	444
Exit linkage editor instructions	445
Field exit routines	446
Field exit routines shipped with DataInterchange	446
Send parameters	447
Service name block (SNB)	447
Common control block (CCB)	447
Field value	447
Field offset (4-byte binary value)	448
Field length (4-byte binary value)	448
Permanent work area (4096-byte buffer)	448
Temporary work area (1024-byte buffer)	448
Return field length (4-byte binary value)	448
Receive parameters	448
Service name block (SNB)	448
Common control block (CCB)	449
Data element value	449
Field offset (4-byte binary value)	449
Field length (4-byte binary value)	449
Permanent work area (4096-byte buffer)	449
Temporary work area (1024-byte buffer)	449
Return field length (4-byte binary value)	450
Field exit parameter language definitions	450
Assembler definition	450
C definition	450
COBOL definition	451
Transaction exit routines	453
Pre-translation exit	453
Post-translation exit	453
Pre- and Post-translation exit parameters	453
Translation exit language definitions	455
Get/Put envelope exit and service	457
Get envelope call	457
Put envelope call	457
FXXZccc stub program	458
Security routines	460
Enabling security during send	461
Enabling security during receive	462
Security parameters	462
Encryption routine	464

Encryption parameters	464
Encryption routine language definitions	466
Authentication routine	469
Authentication routine parameters	469
Authentication exit language definitions	471
Compression routine	474
Compression parameters	474
Filtering routine	475
Filtering parameters	475
Filtering exit language definitions	476
Security support routines	479
Get data routine	479
Put data routine	480
Call exit routine	481
Independent programs	482
Data extract exit	482
Get/Put envelope program	483
Chapter 6. Using DataInterchange in the CICS environment	485
Running the DataInterchange Utility in the CICS environment	485
Invocation options	486
Passing control information	486
Determining results	487
DataInterchange/MVS- CICS abend return codes	487
CICS storage mechanisms	487
CICS envelope queue alternatives	489
Pre- and Post-envelope programs	489
Processing multiple incoming TS queues	489
Ensuring serial processing of DataInterchange Utility files	490
Units of work and recovery considerations	491
DataInterchange Utility unit of work	493
Including DataInterchange changes in your application's unit of work	496
Terminal-attached applications	496
Running the DataInterchange Utility in a separate CICS region	497
DB2 setup considerations	497
CICS startup considerations	499
Running DataInterchange/MVS-CICS in a HOT-DI environment	500
Initializing HOT-DI	500
Initializing DataInterchange	501
Initializing multiple HOT-DI tasks	501
HOT-DI processing considerations	502
Call utility services	504
DataInterchange return code considerations	504
Terminating DataInterchange	504
Terminating HOT-DI	505
HOT-DI termination diagram	505
Outbound communications	505
DataInterchange Utility control information	507
Format of DataInterchange Utility control information	507
DataInterchange Utility control information field descriptions	509
Continuous receive considerations	516
Continuous receive using MQSeries	516

Continuous receive selection criteria	517
DataInterchange processing after data is received	518
Effects of defining the EDI1 TD queue	519
Sent to Network status	519
Using continuous receive outside Expedite/CICS	520
Response applications	521
Invoking your application	521
Types of response applications	521
Persistent environment	526
Running multiple MVS subtasks	526
Sizing the MVS data space	526
Enabling and disabling the persistent environment	527
Using multiple regions	527
Reserved TS and TD queues	527
TS queues that might require additional processing	527
Queues used by export and import	528
TD queues EDI2 and EDI3	529
Interface between DataInterchange/MVS-CICS, Expedite/CICS and Information Exchange	531
Information Exchange sessions	531
Information Exchange session cleanup	533
Continuous receive sessions	534
Starting and stopping continuous receive sessions	534
Continuous receive session cleanup	536
DataInterchange-supplied transactions	539
Performance monitor user exit	541
Format of performance monitor commarea	542
Using EDIWI to invoke the DataInterchange Utility	544
Chapter 7. Interfacing to other networks and applications	545
Generalized networks	546
Point-to-point networks	549
Activating point-to-point connections	549
Application-to-network flow diagram	551
Parameters passed to the communications routine	554
Building network commands	554
Network commands profile	555
Network command example	557
Message handler	559
Special communications routine for CICS	560
Network profile definition for CICS	560
Network program control information for CICS	561
Message handler control information	564
Continuous receive interface (CICS only)	565
Invoking the continuous receive interface	566
Interfacing with SAP	568
Outbound processing and SAP status	568
Inbound processing and SAP status	568
SAP status codes supported by DataInterchange	569
Extracting SAP status records	569
Removing SAP status records	569
Interfacing with MQSeries	570
XML special considerations	573

XML DTD resolution	573
XML encoding considerations	574
Appendix A. DataInterchange Control Blocks	577
Service Name Block (SNB)	579
SNB field descriptions	579
Common Control Block (CCB)	581
CCB field descriptions	582
Function control block (FCB)	584
FCB field descriptions	584
Translator Control Block (TRCB)	586
TRCB field descriptions	590
Translator Error Codes	614
Translator Input Data Block (TRIDB)	616
TRIDB field descriptions	617
Translator Output Data Block (TRODB)	618
TRODB field descriptions	619
Communication Control Block (CMCB)	621
CMCB field descriptions	622
Trading Partner Profile Block (TPPDB)	632
TPPDB field descriptions	634
Communication Data Block (DATBLK)	644
Data blocks up to 32-K bytes	644
Data blocks more than 32-K bytes	644
DATBLK field descriptions	644
Network Profile Block (NPDB)	645
NPDB field descriptions	646
Mailbox (Requestor) Profile Block (REQDB)	649
REQDB field descriptions	650
Appendix B. DataInterchange condition codes and API return codes	655
DataInterchange condition codes	656
TRANSFORM request return codes	659
Translation condition codes	661
Outbound translation considerations	661
Inbound translation considerations	662
Translation condition code tables	663
Communication condition codes	674
MVS condition codes	674
CICS condition codes	675
Combination command condition codes	677
TRANSLATE AND SEND errors	677
ENVELOPE AND SEND errors	677
RECEIVE Errors	677
API return codes	678
Initialization return codes	678
Negative return codes	679
DataInterchange termination return codes	679
Translation return codes	680
Normal or warning conditions	680
Data element errors	681
Segment errors	682

Transaction syntax errors	683
Transaction environmental errors	684
Group translation errors	686
Interchange translation errors	687
Invalid data errors	688
Environmental and program errors	689
Communication return codes	691
Communication extension codes	692
Status update return codes	693
SYNC initialization return codes	693
COMMIT request return codes	694
ROLLBACK request return codes	694
Appendix C. Using sample JCL	695
DataInterchange Utility (EDIUTIL) JCL	695
Section 1 JCL modifications	695
Section 2 DataInterchange Utility parameters	696
Section 3 STEPLIB requirements	697
Section 4 PRTFILE	697
Section 5 TRANSLATE TO STANDARD files	697
Section 6 Destination files	697
Section 7 FFSTRAK file	698
Section 8 FFSWORK file	698
Section 9 Envelope data file	699
Section 10 Network communications	699
Section 11 EDI standards	701
Section 12 ddname	702
Section 13 Report file	702
Section 14 Output file	703
Section 15 Export/Import statements	703
Section 16 Functional acknowledgment overrides	704
Section 17 Perform command file	705
Section 18 Pageable translation work file	706
Section 19 DB2 parameters	706
Section 20 XML parameters	707
Required utility data sets	708
Archive DB2 event log entries (EDIELARD)	710
Appendix D. Performance considerations	713
General optimization techniques	713
DataInterchange/MVS considerations	714
DataInterchange/MVS-CICS considerations	715
File maintenance techniques	717
Transaction Store query techniques	717
VS COBOL II field exit considerations	720
Appendix E. Notices	721
Trademarks and service marks	722
Glossary	723
Index	731



To the reader

This book provides information for the application programmers working with DataInterchange/MVS and DataInterchange/MVS-CICS Version 4 Release 1. It describes general-use programming and provides reference information for developing application programs that use DataInterchange.

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:

- Multiple Virtual Storage (MVS)
- Customer Information Control System (CICS)
- DataInterchange
- A programming language such as Assembler, C, or COBOL
- An IBM relational database management system such as Database 2 (DB2)

How to use this book

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 contain information related to the topics covered in this book. For your convenience, you can view these documents on the library page of the IBM Interchange Services Web site at: <http://www.ibm.com/services/e-business/interchange/>.

- *DataInterchange Administrator's Guide*, SB34-2002
This book provides information for the electronic data interchange (EDI) administrator about entering, sending, and receiving EDI transactions and other documents interactively.
- *DataInterchange Client User's Guide*, SB34-2010
This book provides information on the DataInterchange Client/Server user interface.
- *DataInterchange Installation Guide*, GB09-8070
This book provides information about installing DataInterchange on MVS and CICS systems.
- *DataInterchange Messages and Codes*, SB34-2000
This book provides information to assist you in diagnosing errors.
- *Expedite/MVS Host Programming Guide*, GC34-2242
This book provides a description of the MVS (batch) interface.
- *Expedite Base/MVS Programming Guide*, GC34-2204
This book provides a description of the Expedite Base/MVS communications program IEBASE.
- *Expedite/CICS Display Application User's Guide*, GC34-3303
This book provides information about the Expedite/CICS Display Application.
- *Customizing and Developing Applications with Expedite/CICS*, GC34-3304
This book provides a description of the CICS (interactive) interface.
- *Information Exchange Administration Services User's Guide*, GC34-2221
This book explains the major administrative tasks and describes how to use them.
- *Information Exchange Interface Programming Guide*, GC34-2222
This book provides information about programming for the Information Exchange interface.
- *VS COBOL II Application Programming Guide*, SC26-4045
This book provides information on run-time parameters for improving performance.
- *Using EDI VAN Interconnect*, GC34-2263
This book provides information about using EDI VAN Interconnect to connect to multiple networks.

Summary of changes

This edition of the DataInterchange Programmer's Reference is based on the DataInterchange/MVS and DataInterchange/MVS-CICS product set. The changes made to this Version 4.1 publication are as follows:

- Removed from the DataInterchange Programmer's Reference and detailed in the DataInterchange Client User's Guide:
 - Importing and requesting EDI standards
 - Customizing EDI standards and envelope standards
 - Mapping your application data to an EDI standard transaction set
 - Customizing business documents
- Moved to client/server support:
 - Mapping (envelope standards, defining and working with data formats)
- Removed from the DataInterchange Programmer's Reference and now only available on the product tape:
 - Sample programs
 - Copy books and include files
 - Space calculation formulas

Using The Datainterchange Utility

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

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

Any-to-any data translation

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

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

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

Data transformation maps are created using DataInterchange Client. For more information about the mapping features in DataInterchange Client, refer to the *DataInterchange Client User's Guide*.

Outbound processing using send maps

Outbound processing using send maps includes the following functions:

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

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

ENVELOPE	END
ENVELOPE AND SEND	SENDFILE
RCVFILE AND SEND	TRANSLATE AND ENVELOPE
REENVELOPE	TRANSLATE AND SEND
REENVELOPE AND SEND	TRANSLATE TO STANDARD

For more information on command syntax and command examples, see Chapter 2, “DataInterchange commands and keywords”.

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

Fixed-to-fixed translation using send maps

The same outbound processes used for EDI standard translations are also used for fixed-to-fixed translation and enveloping. Fixed-to-fixed translation remains available for use with existing maps in DataInterchange Version 4 Release 1, but all new maps should be translated using any-to-any translation.

When translating to an EDI standard data format, fixed-to-fixed translation is only done when the standard has an envelope type of **F**. The application data being translated can include a mixture of fixed-to-fixed and EDI standard data types. The results of the translation and an image of the data translated are saved to the Transaction Store in C and D record format and are eligible for enveloping. For more information on the command syntax and command examples, see “TRANSLATE TO STANDARD command” on page 112.

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

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

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

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

Sending with fixed-to-fixed translation

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

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

For more information on the command syntax and command examples, see Chapter 2, "DataInterchange commands and keywords".

Inbound processing using receive maps

Inbound processing using receive maps includes the following functions:

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

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

DEENVELOPE	RECEIVE AND TRANSLATE
DEENVELOPE AND TRANSLATE	RCVFILE
RECEIVE	RESTART RECEIVE
RECEIVE AND DEENVELOPE	RETRANSLATE TO APPLICATION
RECEIVE AND SEND	TRANSLATE TO APPLICATION

For more information on command syntax and command examples, see Chapter 2, “DataInterchange commands and keywords”.

In CICS, you can not use the combination RECEIVE commands when Management Reporting is turned on and SYNCVAL is -1. For more information on SYNCVAL, see “DataInterchange Utility control information field descriptions” on page 509.

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

Managing data

Data management includes the following functions:

- Rebuilding interchanges from transactions in the Transaction Store
- Updating status of transactions in the Transaction Store
- Removing transactions from the Transaction Store
- Updating management reporting statistics
- Removing management reporting statistics

The following commands are used for managing data:

HOLD	REMOVE LOG ENTRIES
LOAD LOG ENTRIES	REMOVE STATISTICS
PROCESS NETWORK ACKS	REMOVE TRANSACTIONS
PURGE	RESET STATISTICS
QUERY	UNLOAD LOG ENTRIES
RECONSTRUCT	UNPURGE
RECONSTRUCT AND SEND	UPDATE STATISTICS
RELEASE	UPDATE STATUS

For more information on command syntax and command examples, see Chapter 2, “DataInterchange commands and keywords”.

Removing and archiving event log entries

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

Remove DB2 log entries:	Run DataInterchange PERFORM command:	Run DBS Utility:
Without archive Without DB2 reorg	REMOVE LOG ENTRIES WHERE APPLID (<i>value</i>)	N/A
With archive Without DB2 reorg	UNLOAD LOG ENTRIES WHERE APPLID (<i>value</i>) ARCHIVEFILE (<i>value</i>) HOLDFILE (<i>value</i>)	N/A
Without archive With DB2 reorg	REMOVE LOG ENTRIES WHERE APPLID (<i>value</i>)	REORG TABLESPACE EDIELOG
With archive With DB2 reorg	UNLOAD LOG ENTRIES WHERE APPLID (<i>value</i>) ARCHIVEFILE (<i>value</i>) HOLDFILE (<i>value</i>)	REORG TABLESPACE EDIELOG

Reporting and extracting data

Data reporting and extraction includes the following functions:

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

DataInterchange provides two mechanisms for producing reports:

- Management Reporting data extracts
- Transaction Store data extracts

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

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

DataInterchange collects information in four management reporting categories and two Transaction Store reporting categories, as follows:

- Management Reporting Categories
 - Trading Partner Profile - provides general trading partner information alphabetically, by trading partner. Reported information includes company name, address, point of contact, telephone numbers, account ID, and user ID.
 - Trading Partner Capability - relates trading partner names used. For example, this report allows you to determine which maps are installed in test and production, which trading partners are using which translation usages/rules, direction of the translation, EDI standards used, transaction ID, total number of transactions processed, and number of transactions that had errors.
 - Transaction Activity - provides information relating transaction volumes to trading partner names. For example, this report can calculate the total number of transactions sent to a trading partner (by date and by map ID) or the total transactions errors by trading partner (and by date or map ID). This information is often used to gauge a trading partner's activity or to reconcile documents sent or received.
 - Network Activity - provides information such as network ID, name and account number, user ID, direction, charge code, and total number of bytes sent or received. This information is typically used to determine network charges, by trading partner or application.

- Transaction Store Reporting Categories
 - Transaction Data Extract - extracts detailed technical information from the Transaction Store. It provides data such as trading partner nickname, direction (S/R), transaction image, send acknowledgment data and image, receive acknowledgment data and image, and transaction handle. This information is commonly used for creating daily reports, such as overdue functional acknowledgments.
 - Envelope Data Extract - extracts detailed technical information from the Transaction Store. It provides the same types of information as the transaction data extract except envelope data extract only reports on transactions that have been enveloped, and it sorts the data differently. This information is commonly used for creating daily reports.

PRINT commands

You can use the PRINT commands with selection criteria to extract information from the Transaction Store and write formatted reports to ddname **RPTFILE**. To obtain the same type of information without formatting, use the ENVELOPE and TRANSACTION DATA EXTRACT commands. You can print the information using your system facilities. The cover page for each report lists the selection criteria you entered. You can view most of the information in these reports using the Transaction Store Facility. Samples of the different formats are listed in Chapter 2 with the commands that generate them. For more information, see “Producing management reports from the Transaction Store” on page 9.

The following commands are used for printing reports:

PRINT	PRINT STATUS SUMMARY
PRINT ACKNOWLEDGEMENT IMAGE	PRINT STATUS SUMMARY2
PRINT ACTIVITY SUMMARY	PRINT TRANSACTION DETAILS
PRINT EVENT LOG	PRINT TRANSACTION IMAGE

For more information on command syntax and command examples, see Chapter 2, “DataInterchange commands and keywords”.

Producing management reports from the Transaction Store

The following commands are used for extracting data from the Transaction Store:

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

For more information on command syntax and command examples, see Chapter 2, “DataInterchange commands and keywords”.

Creating management reporting reports

To create management reports, do the following:

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

For any trading partner profiles that you want reported, include selection information in the profile fields, such as a department name in a comment field. For more information on management reporting, see “TRADING PARTNER PROFILE DATA EXTRACT command examples” on page 100.

2. Execute the UPDATE STATISTICS command to move the pending information to the statistics tables. You must perform this step prior to data extraction. For more information, see “UPDATE STATISTICS command” on page 116.

PERFORM UPDATE STATISTICS

3. Execute the DATA EXTRACT command.

PERFORM DATA EXTRACT

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

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

For more information, see the report examples included with the data extract commands in Chapter 2.

4. Process the data extract file into reports by:
 - Verifying that you successfully created a QSAM data extract file (EDIQUERY) containing a sequential set of independent records with a record length of 1024. For more information, see “Management reporting” on page 276.
 - Sorting the data according to your requirements through a sort utility.
 - Formatting the data using a report-writing utility or a user-written program.
5. Remove previously collected daily statistics by executing the REMOVE STATISTICS command.

```
PERFORM REMOVE STATISTICS
```

Because statistics are not dependent on the Transaction Store, running the REMOVE TRANSACTIONS command does not affect them. For more information, see “PRINT commands” on page 8.



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

Creating transaction data or transaction envelope reports

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

- Reporting the number of purchase orders sent in a given period of time.
- Reporting the total number of bytes sent in a given period of time. (This can be useful for charging back costs to other departments based on their EDI usage.)
- Creating a customized functional acknowledgment tracking report, selected by application or by department. For example, you could create an Exception Report on Purchase Orders sent more than 2 days ago that have not been acknowledged.
- Creating an exception report flagging missing control numbers for inbound envelopes.
- Creating statistical reports identifying the most frequently used transactions, EDI standards, version sizes, or delimiters.

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

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

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

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

For more information on report record layouts, see “Transaction data extract record layout” on page 284.

Creating Transaction Store reports

To create Transaction Store reports, do the following:

1. Select the type of information to collect in the Transaction Store:
 - a. To collect information, set the **Trx. store active?** field in the APPDEFS profile to a value other than N.
 - b. To collect transaction images, set the **Trx. image wanted?** field in the APPDEFS profile to a value other than N.
 - c. To collect functional acknowledgment images, set the **FA image wanted?** field to a value other than N.
2. Perform the usual translations and communications.
3. To extract the information for the reports, execute the appropriate DATA EXTRACT command using one of the following commands:

```
PERFORM TRANSACTION DATA EXTRACT
```

or

```
PERFORM ENVELOPE DATA EXTRACT
```

4. Create a user-written program to format the data extract file output (EDIQUERY) into reports.
5. Remove transactions periodically using the following command:

```
PERFORM REMOVE TRANSACTIONS
```

For more information on these commands, see “Reporting and extracting data” on page 7.

Exporting and importing

The EXPORT and IMPORT commands allow you to export or import the following:

- Maps (without reworking the entire map)
- Trading partner setup information (such as profile members)
- Administrative data

You tell DataInterchange specifically which setup information you want to exchange using a control file. For more information on the Export/Import control file, see “Export/Import control file (CTLFILE)” on page 187. For more information about using the Export/Import utility, refer to the *DataInterchange Administrator's Guide*.

The following commands are used for exporting and importing:

```
CLOSE MAILBOX  
EXPORT  
IMPORT
```

For more information on command syntax and command examples, see Chapter 2, “DataInterchange commands and keywords”.

Profile maintenance

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

```
DELETE PROFILE  
QUERY PROFILE
```

For more information on command syntax and command examples, see Chapter 2, “DataInterchange commands and keywords”.

Continuous receive

When a continuous receive is started through DataInterchange/MVS-CICS, DataInterchange and Expedite/CICS each keep a control record about the continuous receive. Although DataInterchange and Expedite/CICS can get out of sync when managing continuous receives, this is an exception to normal processing. For more information about out of sync situations, see “Continuous receive session cleanup” on page 536.

The following commands are used for continuous receive:

```
REPORT CONTINUOUS RECEIVE STATUS  
START CONTINUOUS RECEIVE  
STOP CONTINUOUS RECEIVE
```

For more information on command syntax and command examples, see Chapter 2, “DataInterchange commands and keywords”.

Reporting continuous receive status

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



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

Continuous receive status codes

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

Continuous receive status report record format

Name	Offset	Length	Type	Description
CRSTATUS	0	16	Char	Report status
CDPROFID	16	16	Char	DataInterchange continuous receive profile member
CDREQACT	32	08	Char	DataInterchange requestor's account

Name	Offset	Length	Type	Description
CDREQUSE	40	08	Char	DataInterchange requestor's user ID
CDUNIQUE	48	08	Char	DataInterchange continuous receive unique ID
CDMSGUC	56	08	Char	DataInterchange message user class
CDTPACCT	64	08	Char	DataInterchange trading partner's account
CDTPUSER	72	08	Char	DataInterchange trading partner's user ID
CEREQACT	80	08	Char	Expedite/CICS account
CEREQUSE	88	08	Char	Expedite/CICS user ID
CEUNIQUE	96	08	Char	Expedite/CICS continuous receive unique ID
CEMSGUCL	104	08	Char	Expedite/CICS message user class
CETPACCT	112	08	Char	Expedite/CICS trading partner's account
CETPUSER	120	08	Char	Expo/CICS trading partner's user ID
RESERVED	128	04	Char	Blanks

Persistent environment

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

The following commands are used for debugging the persistent environment:

```
GLB DUMP
GLB TRACE
```

For more information on command syntax and command examples, see Chapter 2, "DataInterchange commands and keywords".

Using the DataInterchange Utility in the MVS environment

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

APPLID=aaaaaaa	Specifies the application ID to run the DataInterchange Utility. This parameter also identifies the log file specified in the ACTLOGS profile. Replace <i>aaaaaaa</i> with the ID of the application that initialized DataInterchange. If you specify this parameter, the activity log profile must contain a matching entry to define which log file is used for recording errors and events pertaining to the application. The two APPLID values shipped with DataInterchange are: EDIFFS Associated with the LOGFFS ddname. The default (default) APPLID and log when using the utilities EDIMP Associated with the LOGEDI ddname. The APPLID and log used during online DataInterchange processing
SYSID=bbbbbbb	Identifies the installation-defined DataInterchange/MVS system used to run the EDIUTIL utility. Specifying this parameter controls access to various components of DataInterchange. The SYSID is part of the resource name defined using RACF or some other resource control product. Replace <i>bbbbbbb</i> with the ID used to protect DataInterchange services (for example, RACF). The default ID is DIENU .
LANGID=ccc	Identifies the language ID used to run the EDIUTIL utility. The value supplied must match an entry in the LANGPROF profile. The language ID is used to establish values such as date formats and decimal notation. Replace <i>ccc</i> with the following value for the language version: ENU English
DLM=d	Specifies the delimiter used in place of left and right parentheses to enclose values in the EDIUTIL utility command language. Replace <i>d</i> with the delimiter you want to use in place of the left and right parentheses to enclose values in the DataInterchange Utility command language. You must supply this parameter if a keyword value contains either a left or right parenthesis.
MQSYSIN=eeeeeee	Replace <i>eeeeeee</i> with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to either ddname SYSNAME or ddname EDISYSIN.
MQPRT=ffffff	Replace <i>ffffff</i> with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname PRTFILE.
MQRPT=gggggggg bbbbbb	Replace <i>gggggggg</i> with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname RPTFILE.

MQEXCP=hhhhhhhh	Replace <i>hhhhhhhh</i> with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname FFSEXCP.
MQTRAK=iiiiiii	Replace <i>iiiiiii</i> with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname FFSTRAK.
MQQUERY=jjjjjjj	Replace <i>jjjjjjj</i> with the DataInterchange MQSeries Queue profile member you want to use instead of a sequential file allocated to ddname EDIQUERY.
PLAN=kkkkkkkk	In a DB2 environment, if the utility is invoked by way of IKJEFT01 and there is no EDITSIN data set allocated, then this parameter is required. For more information, see “DataInterchange DB2 command file (EDITSIN)” on page 174. Replace <i>kkkkkkkk</i> with the DB2 plan name.
SYSTEM=///	In a DB2 environment, if the utility is invoked by way of IKJEFT01 and there is no EDITSIN data set allocated, then this parameter is required. For more information, see “DataInterchange DB2 command file (EDITSIN)” on page 174. Replace <i>///</i> with the name of the DB2 subsystem (or group, if Data Sharing).

Optional JCL parameter example

```
PARM( ' SYSID=TEST  LANGID=ENU  SYSTEM=DB93  PLAN=EDIENU41 ' )
```

DataInterchange commands and keywords

This chapter explains how to use the DataInterchange Utility command language. You can invoke the DataInterchange Utility from an MVS environment or an MVS-CICS environment. For additional information about invoking the DataInterchange Utility in an MVS environment, see “Using the DataInterchange Utility in the MVS environment” on page 15. For additional information about invoking the DataInterchange Utility in a MVS-CICS environment, see “Running the DataInterchange Utility in the CICS environment” on page 485.

Command language syntax

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

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

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

```
PERFORM TRANSLATE TO APPLICATION
```

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

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

```
WHERE TPNICKN(PISCES)
```



NOTE: In MVS, you can redefine the parentheses delimiter with the **DLM** parameter. For more information, see “Using the DataInterchange Utility in the MVS environment” on page 15.

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

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

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

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

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

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

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

The command language format is free-form. You can insert blanks between keywords to improve readability, but do not insert blanks between a keyword and its associated value. Characters can be in upper, lower, or mixed case. Most fields are not case sensitive. To determine if a field is case sensitive, see the field's definition in “Keyword descriptions” on page 118.

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

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

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

DATE, TIME, and HANDLE keywords

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

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

Command language validation

DataInterchange performs basic validation on the command language input, as follows:

- Each PERFORM command (such as TRANSLATE TO STANDARD) is validated. DataInterchange generates an error if the command is not valid.
- All keywords are validated. DataInterchange generates an error if a keyword is not valid.
- If a keyword is specified, then an associated value must also be specified. DataInterchange generates an error if you do not specify a keyword value.
- You must enclose each keyword value with beginning and ending delimiters. The default delimiters are opening and closing parentheses. DataInterchange generates an error if you do not enclose a keyword value with delimiters.
- The length of each keyword value is validated against the maximum allowable length for that keyword. DataInterchange generates an error if the length of the value you supply is greater than the maximum allowable length for the keyword.
- Keywords cannot be used more than once in a WHERE clause. However, you can specify multiple WHERE clauses. DataInterchange generates an error if you specify a keyword more than once in the same WHERE clause.
- Date and time fields are validated. DataInterchange generates an error if the date or time values you specify are not valid.

- When a keyword has an associated default value, keyword values that are not valid are ignored, and defaults are used. An example of a keyword value that is not valid is **RAWTEST(X)**. In this situation, DataInterchange ignores the value of **X** and does not generate an error. For specific default values, see “Keyword descriptions” on page 118.
- If a keyword is used as part of selection criteria against the Transaction Store, the associated value is accepted. If the associated value is not valid, DataInterchange uses the value as part of the selection criteria, and does not find a transaction match. Verify the associated values for each keyword used for Transaction Store selection criteria.

Error filtering

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

When the **DIERRFILTER** variable is specified as **SET** or **SAVED**, the translator parses the value of the variable as an indicator of the errors to ignore. The **DIERRFILTER** variable value should consist of a list of the error codes to be ignored. These values are documented in “Translator Error Codes” on page 614 and in “DataInterchange condition codes” on page 656, where translator errors are documented.

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

Overriding utility condition codes

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



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

These keywords are entered as follows:

- IFCC(*c1,c2,c3,c4,c5,c6,c7,c8,c9,c10*)

The values *c1-c10* represent the utility condition codes to be checked.

- SETCC(*n1,n2,n3,n4,n5,n6,n7,n8,n9,n10*)

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

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

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

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



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

CLOSE MAILBOX command

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

Syntax

CLOSE MAILBOX

REQID(*requestor ID*)

CLOSE MAILBOX command example

Close the mailbox for requestor **DEPTA7F**.

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

DEENVELOPE command

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

Syntax

DEENVELOPE

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

DEENVELOPE command examples

Example 1: Deenvelope the transactions previously received for requestor ID **DEPTA7F**. Place the results in the Transaction Store. Mark these transactions for purging after 10 days in the store.

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

Example 2: Deenvelope the transactions in ddname **A7FIN**. Place the results in the Transaction Store. Do not allow duplicate interchanges to be processed.

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

DEENVELOPE AND TRANSLATE command

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



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

Syntax

DEENVELOPE AND TRANSLATE

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

DEENVELOPE AND TRANSLATE command examples

Example 1: Deenvelope and translate the EDI documents in the receive file specified by requestor ID **DEPTA7F**.

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

Example 2: Deenvelope and translate the EDI documents in the receive file specified by requestor ID **DEPTA7F** into raw data format. Return the information records to the exception file.

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

DELETE PROFILE command

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

Syntax

DELETE PROFILE

ID(*DataInterchange profile name*)

MEMBER(*member profile name*)

DELETE PROFILE command examples

Example 1: Delete trading partner profile member **TPMEM**.

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

Example 2: Delete mailbox (requestor) profile member **REQMEM**.

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

ENVELOPE command

This command takes the EDI transactions from the Transaction Store that are available for enveloping, envelopes them, and places the results in an envelope file. The envelope file is the TD queue specified in the network profile member or a file specified in the command. Your selection criteria determine which transactions go into the interchange envelope. The enveloper sorts the transactions to create the fewest number of functional groups and interchange envelopes. For more information about envelope processing, see “Enveloping services” on page 365.

Syntax

ENVELOPE

ACFIELD(*starting application control field data*) TO(*ending application control field data*)

APPLID(*application ID*)

BATCH(*translated transaction batch ID*)

ENVPRBREAK(*start new envelope*)

EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)

FILEID(*processing file ddname*)

FIXEDFILEID(*fixed-to-fixed output ddname*)

FORMAT(*data format ID*)

HANDLE(*starting transaction ID*) TO(*ending transaction ID*)

IACCESS(*IEXIT access*)

IAREA(*IEXIT information*)

IEXIT(*interchange control program*)

IFCC(*override condition codes*)

INMEMTRANS(*transactions in memory*)

ITPBREAK(*new interchange envelope*)

ITYPE(*IEXIT program type*)

NETID(*network ID*)

OPTRECS(*optional record type*)

PAGE(*pageable translation*)

RAWDATA(*translate to raw data format*)

RECOVERY(*recovery unit of work*)

SAPUPDT(*track SAP status*)

SERVICESEGVAL(*service segment validation level*)

SETCC(*condition codes*)

STDTRID(*EDI standard transaction set ID*)

TPID(*trading partner ID*)

TPNICKN(*trading partner nickname*)

TRERLVL(*maximum translation error level*)

TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)

TRXTIME(*starting transaction time*) TO(*ending transaction time*)

VERIFY(*verify transaction status*)

XML(***XML required***)

XMLDICT(***XML dictionary address***)

XMLDTDS(***XML DTD path***)

ENVELOPE command examples

Example 1: Envelope all the EDI documents in the Transaction Store that are associated with data format ID **PO850** or **CORP861**, and with network **IINR41**. The **NETID** keyword is included in both WHERE clauses to make sure that only **IINR41** transactions are selected. (Each WHERE clause is a separate set of selection criteria.)

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

Example 2: Envelope all the EDI documents in the Transaction Store that are associated with data format ID **PO850** or **CORP861**, and with network **IINR41**. Place the results in file **NETQUEUE**. The **NETID** keyword is included in both WHERE clauses to make sure that only **IINR41** transactions are selected. (Each WHERE clause is a separate set of selection criteria.)

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


ENVELOPE AND SEND command

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

For more information on determining the number of groups and envelopes needed, see “Enveloping services” on page 365.

Syntax

ENVELOPE AND SEND

ACFIELD(*starting application control field data*) TO(*ending application control field data*)

APPLID(*application ID*)

BATCH(*translated transaction batch ID*)

ENVPRBREAK(*start new envelope*)

EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)

FILEID(*processing file ddname*)

FIXEDFILEID(*fixed-to-fixed output ddname*)

FORMAT(*data format ID*)

HANDLE(*starting transaction ID*) TO(*ending transaction ID*)

IACCESS(*IEXIT access*)

IAREA(*IEXIT information*)

IEXIT(*interchange control program*)

IFCC(*override condition codes*)

INMEMTRANS(*transactions in memory*)

ITPBREAK(*new interchange envelope*)

ITYPE(*IEXIT program type*)

NETID(*network ID*)

OPTRECS(*optional record type*)

PAGE(*pageable translation*)

RAWDATA(*translate to raw data format*)

RECOVERY(*recovery unit of work*)

SAPUPDT(*track SAP status*)

SERVICESEGVAL(*service segment validation level*)

SEQNUM(*increment network profile member numbers*)

SETCC(*condition codes*)

STDTRID(*EDI standard transaction set ID*)

TPID(*trading partner ID*)

TPNICKN(*trading partner nickname*)

TRERLVL(*maximum translation error level*)

TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)

TRXTIME(*starting transaction time*) TO(*ending transaction time*)

VERIFY(*verify transaction status*)

ENVELOPE AND SEND command examples

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

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

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

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

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

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



NOTE: These transactions will be sent using different networks.

ENVELOPE DATA EXTRACT command

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

You can use this command to create report data to:

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

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

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

To extract detailed information about transactions, use the TRANSACTION DATA EXTRACT command (see page 103).

You can use wild cards with this command, as follows:

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

Syntax

ENVELOPE DATA EXTRACT

ACFIELD(*starting application control field data*) TO(*ending application control field data*)

APPLICATION(*write application data record*)

APPLID(*application ID*)

APPRECID(*application receiver department ID*)

APPSNDID(*sender's department ID*)

BATCH(*translated transaction batch ID*)

CONCATENATE(*concatenate extract data*)

DIR(*processing direction*)

DLVDATE(*starting delivery date*) TO(*ending delivery date*)

DLVTIME(*starting delivery time*) TO(*ending delivery time*)

ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*)

ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*)

ENVTYPE(*transaction envelope type*)

EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)

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

ENVELOPE DATA EXTRACT command examples

Example 1: The EDI users in Company XYZ want to view the status of their EDI documents for the last month. Specifically, the Purchasing department would like to look up purchase orders, by purchase order number, to determine the daily status. The report must contain:

- Document type (purchase order, invoice, and so on)
- Control numbers
- Date and time
- Trading partner nickname
- Reference number (purchase order number, invoice number, and so on)
- Transaction status

The programmers at Company XYZ have decided to use a ENVELOPE DATA EXTRACT command to collect the necessary information from DataInterchange. They have written a COBOL program, which they run daily, that will format the records into a customized report. Users will read this customized report, searching on the reference numbers to easily find the status they are interested in.

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

The following is an example of a customized report.

XYZ COMPANY							
TYPE	CONTROL NUMBER	SET NUMBER	DATE	TIME	TRADING NICKNAME	PO # or INVOICE #	STATUS
PO	00000000375	000000003386	03/02/01	06:02:93	BOBS	PO100332201	ENVELOPED
PO	00000000383	000000003394	03/01/01	11:12:53	BOBS4790	PO100222210	ENVELOPED
PO	00000000388	000000003399	03/02/01	11:33:06	BOBS4790	PO100232115	ENVELOPED
PO	00000000389	000000003400	03/01/01	06:52:23	BOBS4790		ENVELOPED
PO	00000000377	000000003388	03/01/01	11:41:23	BOBS	PO100123130	ENVELOPED
PO	00000000381	000000003392	03/01/01	06:22:56	BOBS4790	PO121312322	ENVELOPED
IN	00000000145	000000000383	03/02/01	12:53:42	DLGLEVEL	IN94410	RECEIVED
IN	00000000144	000000000382	03/02/01	09:12:15	DLGLEVEL	IN95443	RECEIVED
PO	00010000130	000000004853	03/02/01	09:43:02	DLGOCCUR	IN76445	ENVELOPED
PO	00010000130	000000004854	03/01/01	07:25:16	DLGOCCUR	IN76835	ENVELOPED
PO	00010000130	000000004855	03/01/01	11:51:06	DLGOCCUR	IN76337	ENVELOPED
PO	00010000130	000000004856	03/02/01	06:34:29	DLGOCCUR	IN76838	ENVELOPED

Example 2: Retrieve the interchange, group, and transaction information for all transactions in a particular envelope with trading partner name **PISCES** and interchange control number **000008888**. Place the results in the EDIQUERY file.

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

Example 3: Retrieve the interchange, group, and transaction information for all envelopes with an application sender ID of **SERVO**. Include application and image information.

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

EXPORT command

This command uses the control file to extract setup information from DataInterchange and write records to one or more files. (See “Export/Import control file (CTLFILE)” on page 187 for ddnames.) You can request a particular file record format with the **EIFORMAT** keyword. If the Export/Import file is empty, the requested format is used. If the Export/Import file is not empty, the format of the data already in the file is used.

Syntax

EXPORT

CTLFILE(*control file ddname*)

CTLTYPE(*control file type*)

EIFORMAT(*requested export file record format*)

IFCC(*override condition codes*)

NOMSG(*print extraneous messages*)

SETCC(*condition codes*)

EXPORT command example

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

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

GLB DUMP command

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

Syntax

GLB DUMP

IFCC(*override condition codes*)

LEVEL(*dump or trace level*)

RESET(*reset output file*)

SETCC(*condition codes*)

GLB DUMP command examples

Example 1: Open EDIGDMP1 for output and generate a complete persistent environment dump.

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

Example 2: Open EDIGDMP1 for append and generate a persistent environment dump of just the data area.

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

GLB TRACE command

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

Syntax

GLB TRACE

IFCC(*override condition codes*)

LEVEL(*dump or trace level*)

RESET(*reset output file*)

SETCC(*condition codes*)

GLB TRACE command examples

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

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

Example 2: End the persistent environment trace.

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


HOLD command

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

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

Syntax

HOLD

ACFIELD(*starting application control field data*) TO(*ending application control field data*)

APPLID(*application ID*)

APPRECID(*application receiver department ID*)

APPSNDID(*sender's department ID*)

BATCH(*translated transaction batch ID*)

DIR(*processing direction*)

DLVDATE(*starting delivery date*) TO(*ending delivery date*)

DLVTIME(*starting delivery time*) TO(*ending delivery time*)

ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*)

ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*)

ENVTYPE(*transaction envelope type*)

EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)

FORMAT(*data format ID*)

FUNACKP(*pending functional acknowledgment*)

GRPCTLNO(*starting sender's group control number*) TO(*ending sender's group control number*)

HANDLE(*starting transaction ID*) TO(*ending transaction ID*)

IFCC(*override condition codes*)

INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*)

INTRECID(*interchange receiver ID*)

INTSNDID(*interchange sender ID*)

NETACKP(*pending network acknowledgment*)

NETID(*network ID*)

NETSTAT(*network transaction status*)

SETCC(*condition codes*)

SNDDATE(*starting request sent date*) TO(*ending request sent date*)

SNDTIME(*starting request sent time*) TO(*ending request sent time*)

STDTRID(*EDI standard transaction set ID*)

STSTAT(*transaction status*)

TPID(*trading partner ID*)

TPNICKN(*trading partner nickname*)

TRERLVL(*maximum translation error level*)

TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)

TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)

TRXTIME(*starting transaction time*) TO(*ending transaction time*)

HOLD command example

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

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

IMPORT command

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

Syntax

IMPORT

ACTUSAGE(*activate imported usage*)

CTLFILE(*control file ddname*)

CTLTYPE(*control file type*)

DUPCHECK(*perform duplicate checks*)

IFCC(*override condition codes*)

NOMSG(*print extraneous messages*)

SETCC(*condition codes*)

WRTCTLNO(*import trading partner control number*)

IMPORT command example

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

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

LOAD LOG ENTRIES command

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

Syntax

LOAD LOG ENTRIES

APPLID(*application ID*)

HOLDFILE(*event log hold file name*)

HOLDTYPE(*event log hold file type*)

IFCC(*override condition codes*)

LOGAEID(*starting event log associated entry ID*) **TO**(*ending event log associated entry ID*)

LOGDATE(*starting event log date*) **TO**(*ending event log date*)

LOGFORM(*starting event log format ID*) **TO**(*ending event log format ID*)

LOGTIME(*starting event log time*) **TO**(*ending event log time*)

LOGUSER(*starting event log user ID*) **TO**(*ending event log user ID*)

NEWAPPLID(*new application ID*)

SETCC(*condition codes*)

Load log entries command example

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

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

NETWORK ACTIVITY DATA EXTRACT command

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

You can use this command to create reports to:

- Reconcile network charges
- List network usage by user ID
- List heaviest network users
- List inactive requestors

You can use wild cards with this command, as follows:

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

Syntax

NETWORK ACTIVITY DATA EXTRACT

ACCTID(*network account ID*)

DAYS(*starting date*) TO(*ending date*)

DYNSQL(*use dynamic SQL*)

IFCC(*override condition codes*)

NETID(*network ID*)

NETNAME(*network name*)

REQID(*requestor ID*)

SETCC(*condition codes*)

USERID(*network user ID*)

USERPGM(*user program*)

NETWORK ACTIVITY DATA EXTRACT command examples

Example 1: Your network charges jumped significantly last month and you want to understand which of your applications has increased its network usage. To do this, you need a report that shows the:

- Account ID
- User ID
- Number of envelopes exchanged, both sent and received
- Number of characters exchanged, both sent and received

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

The data resulting from this command was written to file EDIQUERY, sorted by a local sort utility, and then used as input to a user-written program that created the following report.

Network Traffic Activity Report							Date: 01/09/20
							Time: 12:31:02
Interchange Month	Total Network Name	Net ID	Account ID	User ID	S/R	Envelopes	Characters
08/01	AT&T Global Network	IINR41	XTEV	PURCHASE	S	7776	9998376
					R	4356	2333157
				ACCTPAYB	S	1526	8867837
					R	662	333457
						14320	21532827
			XTEW	RECEIVNG	S	856	82251
					R	2856	434457
				TRAFFIC	R	770	66615
						4482	583323
		IINCICS	XTEX	PURCHASE	S	4476	2648376
					R	756	453457
				PRODPO	S	6625	7838902
					R	559	467457
						12416	11408192

Example 2: Extract information to reconcile your EDI traffic through a particular account (**CMPY**) and user ID (**USER21**) on the network for the month of May during 2001. Include a list of the requestors that use that account/user ID on the network, and the number of messages sent and received by each requestor (only the traffic through DataInterchange for this account and user ID).

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

PRINT ACKNOWLEDGMENT IMAGE command

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

Syntax

PRINT ACKNOWLEDGMENT IMAGE

ACFIELD(*starting application control field data*) TO(*ending application control field data*)

APPLID(*application ID*)

APPRECID(*application receiver department ID*)

APPSNDID(*sender's department ID*)

BATCH(*translated transaction batch ID*)

DIR(*processing direction*)

DLVDATE(*starting delivery date*) TO(*ending delivery date*)

DLVTIME(*starting delivery time*) TO(*ending delivery time*)

ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*)

ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*)

ENVTYPE(*transaction envelope type*)

EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)

FORMAT(*data format ID*)

FUNACKP(*pending functional acknowledgment*)

GRPCTLNO(*starting sender's group control number*) TO(*ending sender's group control number*)

HANDLE(*starting transaction ID*) TO(*ending transaction ID*)

IFCC(*override condition codes*)

INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*)

INTRECID(*interchange receiver ID*)

INTSNDID(*interchange sender ID*)

MERGED(*merge transaction image with functional acknowledgment*)

NETACKP(*pending network acknowledgment*)

NETID(*network ID*)

NETSTAT(*network transaction status*)

SEGMENTED(*print segment on new line*)

SETCC(*condition codes*)

SNDDATE(*starting request sent date*) TO(*ending request sent date*)

SNDTIME(*starting request sent time*) TO(*ending request sent time*)

STDTRID(*EDI standard transaction set ID*)

STSTAT(*transaction status*)

TPID(*trading partner ID*)

TPNICKN(*trading partner nickname*)

TRERLVL(*maximum translation error level*)

TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)

TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)

TRXTIME(*starting transaction time*) TO(*ending transaction time*)

PRINT ACKNOWLEDGMENT IMAGE command example

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

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

The following is an example of the Functional Acknowledgment Image report.

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

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

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


PRINT ACTIVITY SUMMARY command

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

Syntax

PRINT ACTIVITY SUMMARY

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

PRINT ACTIVITY SUMMARY command example

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

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



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

The following is an example of the Activity Summary Report.

TF11	Activity Summary Report	Date: 01/12/14
		Time: 12:12:12
Outbound Transactions		Count
Selected transactions :		1940
Translation :		1940
Acceptably translated . . :		1930
Unacceptably translated . :		10
Enveloping :		1000
Enveloped :		900
Enveloping errors :		100
Send requests :		900
Sent :		800
Pending functional ack :		10
Pending network ack . . :		10
Not sent :		100
Error on request to send:		73
Network error :		27
Detached :		0
Inbound Transactions		Count
Selected transactions :		1000
Acceptably translated . . . :		800
Unacceptably translated . . :		10
Not yet translated :		190
Detached :		0

PRINT EVENT LOG command

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

Syntax

PRINT EVENT LOG

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

PRINT STATUS SUMMARY command

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

Status summaries are not available for online viewing.

Syntax

PRINT STATUS SUMMARY

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

PRINT STATUS SUMMARY command example

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

```
PERFORM PRINT STATUS SUMMARY
```

```
WHERE TRXDATE(01/12/14)
```

The following is an example of the Status Summary Report for Outbound Transactions. For more information on the fields included in this report, refer to the *DataInterchange Administrator's Guide*.

TF80	Status Summary Report for Outbound Transactions				Date: 01/12/14 Time: 12:12:12
Transaction Handle Date Enveloped	Trading Partner Nickname Interchange Cntrl No	Data Format ID Network Status	Transaction Status Group Control No	Store Status Func Ack Status	
200112140930120000 01/12/14	Partner111111111 12121212121212	Format111111111 Accepted by network	Transaction accepted 11111111111111	Purge Requested Received	
200112121045330001	Partner111111111	Format333333333	Send translate error	Active	
200112141033110000 01/12/14	Partner222222222 22222222222222	Format222222222 Accepted by network	Transaction accepted 11111111111111	Active Received	
200112141033110004 R 01/12/14	Partner222222222 22222222222222	Format333333333 Accepted by network	Transaction accepted 22222222222222	Active Received	
200112141033110003 01/12/14 01/12/14	Partner333333333 33333333333333 33333333333333	Format111111111 Recall request error Not sent - net error	Transaction accepted 11111111111111 22222222222222	Purge Requested Received	

The same report format is used for Status Summary reports for Inbound Transactions.

TF80	Status Summary Report for Inbound Transactions				Date: 01/12/14 Time: 12:12:12
Transaction Handle Date Translated	Trading Partner Nickname Data Format ID	Standard Trans ID Translation Status	Transaction Status	Store Status	
200112141344270000 01/12/14	Partner222222222 Format1212121212	12121212 Acceptable	Receive translated	Active	
200112141010100001 01/12/14 01/12/14	Partner222222222 Format3333333333 Format3333333333	12345678 Acceptable Unacceptable	Receive translated	Active	

PRINT STATUS SUMMARY2 command

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

Status summaries are not available for online viewing.

Syntax

PRINT STATUS SUMMARY2

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

PRINT STATUS SUMMARY2 command example

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

```
PERFORM PRINT STATUS SUMMARY2
```

```
WHERE TRXSTAT(73)
```

The following is an example of the Selection Criteria for Status Summary Report.

1	Selection Criteria for Status Summary Report				Date: 01/12/14 Time: 14:25:44
SELECTION CRITERIA		1			
Transaction handle. : 20010323000000000000 TO 20010323240000000000					
1TF80	Status Summary Report for Outbound Transactions				Date: 01/12/14 Time: 14:25:44
Transaction Handle Trading Partner Nickname Data Format ID Transaction Status Store Status					
Application Control Number			Internal Trading Partner ID		
Date Enveloped		Interchange Cntrl No	Network Status	Group Control No	Func Ack Status
200103230921579000		BOBS	BOBS8004833	ENVELOPED	ACTIVE
			1111122222		
01/12/14	000000000000418	ENVELOPED	000000000000399	PENDING	
20011214116562000		BOBS	BOBSPORDER	ENVELOPED	ACTIVE
BOBS 12345678		BOBS			
01/12/14	000000000000419	ENVELOPED	000000000000400	NOT REQUESTED	

PRINT TRANSACTION DETAILS command

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

Syntax

PRINT TRANSACTION DETAILS

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

PRINT TRANSACTION DETAILS command example

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

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

The following is an example of the Transaction Details report. For more information on the fields included in this report, refer to the *DataInterchange Administrator's Guide*.

TF73	Transaction Details	Date: 01/12/14 Time: 12:12:12
Transaction handle	: 20011214101533000001	
Trading partner nickname	: PISCES	
Internal trading partner ID	: 2345678901234567890123456789012345	
Direction	: SEND	
Data format ID	: POSEND	
Application control number	: 12345678901234567890123456789012345	
Interchange control number	: 12345678901234	
Group control number	: 12345678901234	
Transaction control number	: 12345678901234	
Transaction status	: Translated	
Added to store	: 01/12/14-10:29:48	
Store status	: Active	
Delivered to application	:	
Translation error level	: 1	
Translation	: Acceptable	
Network acknowledgment requested	: D	
Network status	:	
Functional acknowledgment	:	
Functional ack date	:	
Application ID	: EDIMP	
Batch ID	: BATCH123	
Earliest purge date	: 01/12/14	
Earliest envelope date	: 01/12/14	
Enveloped/deenveloped	:	
Sent	:	
Envelope profile member	: X12V2R2	
Standard ID	: X12V2R2	
Standard version	: V2	
Standard level	: R2	
Standard transaction ID	: 850	
Network ID	: IN	
Usage indicator	: P	
Total segment count	: 22	
Transaction size (bytes)	: 742	

PRINT TRANSACTION IMAGE command

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

Syntax

PRINT TRANSACTION IMAGE

ACFIELD(*starting application control field data*) TO(*ending application control field data*)

APPLID(*application ID*)

APPRECID(*application receiver department ID*)

APPSNDID(*sender's department ID*)

BATCH(*translated transaction batch ID*)

DIR(*processing direction*)

DLVDATE(*starting delivery date*) TO(*ending delivery date*)

DLVTIME(*starting delivery time*) TO(*ending delivery time*)

ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*)

ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*)

ENVTYPE(*transaction envelope type*)

EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)

FORMAT(*data format ID*)

FUNACKP(*pending functional acknowledgment*)

GRPCTLNO(*starting sender's group control number*) TO(*ending sender's group control number*)

HANDLE(*starting transaction ID*) TO(*ending transaction ID*)

IFCC(*override condition codes*)

INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*)

INTRECID(*interchange receiver ID*)

INTSNDID(*interchange sender ID*)

MERGED(*merge transaction image with functional acknowledgment*)

NETACKP(*pending network acknowledgment*)

NETID(*network ID*)

NETSTAT(*network transaction status*)

SEGMENTED(*print segment on new line*)

SETCC(*condition codes*)

SNDDATE(*starting request sent date*) TO(*ending request sent date*)

SNDTIME(*starting request sent time*) TO(*ending request sent time*)

STDTRID(*EDI standard transaction set ID*)

STSTAT(*transaction status*)

TPID(*trading partner ID*)

TPNICKN(*trading partner nickname*)

TRERLVL(*maximum translation error level*)

TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)

TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)

TRXTIME(*starting transaction time*) TO(*ending transaction time*)

PRINT TRANSACTION IMAGE command example

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

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

The following is an example of the Transaction Image report.

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

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

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

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

PROCESS NETWORK ACKS command

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

Syntax

PROCESS NETWORK ACKS

ACKFILE(*network acknowledgment ddname*)

ACKTYPE(*acknowledgment file type*)

PROCESS NETWORK ACKS command example

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

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

PURGE command

This command marks a transaction for purging from the Transaction Store but does not remove it. Transactions are marked for purging when old outbound transactions are still pending functional reconciliation. Transactions marked for purge are deleted from the store when you execute the REMOVE TRANSACTIONS command. For more information, see “REMOVE TRANSACTIONS command” on page 83. If the transaction is one of a related group, all transactions in the group are marked for purging. Only transactions with status of PURGE-USER REQUEST or PURGE-DATE EXPIRED are eligible for purging. Reconciled transactions are automatically marked PURGE-DATE EXPIRED after the purge internal has expired.

The default expiration date is 30 days from translation. You can change the default by specifying a PURGINT value on PERFORM statements that add transactions to the Transaction Store (for example, TRANSLATE AND ENVELOPE). See “PURGINT” on page 152 for applicable PERFORM commands.

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

Syntax

PURGE

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

TPNICKN(*trading partner nickname*)

TRERLVL(*maximum translation error level*)

TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)

TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)

TRXTIME(*starting transaction time*) TO(*ending transaction time*)

PURGE command example

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

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

QUERY command

This command generates a list of transactions that meet the selection criteria you specify in the keywords. In the returned list, transactions are identified by their transaction handle (**Transaction Store ID**). DataInterchange writes the list of handles to the specified file (ddname EDIQUERY for MVS). Each item in the list is 30 bytes long and has the following format:

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

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



NOTE: In CICS, you can pass the file name and type to DataInterchange as parameters.

Syntax

QUERY

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

TPNICKN(*trading partner nickname*)

TRERLVL(*maximum translation error level*)

TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)

TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)

TRXTIME(*starting transaction time*) TO(*ending transaction time*)

QUERY command example

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

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

QUERY PROFILE command

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

Syntax

QUERY PROFILE

ID(*DataInterchange profile name*)

IFCC(*override condition codes*)

MEMBER(*member profile name*)

OUTFILE(*output file name*)

OUTFORMAT(*output format*)

OUTTYPE(*output file type*)

SETCC(*condition codes*)

QUERY PROFILE command examples

Example 1: Query all trading partner profile members and write results to **MYFILE** in fixed format.

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

Example 2: Query mailbox (requestor) profile member **REQMEM** and write results to **MYFILE** in the original format.

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

RECEIVE command

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

Syntax

RECEIVE

CLEARFILE(*clear specified file contents*)

ENVTYPE(*transaction envelope type*)

FILEID(*processing file ddname*)

IFCC(*override condition codes*)

MSGUCLASS(*override message user class*)

ONEMSG(*read only one MQ message*)

REQID (*requestor ID*)

SETCC(*condition codes*)

TPNICKN(*trading partner nickname*)

RECEIVE command examples

Example 1: Receive data for requestors defined in mailbox (requestor) profile members **DEPTA7F** and **DEPTB8R**. Place the data into the receive files specified by the profile members. Clear both files before the data is received. The data is in X12 format.

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

Example 2: In CICS, receive data for the requestor defined by mailbox (requestor) profile member **DEPTA7F**. Place the data in CICS temporary storage queue **RECVPO**. The data is in EDIFACT format.

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

RECEIVE AND DEENVELOPE command

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

Syntax

RECEIVE AND DEENVELOPE

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

RECEIVE AND DEENVELOPE command examples

Example 1: Receive all EDI documents for requestor **DEPTA7F** and deenvelope them. Place them in the Transaction Store. The documents are in X12 format.

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

Example 2: Receive all EDI documents for requestor **DEPTA7F**, deenvelope them. Place them in the Transaction Store. Set the purge interval for these documents at 20 days. The documents are in EDIFACT format.

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

RECEIVE AND SEND command

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

Syntax

RECEIVE AND SEND

APPFILE(*application data file name*)

CLEARFILE(*clear specified file contents*)

ENVTYPE(*transaction envelope type*)

FILEID(*processing file ddname*)

IFCC(*override condition codes*)

MSGUCLASS(*override message user class*)

RAWDATA(*translate to raw data format*)

REQID(*requestor ID*)

SAPUPDT(*track SAP status*)

SCRIPT(*script ID*)

SEQNUM(*increment network profile member numbers*)

SETCC(*condition codes*)

TPNICKN(*trading partner nickname*)

RECEIVE AND SEND command example

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

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

RECEIVE AND TRANSLATE command

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



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

Syntax

RECEIVE AND TRANSLATE

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

RECEIVE AND TRANSLATE command examples

Example 1: Receive and translate all EDI documents for requestor **DEPTA7F**. Along with the transaction data, return all optional records produced during deenveloping and translation. The documents are in X12 format.

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

Example 2: Receive and translate all EDI documents for requestor **DEPTA7F**. Return the transaction data in raw data format. The documents are in EDIFACT format.

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


RECONSTRUCT command

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

Syntax

RECONSTRUCT

DIR(*processing direction*)

FILEID(*processing file ddname*)

IFCC(*override condition codes*)

INTCTLNO(*starting sender's interchange control nbr*) **TO**(*ending sender's interchange control nbr*)

INTRECID(*interchange receiver ID*)

PAGE(*pageable translation*)

RAWDATA(*translate to raw data format*)

SETCC(*condition codes*)

TPNICKN(*trading partner nickname*)

RECONSTRUCT command example

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

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

RECONSTRUCT AND SEND command

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

Syntax

RECONSTRUCT AND SEND

CLEARFILE(*clear specified file contents*)

DIR(*processing direction*)

ENVTYPE(*transaction envelope type*)

FILEID(*processing file ddname*)

IFCC(*override condition codes*)

INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*)

INTRECID(*interchange receiver ID*)

MSGUCLASS(*override message user class*)

PAGE(*pageable translation*)

RAWDATA(*translate to raw data format*)

REQID(*requestor ID*)

SETCC(*condition codes*)

TPNICKN(*trading partner nickname*)

RECONSTRUCT AND SEND command example

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

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

RCVFILE command

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

Syntax

RCVFILE

CLEARFILE(*clear specified file contents*)

FILEID(*processing file ddname*)

IFCC(*override condition codes*)

MSGUCLASS(*override message user class*)

REQID(*requestor ID*)

SETCC(*condition codes*)

TPNICKN(*trading partner nickname*)

RCVFILE command example

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

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

RCVFILE AND SEND command

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

Syntax

RCVFILE AND SEND

CLEARFILE(*clear specified file contents*)

ENVTYPE(*transaction envelope type*)

FILEID(*processing file ddname*)

IFCC(*override condition codes*)

MSGUCLASS(*override message user class*)

REQID(*requestor ID*)

SAPUPDT(*track SAP status*)

SCRIPT(*script ID*)

SETCC(*condition codes*)

TPNICKN(*trading partner nickname*)

RCVFILE AND SEND command example

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

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

REENVELOPE command

This command takes the EDI transactions from the Transaction Store that were previously enveloped, envelopes them again, and places the results in an envelope file. The envelope file is either the TD queue specified in the network profile member or a file specified in the command. Selection criteria determine which transactions go into the interchange envelope. The enveloper sorts the transactions to create the fewest number of functional groups and interchange envelopes. For more information about envelope processing, see “Enveloping services” on page 365.

Syntax

REENVELOPE

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

SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
STDTRID(*EDI standard transaction set ID*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)
VERIFY(*verify transaction status*)

REENVELOPE command example

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

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

REENVELOPE AND SEND command

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

Syntax

REENVELOPE AND SEND

ACFIELD(*starting application control field data*) TO(*ending application control field data*)
 APPLID(*application ID*)
 APPRECID(*application receiver department ID*)
 APPSNDID(*sender's department ID*)
 BATCH(*translated transaction batch ID*)
 CLEARFILE(*clear specified file contents*)
 DIERRFILTER(*initial error filter set*)
 ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*)
 ENVPRBREAK(*start new envelope*)
 ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*)
 ENVTYPER(*transaction envelope type*)
 EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)
 FILEID(*processing file ddname*)
 FIXEDFILEID(*fixed-to-fixed output ddname*)
 FORMAT(*data format ID*)
 FUNACKP(*pending functional acknowledgment*)
 GRPCTLNO(*starting sender's group control number*) TO(*ending sender's group control number*)
 HANDLE(*starting transaction ID*) TO(*ending transaction ID*)
 IACCESS(*IEXIT access*)
 IAREA(*IEXIT information*)
 IEXIT(*interchange control program*)
 IFCC(*override condition codes*)
 INMEMTRANS(*transactions in memory*)
 INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*)
 INTRECID(*interchange receiver ID*)
 INTSNDID(*interchange sender ID*)
 ITPBREAK(*new interchange envelope*)
 ITYPE(*IEXIT program type*)
 MSGUCLASS(*override message user class*)
 NETACKP(*pending network acknowledgment*)
 NETID(*network ID*)
 NETSTAT(*network transaction status*)
 OPTRECS(*optional record type*)
 PAGE(*pageable translation*)
 RAWDATA(*translate to raw data format*)
 RECOVERY(*recovery unit of work*)
 REQID(*requestor ID*)
 SAPUPDT(*track SAP status*)
 SCRIPT(*script ID*)
 SNDDATE(*starting request sent date*) TO(*ending request sent date*)

SNDTIME(*starting request sent time*) TO(*ending request sent time*)
SERVICESEGVAL(*service segment validation level*)
SETCC(*condition codes*)
STDTRID(*EDI standard transaction set ID*)
TPID(*trading partner ID*)
TPNICKN(*trading partner nickname*)
TPNICKNESEND(*trading partner profile member*)
TRERLVL(*maximum translation error level*)
TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)
TRXDATE(*starting transaction date*) TO(*ending transaction date*)
TRXSTAT(*transaction processing status*)
TRXTIME(*starting transaction time*) TO(*ending transaction time*)
VERIFY(*verify transaction status*)

REENVELOPE AND SEND command example

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

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


RELEASE command

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

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

Syntax

RELEASE

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

RELEASE command example

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

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

REMOVE LOG ENTRIES command

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

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

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

Syntax

REMOVE LOG ENTRIES

APPLID(*application ID*)

ARCHIVEFILE(*event log archive file name*)

ARCHIVETYPE(*event log archive file type*)

HOLDFILE(*event log hold file name*)

HOLDTYPE(*event log hold file type*)

IFCC(*override condition codes*)

LOGAEID(*starting event log associated entry ID*) TO(*ending event log associated entry ID*)

LOGDATE(*starting event log date*) TO(*ending event log date*)

LOGFORM(*starting event log format ID*) TO(*ending event log format ID*)

LOGTIME(*starting event log time*) TO(*ending event log time*)

LOGUSER(*starting event log user ID*) TO(*ending event log user ID*)

NUMDELS(*number of database deletes before commit*)

SETCC(*condition codes*)

REMOVE LOG ENTRIES command example

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

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

REMOVE STATISTICS command

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

Syntax

REMOVE STATISTICS

IFCC(*override condition codes*)

NUMDELS(*number of database deletes before commit*)

PRIORTO(*deletion end date*)

SETCC(*condition codes*)

REMOVE STATISTICS command example

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

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



NOTES:

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

REMOVE TRANSACTIONS command

This command deletes transactions from the Transaction Store that have a status of either:

- PURGE-STORE TIME EXPIRED
- PURGE-USER REQUEST

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

The default expiration date is 30 days from translation. You can change the default by specifying a PURGINT value on PERFORM statements which add transactions to the Transaction Store (for example, TRANSLATE AND ENVELOPE). See “PURGINT” on page 152 for applicable PERFORM commands.



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

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

After purging documents from the Transaction Store, you can use the Archive action to remove the event logs associated with these transactions. For more information, refer to the *DataInterchange Administrator's Guide*.

This command does not delete ACTIVE or HELD transactions.

- To make ACTIVE transactions eligible for removal, they must first be marked for purge.
- To make HELD transactions eligible for removal, they must first be released from held status.

Syntax

REMOVE TRANSACTIONS

ACFIELD(*starting application control field data*) TO(*ending application control field data*)

APPLID(*application ID*)

APPRECID(*application receiver department ID*)

APPSNDID(*sender's department ID*)

BATCH(*translated transaction batch ID*)

DIR(*processing direction*)

DLVDATE(*starting delivery date*) TO(*ending delivery date*)

DLVTIME(*starting delivery time*) TO(*ending delivery time*)

ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*)

ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*)

ENVTYPE(*transaction envelope type*)

EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)

FORMAT(*data format ID*)

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

REMOVE TRANSACTIONS command example

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

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

REPORT CONTINUOUS RECEIVE STATUS command

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

Syntax

REPORT CONTINUOUS RECEIVE STATUS

IFCC(*override condition codes*)

MEMBER(*member profile name*)

SETCC(*condition codes*)

REPORT CONTINUOUS RECEIVE STATUS command examples

Example 1: Report all continuous receives.

```
PERFORM REPORT CONTINUOUS RECEIVE STATUS
```

Example 2: Report the status of continuous receive profile member CRMEM.

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

RESET STATISTICS command

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

Syntax

RESET STATISTICS

IFCC(*override condition codes*)

NUMDELS(*number of database deletes before commit*)

PRIORTO(*deletion end date*)

SETCC(*condition codes*)

RESET STATISTICS command example

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

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


RESTART RECEIVE command

This command is used when receiving EDI data, and is valid only when:

- the network supports restart, and
- you specify checkpoint-level recovery when initially receiving the data

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



NOTE:

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

Syntax

RESTART RECEIVE

REQID(requestor ID)

RESTART RECEIVE command example

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

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

RESTART SEND command

This command is used when sending EDI data, and is valid only when

- the network supports restart, and
- you specified checkpoint-level recovery when initially sending the data

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



NOTES:

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

Syntax

RESTART SEND

CLEARFILE(*clear specified file contents*)

ENVTYPE(*transaction envelope type*)

FILEID(*processing file ddname*)

IFCC(*override condition codes*)

MSGUCLASS(*override message user class*)

REQID(*requestor ID*)

SAPUPDT(*track SAP status*)

SCRIPT(*script ID*)

SEQNUM(*increment network profile member numbers*)

SETCC(*condition codes*)

TPNICKN(*trading partner nickname*)

RESTART SEND command example

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

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

RETRANSLATE TO APPLICATION command

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

Syntax

RETRANSLATE TO APPLICATION

ACFIELD(*starting application control field data*) TO(*ending application control field data*)

APPLID(*application ID*)

APPRECID(*application receiver department ID*)

APPSNDID(*sender's department ID*)

ASSERTLVL(*session assertion level*)

BATCH(*translated transaction batch ID*)

BATCHSET(*set transaction batch ID*)

CCEXCEPTION(*job-step condition code*)

DIERRFILTER(*initial error filter set*)

DLVDATE(*starting delivery date*) TO(*ending delivery date*)

DLVTIME(*starting delivery time*) TO(*ending delivery time*)

ENVTYPE(*transaction envelope type*)

EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)

EXTENDC(*translate with extended C record format*)

FORCETEST(*force test usage*)

FORMAT(*data format ID*)

GRPCTLNO(*starting sender's group control number*) TO(*ending sender's group control number*)

HANDLE(*starting transaction ID*) TO(*ending transaction ID*)

IFCC(*override condition codes*)

INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*)

INTRECID(*interchange receiver ID*)

INTSNDID(*interchange sender ID*)

NETID(*network ID*)

OPTRECS(*optional record type*)

PAGE(*pageable translation*)

RAWDATA(*translate to raw data format*)

SETCC(*condition codes*)

STDTRID(*EDI standard transaction set ID*)

TPID(*trading partner ID*)

TPNICKN(*trading partner nickname*)

TRERLVL(*maximum translation error level*)

TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)

TRXDATE(*starting transaction date*) TO(*ending transaction date*)

TRXSTAT(*transaction processing status*)

TRXTIME(*starting transaction time*) TO(*ending transaction time*)

RETRANSLATE TO APPLICATION command example

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

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

SAP STATUS EXTRACT command

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

Syntax

SAP STATUS EXTRACT

OUTFILE(*output file name*)

OUTTYPE(*output file type*)

CLIENT(*SAP client ID*)

SAPSTAT(*SAP status starting value*) **TO**(*SAP status ending value*)

DAYS(*record starting date*) **TO**(*record ending date*)

SAP STATUS EXTRACT command example

Extract SAP status records from the DataInterchange database and write them to the file specified in the keywords **OUTFILE** and **OUTTYPE**.

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

SAP STATUS REMOVE command

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

Syntax

SAP STATUS REMOVE

CLIENT(*SAP client ID*)

PRIORTO(*Date before which to remove records*)

SAPSTAT(*SAP status starting value*) TO(*SAP status ending value*)

DAYS(*Record starting date*) TO(*Record ending date*)

SAP STATUS REMOVE command example

Remove SAP status information from the DataInterchange database for all clients with status **16**.

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

SEND command

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

Syntax

SEND

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

SEND command examples

Example 1: Send the transactions in the TD queue. Use the network specified by mailbox (requestor) profile member **IINREQ**.

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

Example 2: Send the EDIFACT transactions in the file **NETQUEUE**. Use the network specified by mailbox (requestor) profile member **IINREQ**.

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

SENDFILE command

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

Syntax

SENDFILE

CLEARFILE(*clear specified file contents*)

FILEID(*processing file ddname*)

IFCC(*override condition codes*)

MSGUCLASS(*override message user class*)

REQID(*requestor ID*)

SCRIPT(*script ID*)

SEQNUM(*increment network profile member numbers*)

SETCC(*condition codes*)

TPNICKN(*trading partner nickname*)

SENDFILE command examples

Example 1: Send the data in the file named **FLATFILE** to trading partner **MYTP**.

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

Example 2: Send the data in the file named **BULK** to trading partner **YOURTP** with a network message user class of **SPECIAL**.

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


START CONTINUOUS RECEIVE command

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

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

Syntax

START CONTINUOUS RECEIVE

IFCC(*override condition codes*)

MEMBER(*member profile name*)

SETCC(*condition codes*)

START CONTINUOUS RECEIVE command examples

Example 1: Start all eligible continuous receives.

```
PERFORM START CONTINUOUS RECEIVE
```

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

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

STOP CONTINUOUS RECEIVE command

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

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

Syntax

STOP CONTINUOUS RECEIVE

IFCC(*override condition codes*)

MEMBER(*member profile name*)

SETCC(*condition codes*)

STOP CONTINUOUS RECEIVE command examples

Example 1: Stop all active continuous receives.

```
PERFORM STOP CONTINUOUS RECEIVE
```

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

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

TRADING PARTNER CAPABILITY DATA EXTRACT command

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

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

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

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



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

Syntax

TRADING PARTNER CAPABILITY DATA EXTRACT

ADDRNL1(*trading partner address line 1*)
 ADDRNL2(*trading partner address line 2*)
 CMMTLN1(*trading partner comment line 1*)
 CMMTLN2(*trading partner comment line 2*)
 CMPYNM(*trading partner company name*)
 DAYS(*starting date*) TO(*ending date*)
 DIR(*processing direction*)
 DYNSQL(*use dynamic SQL*)
 IFCC(*override condition codes*)
 MAPID(*map name*)
 SETCC(*condition codes*)
 STDDESC(*EDI standard description*)
 STDID(*EDI standard ID*)
 STDLV(*EDI standard release level*)
 STDTRID(*EDI standard transaction set ID*)
 STDVR(*EDI standard version*)
 TESTMODE(*test transaction mode*)
 TPID(*trading partner ID*)
 TPNICKN(*trading partner nickname*)
 USERPGM(*user program*)

TRADING PARTNER CAPABILITY DATA EXTRACT command examples

Example 1: Company X is implementing EDI with all suppliers to their Atlanta assembly plant. They have set a target date of 12/31 to have 100% EDI on several transactions. Suppliers start out in test mode for each new transaction and, after 100 successful test transactions, are eligible to move into production. The EDI steering committee meets monthly to review progress and needs a report that shows information for each supplier, such as:

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

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

The following is an example of a trading partner capability report.

Trading Partner Capability Report Data												
Company NameID	Location	Int ID	Trx	Std.	ID	Dir	ID	Started Testing	Test Trx's	Started Productn	Product Trx's	
Global Services	Tampa, FL	233119	850	X12V2R1		S	NONPRODX12V2R1PO	07/03/01	101	08/15/01	262	
Only1 Co.	Lima, OH	399129	850	X12V2R3		S	PRODSUPX12V2R3PO	08/15/01	71	*NODATE*	0	
Pulse Services	Armonk, NY	997223	850	X12V2R3		S	PRODSUPX12V2R3PO	*NODATE*	0	*NODATE*	0	
Handlebars Etc.	Raleigh, NC	877519	850	X12V2R3		S	PRODSUPX12V2R3PO	05/07/01	115	07/02/01	554	
				855	X12V2R3		R	PRODSUPX12V2R3PA	05/22/01	109	07/26/01	445
				856	X12V2R3		R	PRODSUPX12V2R3SN	06/13/01	187	07/22/01	1878
				861	X12V2R3		S	PRODSUPX12V2R3RA	08/08/01	242	*NODATE*	0
IBM	Dallas, TX	660599	850	X12V2R3		S	PRODSUPX12V2R3PO	01/08/01	126	03/14/01	927	
				855	X12V2R3		R	PRODSUPX12V2R3PA	08/22/01	054	*NODATE*	0
				856	X12V2R3		R	PRODSUPX12V2R3SN	04/06/01	378	*NODATE*	0
Pen&Paper	Carson, CA	523567	850	X12V2R3		S	PRODSUPX12V2R3PO	02/02/01	113	03/02/01	856	
				855	X12V2R3		R	PRODSUPX12V2R3PA	03/26/01	104	03/12/01	535
				856	X12V2R3		R	PRODSUPX12V2R3SN	03/13/01	127	11/23/01	278
				861	X12V2R3		S	PRODSUPX12V2R3RA	04/08/01	102	06/08/01	322

Example 2: You want to extract information associated with migrating the purchase order transaction from X12V2R1 to X12V3R1. Create a list of all the trading partners and usages or rules you must migrate to the new EDI standard.

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

TRADING PARTNER PROFILE DATA EXTRACT command

This command provides detailed information about your trading partners including:

- Company name
- Address
- Contacts
- Telephone number
- Nickname
- Network name
- Interchange ID
- Account ID
- User ID
- Latest data transmission date, sorted by trading partner

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

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

- Simple trading partner lists
- Comprehensive trading partner lists
- Trading partners sorted by division
- Inactive trading partners (no activity for x period of time)

Syntax

TRADING PARTNER PROFILE DATA EXTRACT

ACCTID(*network account ID*)

ADDRLN1(*trading partner address line 1*)

ADDRLN2(*trading partner address line 2*)

CMMTLN1(*trading partner comment line 1*)

CMMTLN2(*trading partner comment line 2*)

CMPYNM(*trading partner company name*)

CNTCTNM(*trading partner contact name*)

CNTCTPH(*trading partner contact phone number*)

DYNSQL(*use dynamic SQL*)

GRPCTLNO(*starting sender's group control nbr.*) TO(*ending sender's group control nbr.*)

IFCC(*override condition codes*)

INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*)

INTID(*trading partner interchange ID*)

LASTTRXDATE(*starting latest transaction date*) TO(*ending latest transaction date*)

NETID(*network ID*)

SETCC(*condition codes*)

TPNICKN(*trading partner nickname*)

TRXCTLNO(*starting transaction set control number*) TO(*ending transaction set control number*)

USERID(*network user ID*)

USERPGM(*user program*)

TRADING PARTNER PROFILE DATA EXTRACT command examples

Example 1: Your EDI Administrator has requested that you provide a hard copy report every Monday morning that shows the trading partners who have sent data within the last six months to the Metal Finishing and Metal Shipping Divisions. This report is used to interface with the various departments and to contact the trading partner in case of trouble. The report contains:

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

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

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

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

The following is an example of a trading partner profile report.

Trading Partner Information Report Data									
Company Name	Location	IntID	Net	Acct	UserID	Contact Name	Contact Phone	LastTrx Date	Ctrl No.
A0 Corp	Morris, MN	333219	NETA	MMIDOO	55532214	Sam Heusen	813-555-5668	08/01/01	00000171
IBM	Tampa, FL	200762	NETA	ADV001	99665593	Deb Garrie	762-555-2328	08/15/01	00990171
Alloy Inc.	Chicago, IL	543189	NETA	ALUM00	18654771	Lou Green	619-555-7784	08/15/00	00044573
DeKant Systems	Queens, NY	337690	IIN	DECANT	USER01	Bob Kling	252-555-9006	11/06/01	00670170
	London, UK	856690	IIN	DECAHH	USER02	Joe Miller	413-555-5555	03/17/01	00006392
	London, UK	009210	IIN	DECA00	USER03	Heather Liu	413-555-2256	03/27/01	00066392
IBM	Paris, FR	665767	IIN	IBMIM	01NYC	Mary Lockler	919-555-8585	02/22/01	00601054
	Raleigh, NC	976554	IIN	IB	SANJOSE	Jane Hart	878-555-2876	02/06/01	00070378
MMM Motors	Edison, NJ	332323	NETA	MMM01	MMADMIN	Jon Murphy	828-555-9058	03/18/01	05567836
	Miami, FL	003292	IIN	MMM02	MMTECH	Pauline Gold	813-555-8421	01/22/01	00650054

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

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

TRANSACTION ACTIVITY DATA EXTRACT command

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

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

You can use this command to create reports such as:

- Outbound Document Audit Summary (overview)
- Inbound Document Audit Summary (detailed)
- Transaction Activity by Transaction
- Transaction Activity by Trading Partner
- Outbound/Inbound Errors
- Activity Summary by Trading Partner or by Transaction

Syntax

TRANSACTION ACTIVITY DATA EXTRACT

ADDRLN1(*trading partner address line 1*)

ADDRLN2(*trading partner address line 2*)

CMMTLN1(*trading partner comment line 1*)

CMMTLN2(*trading partner comment line 2*)

CMPYNM(*trading partner company name*)

DAYS(*starting date*) TO(*ending date*)

DIR(*processing direction*)

DYNSQL(*use dynamic SQL*)

IFCC(*override condition codes*)

MAPID(*map name*)

SETCC(*condition codes*)

STDDESC(*EDI standard description*)

STDID(*EDI standard ID*)

STDLV(*EDI standard release level*)

STDTRID(*EDI standard transaction set ID*)

STDVR(*EDI standard version*)

TESTMODE(*test transaction mode*)

TPID(*trading partner ID*)

TPNICKN(*trading partner nickname*)

USERPGM(*user program*)

TRANSACTION ACTIVITY DATA EXTRACT command example

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

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

The following is an example of a transaction activity data extract report, after the data has been sorted and formatted.

EDI Invoice Activity Summary for 1H01									
Company Name	Location	Int ID	Trx ID	Std. ID	Dir	Trx MTD	Errors MTD	Trx YTD	Errors YTD
IBM Global Services	Tampa, FL	933489	810	X12V2R1	R	14	1	136	9
CULATER	Dallas, TX	322217	810	X12V2R3	R	49	5	450	8
			810	X12V2R3	R	10	5	90	14
			810	X12V2R3	R	250	23	2200	28
			INVOIC	EDIL921	R	24	0	224	0
Definite Inc.	Paris, FR	005901	810	X12V2R3	R	456	0	4104	0
			810	X12V2R3	R	45	0	405	0
			INVOIC	X12V2R3	R	789	0	7101	0
Jay's Turbo	Perry, NC	555489	810	X12V2R3	R	102	2	918	7
			810	X12V2R3	R	83	3	747	12
			810	X12V2R3	R	89	0	789	0
			810	X12V2R3	R	105	0	945	0
Cool Pool Service	Carmel, CA	748723	810	X12V2R3	R	234	0	1245	2
			810	X12V2R3	R	216	0	1245	4

TRANSACTION DATA EXTRACT command

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

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

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

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

To extract detailed information about enveloped transactions, use the ENVELOPE DATA EXTRACT command (see page 33).

Syntax

TRANSACTION DATA EXTRACT

ACFIELD(*starting application control field data*) TO(*ending application control field data*)

APPLICATION(*write application data record*)

APPLID(*application ID*)

APPRECID(*application receiver department ID*)

APPSNDID(*sender's department ID*)

BATCH(*translated transaction batch ID*)

CONCATENATE(*concatenate extract data*)

DIR(*processing direction*)

DLVDATE(*starting delivery date*) TO(*ending delivery date*)

DLVTIME(*starting delivery time*) TO(*ending delivery time*)

ENVDATE(*starting transaction envelope date*) TO(*ending transaction envelope date*)

ENVTIME(*starting transaction envelope time*) TO(*ending transaction envelope time*)

ENVTYPE(*transaction envelope type*)

EPURDATE(*starting transaction purge date*) TO(*ending transaction purge date*)

FORMAT(*data format ID*)

FUNACKP(*pending functional acknowledgment*)

GROUP(*write group data record*)

GRPCTLNO(*starting sender's group control number*) TO(*ending sender's group control number*)

HANDLE(*starting transaction ID*) TO(*ending transaction ID*)

IFCC(*override condition codes*)

IMAGE(*write image data record*)

INTCTLNO(*starting sender's interchange control nbr*) TO(*ending sender's interchange control nbr*)

INTERCHANGE(*write interchange data record*)

INTRECID(*interchange receiver ID*)

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

TRANSACTION DATA EXTRACT command examples

Example 1: Retrieve the interchange, group, and transaction information for all transactions with trading partner name **PISCES** and interchange control number **000008888**. Place the results in the EDIQUERY file.

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

Example 2: Retrieve the interchange, group, and transaction information for all transactions with application sender ID **SERVO**. Include application and image information.

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

TRANSFORM command

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

Syntax

TRANSFORM

CLEARFILE(*clear specified file contents*)
DICTIONARY(*input dictionary name*)
DOCUMENT(*input data document name*)
IFCC(*override condition codes*)
INFILE(*input data file name*)
INTYPE(*input data file type*)
MAPID(*map name*)
OUTFILE(*output data file name*)
OUTLEN(*maximum output record length*)
OUTTYPE(*output data file type*)
SETCC(*condition codes*)
SYNTAX(*input data syntax type*)
TRACELEVEL(*trace level*)
XMLBCDIC(*EBDCDIC indicator*)
XMLVALIDATE(*XML validation indicator*)
XMLDTDS(*XML DTD path*)

TRANSFORM command example

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

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

TRANSLATE AND ENVELOPE command

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

Syntax

TRANSLATE AND ENVELOPE

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

TRANSLATE AND ENVELOPE command examples

Example 1: Translate and envelope the transactions in ddname **APPDATA**. Place the enveloped transactions in the TD queue specified by the network profile. Return the information record created during translation.

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

Example 2: In CICS, translate and envelope the transactions in TD queue **AP01**. Place the enveloped transactions in TS queue named **ENVDATA**. Return the information record and envelope headers created during translation.

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

TRANSLATE AND SEND command

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



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

Syntax

TRANSLATE AND SEND

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

TRANSLATE AND SEND command examples

Example 1: Translate and send the transactions in file **APPDATA**. Assign batch number **121401** to these transactions.

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

Example 2: Translate and send the transactions in file **RAWFILE**. Data format **POSEND** describes these transactions, which are in raw data format.

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

TRANSLATE TO APPLICATION command

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



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

Syntax

TRANSLATE TO APPLICATION

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

XMLSTDID(*destination EDI standard ID*)
XMLVALIDATE(*XML validation indicator*)

TRANSLATE TO APPLICATION command examples

Example 1: Translate all EDI documents in the Transaction Store that were received on December 14, 2001 with network ID **IINR41**. Place the data in the application file specified by the data format or in the override file specified in the receive usage/rule for the trading partner.

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

Example 2: Translate all EDI documents in the Transaction Store that were received between December 14, 2001 and today's date. Place the data in the application file specified by the data format or in the override file specified in the receive usage/rule for the trading partner.

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

Example 3: Translate all EDI documents in the Transaction Store that were received on December 14, 2001 with network ID **IINR41**. Also translate all EDI data received on December 14, 2001 from trading partner **PISCES**. Place the data in the application file specified by the data format or in the override file specified in the receive usage/rule for the trading partner.

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

TRANSLATE TO STANDARD command

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

Syntax

TRANSLATE TO STANDARD

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

TRANSLATE TO STANDARD command examples

Example 1: Translate the transactions in file **INVOICES** from a raw data format to an EDI standard format and place the results in the Transaction Store. The data format **SENDINVOICES** describes the raw data.

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

Example 2: Translate the transactions in two application files from C and D record format to EDI standard format and place the results in the Transaction Store. Assign batch ID **121401** to these transactions.

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

UNLOAD LOG ENTRIES command

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

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

To ensure concurrency, set the number of deletes (**Numdels**) performed before a COMMIT is issued to a relatively low value. This command may be run as the first step of a multi-step process. (See the table on page 6.)

Syntax

UNLOAD LOG ENTRIES

APPLID(*application ID*)

ARCHIVEFILE(*event log archive file name*)

ARCHIVETYPE(*event log archive file type*)

HOLDFILE(*event log hold file name*)

HOLDTYPE(*event log hold file type*)

IFCC(*override condition codes*)

LOGAEID(*starting event log associated entry ID*) **TO**(*ending event log associated entry ID*)

LOGDATE(*starting event log date*) **TO**(*ending event log date*)

LOGFORM(*starting event log format ID*) **TO**(*ending event log format ID*)

LOGTIME(*starting event log time*) **TO**(*ending event log time*)

LOGUSER(*starting event log user ID*) **TO**(*ending event log user ID*)

NUMDELS(*number of database deletes before commit*)

SETCC(*condition codes*)

UNLOAD LOG ENTRIES command example

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

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

UNPURGE command

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

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

Syntax

UNPURGE

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

UNPURGE command example

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

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

UPDATE STATISTICS command

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



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

Syntax

UPDATE STATISTICS

NUMUPDTS(*number of database updates before commit*)

UPDATE STATISTICS command example

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

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

UPDATE STATUS command

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

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

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

Syntax

UPDATE STATUS

IFCC(*override condition codes*)

NETID(*network ID*)

REQID(*requestor ID*)

SETCC(*condition codes*)

UPDATE STATUS command examples

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

```
PERFORM UPDATE STATUS
```

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

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

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

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

Keyword descriptions

This section describes the keywords used with the DataInterchange Utility commands.

ACCTID

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

This keyword is used with the following commands:

NETWORK ACTIVITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT

ACFIELD

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

This keyword is used with the following commands:

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

ACKFILE

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

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



NOTE: In CICS, DataInterchange does not change the **ACKFILE** value to uppercase.

ACKTYPE

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

MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue - main storage
TS	Temporary storage queue - auxiliary storage (default for CICS)
VS	VSAM entry sequenced data set

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

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

ACTUSAGE

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

NOREPL	An active imported usage/rule replaces an existing active usage/rule in the database (default). If an active usage/rule that matches the key for the imported usage/rule already exists, the imported usage/rule is made inactive.
REPL	An active imported usage/rule replaces an existing active usage/rule in the database. In this case, the existing usage/rule is made inactive and the imported usage/rule becomes the active usage.
FORCE	All imported usages/rules are forced to be active regardless of their current status, or the existence of another active usage/rule in the database. If an active usage/rule that matches the key for the imported usage/rule exists in the database, it is made inactive and the imported usage/rule becomes the active.

This keyword is used only with the IMPORT command.

ADDRLN1

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

This keyword is used with the following commands:

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

ADDRLN2

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

This keyword is used with the following commands:

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

APPFILE

When used with EDI processing commands, this keyword specifies the ddname of the file containing the application data. The data can be C and D records or raw data records. For the format of these records, see “Application file” on page 175. The maximum length is 8.

This keyword is used with the following commands:

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



NOTE: In CICS, DataInterchange does not change the **APPFILE** value to uppercase.

APPLICATION

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

Y	Writes application data records
N	Discards application data records (default)

Received transactions can be translated multiple times and will have an application record for each attempt. Send transactions only have one application record. For the format of these records, see “Application data extract record layout” on page 287.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
 TRANSACTION DATA EXTRACT

APPLID

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

EDIFFS	DataInterchange Utility
EDIMP	DataInterchange Facility
(user-defined)	Your application APPLID

You can switch to another application ID by using DataInterchange's API function codes. The maximum length is 8.

This keyword is used with the following commands:

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

APPRECID

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

This keyword is used with the following commands:

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

APPSEC

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

- U** Opens the application files for both read and write. The files can be opened for extend, and then closed (default). This process ensures that each file has been properly initialized and that DataInterchange does not process data that is not valid.
- R** Opens the application files in read only mode. If this option is chosen, you must make sure the application files are properly initialized.

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD

APPSNDID

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

This keyword is used with the following commands:

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

APPTYPE

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

MQ	DataInterchange MQSeries Queue profile member name
PG	Response program (inbound processing only)
TD	Transient data queue
TM	Temporary storage queue - main storage
TS	Temporary storage queue - auxiliary storage (default for CICS)
TX	Response transaction (inbound processing only)

This keyword is used with the following commands:

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

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

ARCHIVEFILE

The event log archive file name. Event log entries selected for removal are written to this file. The maximum length is 8.

This keyword is used with the following commands:

REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES

ARCHIVETYPE

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

MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue - main storage
TS	Temporary storage queue - auxiliary storage (default for CICS)

This keyword is used with the following commands:

REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES

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

ASSERTLVL

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

This keyword is used with the following commands:

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

BATCH

Specifies the batch ID assigned to a transaction when it was translated. You can use this keyword to select only transactions with a particular batch ID. The maximum length is 8.

This keyword is used with the following commands:

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

BATCHSET

Specifies that a user-defined ID be assigned to the transaction when it is translated. You can use this keyword to identify transactions you want to retrieve as a group. The transactions remain independent. An action performed on one transaction does not affect other transactions with the same batch ID. The maximum length is 8. The default batch ID is a date and time stamp in the format *ddhhmmss*.

This keyword is used with the following commands:

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

CCEXCEPTION

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

- Y** Generates a minimum job step condition code of 0003 if application data is written to the exception file.
- N** Generates a job-step condition code of 0000 if translation completes successfully and the application data was written to the exception file (default). Does not report application data written to the exception file as an error.
- X** Generates a job-step condition code of 903 to distinguish this from other errors that result in a 0003 condition code.

This keyword is used with the following commands:

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

CLEARFILE

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

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

- Y** Clears the file before writing the translated output
- N** Does not clear the file before writing the translated output (default)

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

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

ENVELOPE AND SEND	RCVFILE AND SEND
RECEIVE	REENVELOPE AND SEND
RECEIVE AND DEENVELOPE	SEND
RECEIVE AND SEND	TRANSLATE AND SEND
RECEIVE AND TRANSLATE	



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

CLIENT

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

This keyword is used with the following commands:

SAP STATUS EXTRACT
SAP STATUS REMOVE

CMMTLN1

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

This keyword is used with the following commands:

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

CMMTLN2

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

This keyword is used with the following commands:

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

CMPYNM

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

This keyword is used with the following commands:

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

CNTCTNM

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

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

CNTCTPH

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

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

CONCATENATE

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

- Y** Concatenates categories and writes them as a single record
- N** Writes information as separate records (default)

If concatenation is requested, the following hierarchy is used:

1. Interchange
2. Group
3. Transaction
4. Application

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



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

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

CTLFILE

The ddname of an import or export control file that describes what data is to be imported or exported. For information about the control file format, see the table on page 187.

This keyword is used with the following commands:

EXPORT
IMPORT

CTLTYPE

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

MQ DataInterchange MQSeries Queue profile member name
TD Transient data queue
TM Temporary storage queue - main storage
TS Temporary storage queue - auxiliary storage (default for CICS)

This keyword is used with the following commands:

EXPORT
IMPORT

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

DAYS

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

This keyword is used with the following commands:

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

DICTIONARY

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

This keyword is used only with the TRANSFORM command.

DIERRFILTER

Specifies the initial set of errors to filter for this translation session. To override the filters defined in a usage/rule, you can use the DIERRFILTER keyword with a value of **IGNORE** to tell DataInterchange to ignore all reported errors. This allows you to view all errors for a particular session without changing the map generally used. For more information on error filtering, see “Error filtering” on page 21, the field definition for ERRFILTER on page 320, or refer to the *DataInterchange Administrator's Guide*.

This keyword is used with the following commands:

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

DIR

Indicates the direction of the transmission. Valid values are:

R	Receiving
S	Sending

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT	QUERY
HOLD	RECONSTRUCT
PRINT ACKNOWLEDGMENT IMAGE	RECONSTRUCT AND SEND
PRINT ACTIVITY SUMMARY	RELEASE
PRINT EVENT LOG	REMOVE TRANSACTIONS
PRINT STATUS SUMMARY	TRADING PARTNER CAPABILITY DATA EXTRACT
PRINT STATUS SUMMARY2	TRANSACTION ACTIVITY DATA EXTRACT
PRINT TRANSACTION DETAILS	TRANSACTION DATA EXTRACT
PRINT TRANSACTION IMAGE	UNPURGE
PURGE	

DLVDATE

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

This keyword is used with the following commands:

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

DLVTIME

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

This keyword is used with the following commands:

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

DOCUMENT

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

This keyword is used only with the **TRANSFORM** command.

DUPCHECK

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

- Y or (other)** Checks for duplicate IDs and ID qualifiers on TPPROF import (default)
- N** Bypasses the duplicate checks on TPPROF import



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

This keyword is used only with the IMPORT command.

DUPENV

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

- Y** Processes duplicate envelopes (default).
- N** Does not process duplicate envelopes. A condition code of 0005 is returned if you attempt to process a duplicate envelope.

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

DYNSQL

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

- Y** Uses dynamic SQL
- N** Uses static SQL (default)

This keyword is used with the following commands:

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

EENVDATE

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

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

EIFORMAT

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

TAGGED Exports in tagged record format (default)

FIXED Exports in fixed record format

This keyword is used only with the EXPORT command.

ENVDATE

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

This keyword is used with the following commands:

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

ENVPRBREAK

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

Y Starts a new interchange envelope

N Starts a new group envelope (default)

This keyword is used with the following commands:

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

ENVTIME

The time at which the transaction was enveloped. The maximum length is 8.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT	PRINT TRANSACTION IMAGE
HOLD	PURGE
PRINT ACKNOWLEDGMENT IMAGE	QUERY
PRINT ACTIVITY SUMMARY	REENVELOPE
PRINT EVENT LOG	REENVELOPE AND SEND
PRINT STATUS SUMMARY	REMOVE TRANSACTIONS
PRINT STATUS SUMMARY2	TRANSACTION DATA EXTRACT
PRINT TRANSACTION DETAILS	UNPURGE

ENVTYPE

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

E	UNB/UNZ
I	ICS
T	STX/END
U	BG/EG
X	ISA/IEA
0	Envelopes with no interchange header and trailer

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

E	EDIFACT
I	ICS or non-EDI file
T	UN/TDI
U	UCS
X	X12 (default)

This keyword is used with the following commands:

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

EPURDATE

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

This keyword is used with the following commands:

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

EXTENDC

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

- Y** Uses the extended C record format
- N** Does not use the extended C record format (default)

This keyword is used with the following commands:

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

FADELAY

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

- Y** Puts functional acknowledgments in the Transaction Store but does not envelope them
- N** Puts functional acknowledgments in the Transaction Store and also envelopes them to one of the following (default) files:
 - The file specified by the **FUNACKFILE** keyword, if present
 - The TD queue from the network profile
 - QDATA, QDATAU, or QDATAE (depending on envelope type)

This keyword is used with the following commands:

DEENVELOPE
 DEENVELOPE AND TRANSLATE

RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

FILEID

Specifies the ddname of a file used to:

- Write data during an ENVELOPE operation
- Read data during a DEENVELOPE operation
- Send data during a SEND operation
- Write data during a RECEIVE operation

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

For inbound processing, if you do not specify this keyword, the value from the **Receive file name** field from the mailbox (requestor) profile is used as the ddname for the envelope file.

The maximum length is 8.



NOTES:

1. On ENVELOPE and DEENVELOPE commands, all transactions are placed in this file. You should also specify the **NETID** keyword to make sure all transactions you select are for the same network.
2. **FILEID** contains the name of a TS queue. You should include this keyword to make sure that different applications running in the same CICS region do not envelope transactions to the same TS queue.
3. In CICS, DataInterchange does not change the **FILEID** value to uppercase.

This keyword is used with the following commands:

DEENVELOPE	RECONSTRUCT AND SEND
DEENVELOPE AND SEND	RCVFILE AND SEND
ENVELOPE	REENVELOPE
ENVELOPE AND SEND	REENVELOPE AND SEND
RECEIVE	SEND
RECEIVE AND DEENVELOPE	SENDFILE
RECEIVE AND SEND	TRANSLATE AND ENVELOPE
RECEIVE AND TRANSLATE	TRANSLATE AND SEND
RECONSTRUCT	

FIXEDFILEID

Specifies the ddname of the file used for output during fixed-to-fixed translation processing. Data is written to the file during an ENVELOPE operation for fixed-to-fixed translation. The maximum length is 8.

If you do not specify this keyword, the ddname (based on the EDI standard ID) is the same as the **Application file name** in the target ADF definition. To create a unique envelope file for each trading partner, specify the **File Suffix** field in the trading partner profile.



NOTES:

1. On ENVELOPE commands, all transactions are placed in this file. You should also specify the **NETID** keyword to make sure all transactions you select are for the same network.
2. **FIXEDFILEID** contains the name of a TS queue. You should include this keyword to make sure that different applications running in the same CICS region do not envelope transactions to the same TS queue.
3. In CICS, DataInterchange does not change the **FIXEDFILEID** value to uppercase.

This keyword is used with the following commands:

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

FORCETEST

Indicates whether the deenvelope or translate-to-application processes should be forced to select a test usage/rule, regardless of the test indicator value in the envelope. This keyword is most useful when receiving test envelopes that do not have a test indicator (such as the UCS BG). In this case, you can use this keyword to force the translator to consider the envelopes for testing and only look for test usages/rules. Valid values are:

- Y** Forces the process to test mode and select only a test usage/rule if one is defined. If a test usage/rule is not found, an error is generated and the transaction is rejected.
- If **FORCETEST(Y)** is used with the DEENVELOPE command, then it must also be used on the TRANSLATE TO APPLICATION or RETRANSLATE TO APPLICATION commands to select the deenveloped transactions.
- N** Uses the test indicator from the envelope to determine which usage/rule to select (default). An envelope without a test indicator is always considered a production envelope.

This keyword is used with the following commands:

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

FORMAT

The ID of the data format associated with the transaction or transactions you want to select. The maximum length is 16.

This keyword is used with the following commands:

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

FUNACKFILE

The ddname of the file that you want to use for returning functional acknowledgments for the deenveloped transactions. You can use this keyword if you do not want to use the TD queue specified in the network profile. The maximum length is 8.



NOTE: In CICS, DataInterchange does not change the **FUNACKFILE** value to uppercase and **FUNACKFILE** contains the name of a TS queue.

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

FUNACKP

Indicates whether transactions with functional acknowledgments pending should be selected. Valid values are:

- Y** Selects transactions for which a functional acknowledgment was requested but not received
- N** Selects transactions for which a functional acknowledgment was not requested or has already been received

This keyword is used with the following commands:

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

FUNACKREQ

Indicates whether the functional acknowledgment envelope file is required. DataInterchange will produce an error if unable to open it. Valid values are:

Y	Requires the functional acknowledgment envelope file
N	Does not require the functional acknowledgment envelope file (default)

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

GROUP

Indicates whether group data records are written to the EDIQUERY file. Valid values are:

Y	Writes group data records
N	Discards group data records (default)

For the format of these records, see “Group data extract record layout” on page 283.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

GRPCTLNO

Specifies the group control number assigned by the sender to identify the functional group. The maximum length is 14.

This keyword is used with the following commands:

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

HANDLE

Specifies the ID assigned by the system to a transaction when it is placed in the Transaction Store. To ensure uniqueness, the ID is a concatenation of the date, time, and a sequence number in format: *YYYYMMDDHHMMSSnnnnnn*

You can use this keyword to envelope a specific EDI document or all the documents whose time stamp falls within a given range. The system left-justifies and pads your entries. The **FROM** value is padded with 0s and the **TO** value is padded with 9s. To select transactions for the current date, use an asterisk (*).

This keyword is used with the following commands:

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

HOLDFILE

The event log hold file name. The UNLOAD LOG ENTRIES command copies the non-archived event log entries into this file. The LOAD LOG ENTRIES command loads the records from this file into the event log. The maximum length is 8.

This keyword is used with the following commands:

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

HOLDTYPE

Indicates the event log hold file type. Used only for CICS and MQ (which is supported in both MVS and CICS). Valid values are:

- MQ** DataInterchange MQSeries Queue profile member name
- TD** Transient data queue
- TM** Temporary storage queue - main storage
- TS** Temporary storage queue - auxiliary storage (default in CICS)

In MVS, the default is the ddname of the sequential file.

This keyword is used with the following commands:

LOAD LOG ENTRIES
 REMOVE LOG ENTRIES
 UNLOAD LOG ENTRIES

IACCESS

Indicates how the interchange should be presented to the IEXIT program. Valid values are:

- F** Gives the interchange to the exit in a file. The interchange is written to the TD queue file, and then the IEXIT program is started.
- M** Gives the interchange to the exit in virtual storage. Applies only when **ITYPE** is **UE** (user exit).

This keyword is used with the following commands:

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

IAREA

Specifies up to 16 bytes of information, the address of which is provided to the IEXIT program. Applies only to MVS programs. In CICS, the address provided to the IEXIT program is the address of the utility control block. The maximum length is 16.

This keyword is used with the following commands:

DEENVELOPE	RECEIVE AND TRANSLATE
DEENVELOPE AND TRANSLATE	REENVELOPE
ENVELOPE	REENVELOPE AND SEND
ENVELOPE AND SEND	TRANSLATE AND ENVELOPE
RECEIVE AND DEENVELOPE	TRANSLATE AND SEND

ID

The name of a DataInterchange profile such as REQPROF, TPPROF, or NETPROF. This must be a valid DataInterchange profile ID. The maximum length is 8.

This keyword is used with the following commands:

QUERY PROFILE
DELETE PROFILE

IEXIT

The name of the program to receive control as each interchange is processed. See the **ITYPE** field description on page 143 for the types of program you can use.

This keyword is used with the following commands:

DEENVELOPE	RECEIVE AND TRANSLATE
DEENVELOPE AND TRANSLATE	REENVELOPE
ENVELOPE	REENVELOPE AND SEND
ENVELOPE AND SEND	TRANSLATE AND ENVELOPE
RECEIVE AND DEENVELOPE	TRANSLATE AND SEND

IFCC

Specifies the condition codes that you want to override. You can specify up to 10 utility condition codes, separated by commas. The codes are checked by the DataInterchange Utility and overridden with values from the **SETCC** keyword on a one-to-one basis. If you specify this keyword, you must specify the SETCC keyword.

This keyword can be used with any utility PERFORM command.

IMAGE

Indicates whether **image data records** are written to the EDIQUERY file. Valid values are:

Y	Writes image data records
N	Discards image data records (default)

Images are always written as separate records. For the format of these records, see “Transaction/Acknowledgment image data extract record layout” on page 288.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

INFILE

The input file containing the data. For MVS, this is the ddname or MQSeries queue name. For CICS this is TS or TD queue, MQSeries queue, or VSAM data set.

This keyword is used only with the TRANSFORM command.

INMEMTRANS

Specifies the number of transactions held in memory before the database updates are attempted. This field is valid only if envelope level recovery (**RECOVERY(E)**) is in effect. Keeping transactions in storage delays the time when the database lock is obtained and reduces the length of time that the database lock is held. The more transactions kept in storage, the higher concurrency rate DataInterchange can achieve. The amount of storage used for each **transaction** is approximately 2K. Valid values are **1** to **65535**. The default is **100**.

This keyword is used with the following commands:

DEENVELOPE	REENVELOPE
DEENVELOPE AND TRANSLATE	REENVELOPE AND SEND
ENVELOPE	TRANSLATE AND ENVELOPE
ENVELOPE AND SEND	TRANSLATE AND SEND
RECEIVE AND DEENVELOPE	

INTCTLNO

Specifies the interchange control number assigned by the sender to identify the interchange data. The maximum length is 14.

This keyword is used with the following commands:

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

INTERCHANGE

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

Y	Writes interchange data records
N	Discards interchange data records (default)

For the format of these records, see “Interchange data extract record layout” on page 282.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

INTID

The interchange sender/receiver ID of the trading partner as entered in the **Interchange ID** field of the trading partner profile.

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

INTRECID

The interchange receiver ID (assigned by the receiver) that identifies the receiver to the sender. The maximum length is 35.

This keyword is used with the following commands:

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

INTSNDID

The interchange sender ID (assigned by the sender) that identifies the sender to the receiver. The maximum length is 35.

This keyword is used with the following commands:

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

INTYPE

Indicates the type of file specified with the **INFILE** keyword. Used only for CICS and MQ (which is supported in both MVS and CICS). For CICS, when you specify the **INFILE** keyword, you must also specify this keyword. Valid values are:

MQ MQSeries queue profile member name

TD Transient data queue

- TM** Temporary storage queue - main storage
- TS** Temporary storage queue - auxiliary storage (default in CICS)
- VS** VSAM data set

This keyword is used only with the TRANSFORM command.

ITPBREAK

Indicates whether a new interchange envelope starts when the internal trading partner ID changes. Valid values are:

- Y** Always starts new interchange envelope (default)
- N** Does not necessarily start a new interchange envelope

This keyword is used with the following commands:

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

ITYPE

Indicates the type of program specified in IEXIT. If you specify this keyword, you must also specify the IEXIT keyword. Valid values are:

- PG** A program that should be linked to using the EXEC CICS LINK command in CICS
- UE** A DataInterchange user exit program defined in the ADAMCTL profile

This keyword is used with the following commands:

DEENVELOPE	RECEIVE AND TRANSLATE
DEENVELOPE AND TRANSLATE	REENVELOPE
ENVELOPE	REENVELOPE AND SEND
ENVELOPE AND SEND	TRANSLATE AND ENVELOPE
RECEIVE AND DEENVELOPE	TRANSLATE AND SEND

LASTTRXDATE

The date of the last transaction sent to or received from the trading partner.

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

LEVEL

Specifies what information is dumped or how much detail is traced during processing.

This keyword is used with the following commands:

- GLB DUMP
- GLB TRACE

For the GLB DUMP command, valid values are:

- 1** Dumps everything (default)
- 2** Dumps working storage
- 3** Dumps entire dataspace
- 10** Dumps common area
- 11** Dumps record index area
- 12** Dumps data area

For the GLB TRACE command, valid values are:

- 0** Ends trace (the default)
- 1** Starts trace functions only
- 2** Starts trace functions and subroutines
- 3** Starts trace functions, subroutines, and subroutine keypoints

LOGAEID

The associated entry ID in the event log. This keyword can be used with the **TO** keyword to specify a range of associated entry IDs for selecting event log entries. The maximum length is 40.

This keyword is used with the following commands:

LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES

LOGDATE

The event log date. This keyword can be used with the **TO** keyword to specify a range of dates for selecting event log entries. The maximum length is 8.

This keyword is used with the following commands:

LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES

LOGFORM

The event log format ID. This keyword can be used with the **TO** keyword to specify a range of format IDs for selecting event log entries. The maximum length is 16.

This keyword is used with the following commands:

LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES

LOGTIME

The event log time. This keyword can be used with the **TO** keyword to specify a range of times for selecting event log entries. The length is 6.

This keyword is used with the following commands:

LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES

LOGUSER

The event log user ID. This keyword can be used with the **TO** keyword to specify a range of user IDs for selecting event log entries. The maximum length is 8.

This keyword is used with the following commands:

LOAD LOG ENTRIES
REMOVE LOG ENTRIES
UNLOAD LOG ENTRIES

MAPID

The map name. For the TRANSFORM command, the value in this field overrides the map name specified in data transformation rules.

This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT
TRANSFORM

MAXRUNTIME

The maximum time in minutes that the REMOVE TRANSACTIONS process is allowed to run. Once the specified time is reached, the REMOVE TRANSACTIONS process stops even if it has not completed. When the REMOVE TRANSACTIONS process is running, other DataInterchange processes are prevented from running. Setting this keyword and value is useful when the REMOVE TRANSACTION process is to run standalone for a limited time. The default value is **0** (no maximum run time).

This keyword is used only with the REMOVE TRANSACTIONS command.

MEMBER

The name of a DataInterchange profile member. The maximum length is dependent upon the type of profile but cannot exceed 35.

This keyword is used with the following commands:

DELETE PROFILE
QUERY PROFILE
REPORT CONTINUOUS RECEIVE STATUS
START CONTINUOUS RECEIVE
STOP CONTINUOUS RECEIVE

MERGED

Indicates whether the transaction image is printed with a new line for each segment, and merged with the functional acknowledgment image. Valid values are:

- Y** Prints each segment on a new line
- N** Does not print each segment on a new line

This keyword is used only with the PRINT TRANSACTION IMAGE command.

MRREQID

A requestor ID used to associate management reporting statistics with DEENVELOPE processing instead of RECEIVE processing. You can use this keyword if you receive interchanges without using DataInterchange, and you want to keep management reporting receive statistics on those interchanges. Do not use this keyword if management reporting statistics were created when the interchange was received (default). Use this keyword only in exceptional situations.

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE

MSGUCLASS

An override message user class for a send or receive type command. If you specify this keyword, its value overrides the value specified in the mailbox (requestor) profile member identified by the **REQID** keyword. The maximum length is 8.

This keyword is used with the following commands:

ENVELOPE AND SEND	RCVFILE AND SEND
RECEIVE	REENVELOPE AND SEND
RECEIVE AND DEENVELOPE	SEND
RECEIVE AND SEND	SENDFILE
RECEIVE AND TRANSLATE	TRANSLATE AND SEND
RECONSTRUCT AND SEND	

MULTIDOC

Indicates whether the XML input file contains multiple documents. Valid values are:

- Y** Multiple documents. The input message must be in EBCDIC format and each document must begin with an XML declaration (<?xml...).
- N** One document (default)

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE

NETACKP

Indicates whether to select transactions for which network acknowledgments are pending. Valid values are:

- Y** Selects transactions for which a network acknowledgment was requested but not received
- N** Selects transactions for which a network acknowledgment is not pending or was not requested

This keyword is used with the following commands:

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

NETID

The network ID as entered in the network profile. The maximum length is 8.

This keyword is used with the following commands:

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

NETNAME

The name of the network as entered in the **Network name** field of the network profile. This field is case sensitive. The maximum length is 30.

This keyword is used only with the NETWORK ACTIVITY DATA EXTRACT command.

NETSTAT

Indicates the network status of a transaction for which a send has been requested. Valid values are:

30	Enveloped
31	Envelope error
41	Sent with errors
42	Send request error
43	Not sent net error
46	Send started
48	Send requested
49	Sent to network
50	Accepted by network
51	Delivered by network
52	Purged by network
53	Recall requested
54	Recall request error
55	Recalled

For complete status information, refer to the *DataInterchange Administrator's Guide*.

This keyword is used with the following commands:

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

NEWAPPLID

The application ID used to change to another application when loading event log entries into an event log. The maximum length is 8.

This keyword is used only with the LOAD LOG ENTRIES command.

NOMSG

Indicates whether extraneous messages should appear in the print file. Applies only to EXPORT and IMPORT commands, and is normally used when large numbers of records are being exported or imported. For example, it may be used with export-all (exporting after building a control file using program EDIXPEA) or import-all (using a control file with category 9).

Y Suppresses the following messages from appearing in the print file:

EI0048, EI0049, EI0062, FF0512, FF0514

Also suppresses certain error conditions. For example, if in your command file you specify that you want to export CONTRECV members, and no CONTRECV profile exists on your system, this error is ignored. Also, if you specify certain associated objects that do not exist, this error is also ignored.

N Does not suppress error messages or conditions (default).

This keyword is used with the following commands:

EXPORT
IMPORT

NUMDELS

Specifies the number of database deletions performed before DataInterchange issues a database commit. DB2 includes a site-dependent value (in **NUMLKUS**) that controls the maximum number of page locks that a single REMOVE TRANSACTIONS process can apply. When the REMOVE TRANSACTIONS process is not running standalone, page locks are obtained against the database. If the **NUMDELS** value is too high, DB2 may stop the REMOVE TRANSACTIONS process because the value in **NUMLKUS** has been exceeded. If this happens, use a smaller number in **NUMDELS** so that DataInterchange issues database commits more frequently, which will release page locks more quickly. The default is **100** and the maximum is **1000**. If a value greater than 1000 is entered, the default of 100 is used.

This keyword is used with the following commands:

REMOVE LOG ENTRIES	RESET STATISTICS
REMOVE STATISTICS	UNLOAD LOG ENTRIES
REMOVE TRANSACTIONS	

NUMUPDTS

Specifies the number of database updates performed before DataInterchange issues a database commit. If you are experiencing timeouts using a command that includes this parameter, you can reduce the amount of time that the command holds page locks by forcing more frequent COMMITs. The default for NUMUPDTS is **50**. If you want more frequent COMMITs performed, specify a number smaller than 50.

If you are trying to improve the performance of a command that includes this parameter, you can specify a value larger than 50. This forces DataInterchange to COMMIT less frequently and should speed up processing.

This keyword is used only with the UPDATE STATISTICS command.

ONELOGICAPP

Indicates whether the **APPFILE** in the current WHERE clause and all **APPFILEs** in proceeding WHERE clauses are considered one logical file. When chosen, envelope breaks are avoided between **APPFILE** processing. You can use this keyword to avoid **APPFILE** switching that causes envelope breaks when processing multiple raw data files by placing multiple raw data files in the same envelope without delaying enveloping. If the **FILEID** keyword is specified in combination with this keyword in multiple WHERE clauses, the value in the first **FILEID** found is used to envelope the data. You can also use this option with C and D record formats. Valid values are:

- Y** Treats multiple application files as one logical file
- N** Processes each file independently; envelope breaks occur at the end of each application file (default)

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
TRANSLATE AND SEND

ONEMSG

Indicates whether all MQSeries messages are read from an MQSeries queue or only one message is read from the queue. The queue can be either a receive file or an application send file. Applies only to MQ. An MQSeries message is defined as a logical set of MQ records with the same MSGID. This keyword also controls MQ message descriptor propagation when you can set it on the TRANSLATE AND SEND and RECEIVE AND TRANSLATE commands where the application file and the send or receive file are both MQ queues. If you want MQ message descriptors to propagate during processing, set this keyword to **Y**. Valid values are:

- Y** Reads one MQSeries message (one MSGID value) from an MQSeries queue
- N** Reads all MQSeries messages from an MQSeries queue (default)

This keyword is used with the following commands:

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

OPTRECS

Indicates which optional records are created during translation. **I** is valid only for the TRANSLATE TO STANDARD command (delayed enveloping). **Q** is not valid for the TRANSLATE or RETRANSLATE TO APPLICATION commands. The maximum length is 5. Valid values are:

- E** Envelope header
- G** Group header
- I** Information
- Q** Queuing
- T** Transaction header

For more information on these records, see “Optional records” on page 271.

This keyword is used with the following commands:

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

OUTFILE

The ddname of an output file (or the name of a TS queue or TD queue). For CICS, when you specify this keyword, you must also specify the **OUTTYPE** keyword. For the TRANSFORM command, the value in this field overrides any output file names generated during processing, such as the filename from the trading partner profile. The maximum length is 8.

This keyword is used with the following commands:

QUERY PROFILE
SAP STATUS EXTRACT
TRANSFORM

OUTFORMAT

Indicates the format in which the output is written. Valid values are:

F	Fixed
N	Native
T	Tagged

This keyword is used only with the QUERY PROFILE command.

OUTLEN

Specifies the maximum record length for the output data.

This keyword is used only with the TRANSFORM command.

OUTTYPE

Indicates the file type of **OUTFILE**. Applies only for CICS and MQ (which is supported in both CICS and MVS). For CICS, when you specify the **OUTFILE** keyword, you must specify this keyword. Valid values are:

MQ	DataInterchange MQSeries Queue profile member name
TD	Transient data queue
TM	Temporary storage queue (main storage)
TS	Temporary storage queue (auxiliary storage)
VS	VSAM data set

This keyword is used with the following commands:

QUERY PROFILE
SAP STATUS EXTRACT
TRANSFORM

For MVS, if you do not specify this keyword, this field is ignored and the ddname of a sequential file is used. For CICS, the default is **TS**.

PAGE

Indicates whether pageable translation should be enabled. For more information about pageable translation, see “Pageable translation work file (EDIVAX)” on page 182, and “Pageable translation” on page 337. Valid values are:

Y Enables pageable translation
N Disables pageable translation

This keyword is used with the following commands:

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

PRIORTO

Specifies the date before which all statistics will be deleted.

This keyword is used with the following commands:

REMOVE STATISTICS
RESET STATISTICS
SAP STATUS REMOVE

PURGINT

Specifies the number of days that a transaction remains in the Transaction Store before being marked for purging. If you do not specify this keyword and value, or if you specify a value of **0**, the DataInterchange Utility uses **30** days as the default. The maximum value is **9999**. You can use a negative value to indicate that a transaction's store time expired on a date in the past. For example, if you specify **PURGINT with a value of -2**, the purge date is set at two days ago. The minimum value is **-999**. If a functional or network acknowledgment is pending when the store time expires, the transaction retains its current store status until the acknowledgment is no longer pending.

This keyword is used with the following commands:

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

RAWDATA

Indicates whether you want the data translated from EDI standard format to raw format, or to C and D records. Valid values are:

Y	Raw data format
N	C and D record format

For details about these record formats, see “Application file” on page 175. For an explanation of the file to which data will be written, see “Envelope file” on page 176.

This keyword has this meaning when used with the following commands:

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

For Fixed-to-Fixed translation, this keyword indicates whether the output data should be written in raw data format or in the C and D record format. For a description of C and D records, see “DataInterchange Utility records format” on page 257

This keyword has this meaning when used with the following commands:

ENVELOPE	REENVELOPE
ENVELOPE AND SEND	REENVELOPE AND SEND
RECEIVE AND SEND	TRANSLATE AND ENVELOPE
RECONSTRUCT	TRANSLATE AND SEND
RECONSTRUCT AND SEND	TRANSLATE TO STANDARD

RAWFMTID

Applies only for raw data. Specifies the format ID of the application data file. The maximum length is 16.

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD

RAWTEST

Applies only to raw data. Indicates to the receiver whether the transaction is for testing. Valid values are:

- Y** Test transactions. All the transactions are test transactions, and a test usage/rule should be used, if it exists. If a test usage/rule does not exist, the production usage/rule should be used. Same as **RAWUSAGE(T)**.
- N** Production transactions (default). All the transactions are production transactions and only the production usage/rule should be used. Same as **RAWUSAGE(P)**.
- U** Either test or production transactions. DataInterchange determines the status of the transaction based on the presence of an active test usage/rule. If an active test usage/rule exists, the transaction is considered a test transaction. If an active test usage/rule does not exist, the transaction is considered a production transaction. Same as **RAWUSAGE(U)**.



NOTE: This keyword is being replaced by **RAWUSAGE**. The **RAWUSAGE** value overrides **RAWTEST** if both values are specified. **RAWTEST** is provided for transitional purposes only, and will be removed in a future release.

The translator uses this value to set the test indicator in the interchange header (0035 for EDIFACT, I14 for X12). For C and D records, the **TESTIND** field of the control record serves the same purpose as this keyword.

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD

RAWUSAGE

Applies only for raw data. Indicates to the receiver whether the transaction is for production, testing, or information. Valid values are;

- P** Production transactions (default). All of the transactions are production transactions and only the production usage/rule should be used.
- T** Test transactions. All of the transactions are test transactions, and a test usage/rule should be used. If a test usage/rule does not exist, the production usage/rule should be used.
- I** Information transaction. An information usage/rule is used if one exists. If one does not exist, a production usage/rule should be used. If a production usage/rule does not exist, an error occurs.
- U** Either test or production transactions. DataInterchange determines the status of the transaction based on the presence of an active test usage/rule. If an active test usage/rule exists, the transaction is considered a test transaction. If an active test usage/rule does not exist, the transaction is considered a production transaction.

The translator uses this value to set the test indicator in the interchange header (0035 for EDIFACT, I14 for X12). For C and D records, the **TESTIND** field of the control record serves the same purpose as this keyword.

This keyword is used with the following commands:

TRANSLATE AND ENVELOPE
TRANSLATE AND SEND
TRANSLATE TO STANDARD

RECEIVEACKDATA

Indicates whether detailed receive acknowledgment data is written to the EDIQUERY file. Valid values are:

- Y** Writes detailed acknowledgment data
- N** Discards detailed acknowledgment data (default)

Detailed acknowledgment data includes interchange, group, and transaction data for each acknowledgment transaction. This information is concatenated with the group and transaction records to which it applies. For the format of these records, see “Transaction/Acknowledgment image data extract record layout” on page 288.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

RECEIVEACKIMAGE

Indicates whether the receive acknowledgment image is written to the EDIQUERY file. Valid values are:

- Y** Writes receive acknowledgments
- N** Discards receive acknowledgments (default)

Images are always written as separate records. For the format of these records, see “Transaction/Acknowledgment image data extract record layout” on page 288.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

RECOVBAD

Indicates whether the translation process will try to recover from a bad EDI standard data flag.

Valid values are:

- Y** The translation process tries to recover from bad EDI standard data. The translator checks for the standard interchange headers when a segment terminator is missing from a particular standard segment. The translator attempts to reset the delimiters and check the current segment/record for the segment terminator.
- N** The translation process does not try to recover (default).

This keyword is used with the following commands:

DEENVELOPE
DEENVELOPE AND TRANSLATE
RECEIVE AND DEENVELOPE
RECEIVE AND TRANSLATE

RECOVERY

Indicates the unit of work. The default is environmentally dependent. Valid values are:

- E** Issues a database commit after each envelope (default for CICS)
- T** Issues a database commit after each transaction (default for MVS)

This keyword is used with the following commands:

DEENVELOPE	REENVELOPE
DEENVELOPE AND TRANSLATE	REENVELOPE AND SEND
ENVELOPE	TRANSLATE AND ENVELOPE
ENVELOPE AND SEND	TRANSLATE AND SEND
RECEIVE AND DEENVELOPE	

REQID

The requestor ID as entered in the mailbox (requestor) profile. Since each WHERE clause can contain only one requestor ID, you must add a WHERE clause for each requestor ID. A send is issued for each requestor ID for which data was queued. The maximum length is 16.

This keyword is used with the following commands:

CLOSE MAILBOX	RECEIVE AND SEND
DEENVELOPE	RECEIVE AND TRANSLATE
DEENVELOPE AND TRANSLATE	RCVFILE AND SEND
ENVELOPE AND SEND	REENVELOPE AND SEND
NETWORK ACTIVITY DATA EXTRACT	SEND
PROCESS NETWORK ACKS	SENDFILE
RECEIVE	TRANSLATE AND SEND
RECEIVE AND DEENVELOPE	UPDATE STATUS

REQTP

The trading partner associated with a given requestor ID for use with direct connection networks such as point-to-point. This keyword is used to eliminate the need to specify a separate requestor ID for each trading partner.

This keyword is used only with the ENVELOPE AND SEND command.

RESET

Indicates whether the output file is reset to receive new data or the new data is appended to the data already in the output file. Valid values are:

- Y** Resets the output file
- N** Appends data to the output file

This keyword is used with the following commands:

GLB DUMP
GLB TRACE

SAPSTAT

Specifies the SAP status value to extract or, when used with the **TO** keyword, the range of values to extract. Valid values are **all** or **04-22**. The default is **all**.

This keyword is used with the following commands:

RECEIVE AND SEND
SAP STATUS EXTRACT
SAP STATUS REMOVE

SAPUPDT

Indicates whether SAP status tracking is desired. Valid values are:

- Y** Tracks SAP status and writes status records
- N** Does not track SAP status or write status records (default)

This keyword is used with the following commands:

DEENVELOPE	REENVELOPE
DEENVELOPE AND TRANSLATE	REENVELOPE AND SEND
ENVELOPE	SEND
ENVELOPE AND SEND	TRANSLATE AND ENVELOPE
RECEIVE AND SEND	TRANSLATE AND SEND
RCVFILE AND SEND	TRANSLATE TO STANDARD

SCRIPT

Specifies the value that can be used by communication software to identify a set of instructions to follow when processing requests for services. This set of instructions would be part of the communication software package and not part of DataInterchange. The maximum length is 8.

This keyword is used with the following commands:

ENVELOPE AND SEND	RESTART SEND
RECEIVE AND SEND	SEND
RECONSTRUCT AND SEND	SENDFILE
RCVFILE AND SEND	TRANSLATE AND SEND
REENVELOPE AND SEND	

SEGMENTED

Indicates whether the image is printed with each segment starting on a new line. Valid values are:

- Y** Starts a new line for each segment
- N** Does not start a new line for each segment

This keyword is used with the following commands:

PRINT ACKNOWLEDGMENT IMAGE
PRINT TRANSACTION IMAGE

SENDACKDATA

Indicates whether detailed send acknowledgment data is written to the EDIQUERY file. Valid values are:

- Y** Writes detailed acknowledgment data
- N** Discards detailed acknowledgment data (default)

Detailed acknowledgment data includes interchange, group, and transaction data for each acknowledgment transaction. This information is concatenated with the group and transaction record to which it applies. **This** keyword is ignored unless **GROUP** or **TRANSACTION** is set to **Y**. For the format of these records, see “Transaction/Acknowledgment image data extract record layout” on page 288.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

SENDACKIMAGE

Indicates whether the send acknowledgment image is written to the EDIQUERY file. Valid values are:

- Y** Writes the generated acknowledgment record
- N** Discards the generated acknowledgment record (default)

Images are always written as separate records. For the format of these records, see “Transaction/Acknowledgment image data extract record layout” on page 288.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

SEQNUM

Indicates whether network profile member sequence numbers should be incremented during send processing. This is an optional keyword. Specifying a value of **N** turns off network sequence numbers and saves associated overhead. Valid values are:

- Y** Increments network profile member sequence numbers (default)
- N** Does not increment network profile member sequence numbers.

This keyword is used with the following commands:

ENVELOPE AND SEND	SEND
REENVELOPE AND SEND	SENDFILE
RESTART SEND	TRANSLATE AND SEND

SERVICESEGVAL

Indicates at which level the service segments should be validated. The service segments are the segments used when a transaction is enveloped (ISA, GS, ST, UNB, UNH, UNT, and so on). If you do not specify this keyword, no validation occurs. Valid values are:

- 1** Validates the service segments for syntax only. This includes checking for mandatory data that is missing, as well as data elements that are too large or too small.
- 2** In addition to level 1 checking, validates the service segment date and time data elements according to their types, and if a validation table has been specified, also checks the value of the data element.

Validation errors that occur during send processing will terminate processing. Validation errors that occur during receive processing cause the interchange/group/transaction with the error to be skipped (not processed) but processing continues.

This keyword is used with the following commands:

DEENVELOPE	REENVELOPE AND TRANSLATE
DEENVELOPE AND TRANSLATE	REENVELOPE
ENVELOPE	REENVELOPE AND SEND
ENVELOPE AND SEND	TRANSLATE AND ENVELOPE
RECEIVE AND DEENVELOPE	TRANSLATE AND SEND

SETCC

Specifies the condition codes used by the DataInterchange Utility to override the utility condition codes specified in the IFCC keyword. You can have up to 10 override condition codes separated by commas. If you specify the IFCC keyword, you must specify this keyword. If this keyword is omitted, all condition codes specified on the IFCC keyword are overridden to zero (0). If a particular code is omitted in the SETCC keyword, the related condition code in the IFCC keyword is overridden to zero (0). For more information, see “Overriding utility condition codes” on page 22.

This keyword can be used with any utility PERFORM command.

SNDDATE

The date of the previous send request for the transaction. The maximum length is 10.

This keyword is used with the following commands:

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

SNDTIME

The time of the previous send request for the transaction. The maximum length is 8.

This keyword is used with the following commands:

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

STANDALONE

Indicates whether the Remove Transactions process should operate in contention with other DataInterchange processes. In the DB2 environment, this allows DataInterchange to obtain exclusive high-level table locks which improves performance of the Remove Transactions process since DataInterchange does not have to obtain lower-level page locks. Valid values are:

- Y** The Remove Transactions process runs alone. Does not contend with any other DataInterchange process. Run the Remove Transaction process in this mode if possible.
- N** The Remove Transactions process does not run alone. Contends with other DataInterchange processes. High-level table locks are not obtained (default).

This keyword is used only with the REMOVE TRANSACTIONS command.

STDDESC

The description of the EDI standard as entered in the standards database. This field is case sensitive.

This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

STDID

The EDI standard ID as entered in the standards database.

This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

STDLV

The level of the EDI standard, for example, R1 for release 1.

This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

STDTRID

The transaction ID of the EDI standard as specified by the standard, such as 850 for an X12 purchase order. The maximum length is 8.

This keyword is used with the following commands:

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

STDVR

The version of the EDI standard, for example, V3 for version 3.

This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

STSTAT

Indicates the status of the transaction in the Transaction Store. Valid values are:

0	Active
1	Held
3	Purge-date expired
4	Purge-user request

For additional status information, refer to the *DataInterchange Administrator's Guide*.

This keyword is used with the following commands:

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

SYNTAX

Specifies the syntax type for the input data. This is a required field. Valid values are:

D	Application data
E	EDI data
X	XML data

This keyword is used only with the TRANSFORM command.

TESTMODE

Indicates whether the transactions are test, information, or production transactions. Valid values are:

Y	Test or information transactions
N	Production transactions

This keyword is used with the following commands:

TRADING PARTNER CAPABILITY DATA EXTRACT
TRANSACTION ACTIVITY DATA EXTRACT

TPID

The internal trading partner ID used by an application as entered in the trading partner send usage/rule. The maximum length is 35.

This keyword has this meaning with the following commands:

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

This keyword provides the default internal trading partner ID value, if the data format does not define a field that contains this value, or if a field is defined but contains all blanks. This also becomes the default ID value if the internal trading partner ID is blank in the C record.

This keyword has this meaning with the following commands:

TRANSLATE TO STANDARD
TRANSLATE AND ENVELOPE
TRANSLATE AND SEND

TPNICKN

The trading partner nickname for a trading partner profile member. The maximum length is 16. TPNICKN has two distinct uses:

- This keyword specifies a trading partner to receive data from, or a trading partner nickname to use when selecting Transaction Store or management reporting data.
- For the SEND and TRANSLATE AND SEND commands, this keyword is used the same as the keyword **TPNICKNESEND**. This means that the fields in the specified trading partner profile member are used for network override options (such as network charges), and will override the same fields in the mailbox (requestor) profile.



NOTE: When this keyword is used with the SEND and TRANSLATE AND SEND commands to send EDI data, the receiver's mailbox is not specified because that information is contained within the data being sent.

This keyword is used with the following commands:

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

TPNICKNESEND

Specifies a trading partner profile member to use for network override options (such as network charges), that will override the same fields in the mailbox (requestor) profile.



NOTE: When this keyword is used with the SEND and TRANSLATE AND SEND commands to send EDI data, the receiver's mailbox is not specified because that information is contained within the data being sent.

This keyword is used with the following commands:

ENVELOPE AND SEND
REENVELOPE AND SEND

TRACELEVEL

Indicates the level of tracing done during the transform process. Trace data will be returned in MVS to ddname EDIDTTRC, and in CICS to the TD queue defined for EDI standard output. You can change the TD queue to a TS queue, if needed. The value consists of a series of *Cn* values which represent the component and trace level for the component. The valid values for the component ID are:

A	All nodes
D	Deenveloper node
E	Enveloper node
M	Message broker
P	Parsers
R	Rules node
T	Transformation node
V	Validation node

The valid values for the component tracing level are:

- 0** All trace messages are ignored.
- 1** Normal tracing. Only the first 256 bytes of data in the buffer are written to the trace file.
- 2** Extended tracing. The entire contents of the buffer is written to the trace file.
- 3** Utility function tracing. Includes all the tracing done at level **2** plus additional tracing for some frequently called internal utility functions.

For example, a value of D1 V2 R2 would mean the deenveloper node (**D**) is set for normal tracing (**1**), the validation (**V**) and rules (**R**) nodes are set for extended tracing (**2**). Tracing is normally turned off except during problem determination. Activating tracing may negatively impact performance.

This keyword is used only with the TRANSFORM command.

TRANSACTION

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

- Y** Writes transaction data records
- N** Discards transaction data records (default)

For the format of these records, see “Transaction/Acknowledgment image data extract record layout” on page 288.

This keyword is used with the following commands:

ENVELOPE DATA EXTRACT
TRANSACTION DATA EXTRACT

TRERLVL

Indicates the maximum translation error level for the transactions you want to select. You can use this keyword to envelope only EDI documents that are error-free. Valid values are:

- 0** No errors
- 1** Data element errors
- 2** Data element and segment errors
- 3** Severe errors

This keyword is used with the following commands:

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

TRXCTLNO

The transaction set control number assigned by the sender to identify the transaction set to the sender. When combined with the sender ID, it identifies the transaction set to the receiver. The maximum length is 14.

This keyword is used with the following commands:

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

TRXDATE

The date the transaction was added to the Transaction Store. This is also the date on which the transaction was translated to EDI standard format. The maximum length is 10.

This keyword is used with the following commands:

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

TRXSTAT

Indicates the processing status of the transaction.

For ENVELOPE AND SEND, the default value is **21**. Valid value is:

21 Send translated

For REENVELOPE, the default values are **31**, **41**, **42**, and **43**. Valid values are:

29 Transaction detached -- send
30 Enveloped
31 Envelope error
41 Sent with errors
42 Send request error
43 Not sent network error
46 Send started status
48 Send requested
49 Sent to network
50 Accepted by network
51 Delivered by network
52 Purged by network
53 Recall requested
54 Recall request error
55 Recalled
61 Transaction accepted
62 Transaction rejected
63 Transaction accepted with errors

For TRANSLATE, the default value is **70**. For RETRANSLATE TO APPLICATION, the default value is **73**. Valid values for both are:

70 Received
72 Receive translated (RETRANSLATE TO APPLICATION only)
73 Receive translate error (RETRANSLATE TO APPLICATION only)

For UNPURGE TRANSACTIONS and REMOVE TRANSACTIONS, if no value is specified, all values are included in the selection criteria. Valid values are:

20	Send translate error
21	Send translated
29	Transaction detached -- send
30	Enveloped
31	Envelope error
41	Sent with errors
42	Send request error
43	Not sent network error
46	Send started status
48	Send requested
49	Sent to network
50	Accepted by network
51	Delivered by network
52	Purged by network
53	Recall requested
54	Recall request error
55	Recalled
61	Transaction accepted
62	Transaction rejected
63	Transaction accepted with errors
70	Received
71	Receive syntax error
72	Receive translated
73	Receive transaction error
74	Transaction detached -- rcv

This keyword is used with the following commands:

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

TRXTIME

The time the transaction was added to the Transaction Store. This is also the time at which the transaction was translated to EDI standard format. The maximum length is 8.

This keyword is used with the following commands:

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

USERID

The network user ID of the trading partner as entered in the **User ID** field of the trading partner profile. The maximum length is 32.

This keyword is used with the following commands:

NETWORK ACTIVITY DATA EXTRACT
TRADING PARTNER PROFILE DATA EXTRACT

USERPGM

Specifies the program that DataInterchange links to just before writing a record during data extract processing. Your program should return a code to DataInterchange indicating whether the record is written to the EDIQUERY file or discarded. If you do not specify this keyword, DataInterchange writes the requested records to the EDIQUERY file. For more information, see Chapter 1, "Using The Datainterchange Utility." The maximum length is 8.

This keyword is used with the following commands:

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

VERIFY

Indicates whether the status of a transaction is verified before the transaction is put into an envelope. Valid values are:

- Y** Checks for correct status before an ENVELOPE or REENVELOPE operation
- (other)** Does not check status (default)

This keyword is used with the following commands:

ENVELOPE
ENVELOPE AND SEND
REENVELOPE
REENVELOPE AND SEND

WRTCTLNO

Indicates whether trading partner control numbers are included when importing a trading partner profile member. Valid values are:

- Y** Imports control numbers with the trading partner (default).
- N** Does not import control numbers. If this is a new trading partner, the control number defaults to zeros. If this trading partner already exists, the existing control numbers are not overwritten by the control numbers in the import file.

This keyword is used only with the IMPORT command.

XML

Indicates whether XML processing is required for the input data. Valid values are:

- Y** Requires XML processing
- N** Does not require XML processing (default)

This keyword is used with the following commands:

DEENVELOPE	TRANSLATE AND ENVELOPE
DEENVELOPE AND TRANSLATE	TRANSLATE TO APPLICATION
ENVELOPE	TRANSLATE TO STANDARD

XMLDICT

Identifies the PDS or HFS path for the XML dictionary files generated by the DTD conversion utility. This field is required for XML processing. The maximum length is 64.

This keyword is used with the following commands:

DEENVELOPE	TRANSLATE AND ENVELOPE
DEENVELOPE AND TRANSLATE	TRANSLATE TO APPLICATION
ENVELOPE	TRANSLATE TO STANDARD

XMLDTDS

Identifies the PDS or HFS path for the XML DTD members. Applies only to MVS. This field is required for XML DTD processing. The maximum length is 64.

This keyword is used with the following commands:

DEENVELOPE	TRANSLATE AND ENVELOPE
DEENVELOPE AND TRANSLATE	TRANSLATE TO APPLICATION
ENVELOPE	TRANSLATE TO STANDARD
TRANSFORM	

For more information about XML DTD resolution, see “XML special considerations” on page 573.

XMLBCDIC

Indicates whether the incoming XML data should be interpreted as EBCDIC data. Valid values are:

- Y** Interprets incoming XML data as EBCDIC data, regardless of the encoding on the XML declaration
- N** Uses the encoding on the XML declaration

This keyword is used with the following commands:

DEENVELOPE	TRANSLATE TO APPLICATION
DEENVELOPE AND TRANSLATE	TRANSFORM

XMLSEGINP

Indicates whether record boundaries in the input file should be treated as line breaks. This keyword is ignored when **XMLBCDIC** equals **N**.

This keyword is used with the following commands:

DEENVELOPE	TRANSLATE AND ENVELOPE
DEENVELOPE AND TRANSLATE	TRANSLATE TO APPLICATION
ENVELOPE	TRANSLATE TO STANDARD

XMLSTDID

The EDI standard ID created with the DTD conversion utility. If you specify this keyword, this standard ID is used to convert each XML document to a corresponding EDI standard envelope and transaction. If you specify this keyword, the first 8 alphanumeric characters of the root element name are used as the standard ID for each XML document. The maximum length is 8.

This keyword is used with the following commands:

DEENVELOPE	TRANSLATE AND ENVELOPE
DEENVELOPE AND TRANSLATE	TRANSLATE TO APPLICATION
ENVELOPE	TRANSLATE TO STANDARD

XMLVALIDATE

Indicates the level of validation applied to the incoming XML data. Valid values are:

- 0** Ignores external DTD references
- 1** Uses the declared DTD for processing default attributes, entity references, and so on, but does not validate the data against the DTD (default)
- 2** Fully validates the data against the DTD

This keyword is used with the following commands:

DEENVELOPE	TRANSLATE TO APPLICATION
DEENVELOPE AND TRANSLATE	TRANSFORM

File formats and DataInterchange Utility records

This chapter describes the files used by the DataInterchange Utility and the format of data within each file. Appendix C, “Using sample JCL” also contains information about the DataInterchange Utility files, and “Required utility data sets” on page 708 shows the required files for each PERFORM command.

Transaction Store input and output files

This section describes the input and output files used by the DataInterchange Utility. To indicate to DataInterchange 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 DataInterchange Utility commands that you want executed. The command language syntax is fairly free-form and, except for values associated with the following keywords, is not case sensitive:

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

For information about PERFORM commands, see Chapter 2, “DataInterchange commands and keywords”.

The DataInterchange Utility first verifies that EDISYSIN is allocated. If so, ddname EDISYSIN is used. If EDISYSIN does not exist, ddname SYSIN is used. Prior to Release 4 of DataInterchange, the command file was always allocated to ddname SYSIN. The command file is opened for read processing only, so it can be allocated as an inline data set in your DataInterchange Utility JCL. File specifications are:

Use	Input file containing DataInterchange Utility PERFORM commands.
Specified	Does not apply.
ddname	EDISYSIN or SYSIN. EDISYSIN takes precedence and is used if allocated. You can override these ddnames by specifying a DataInterchange MQSeries queue profile member to use with the MQSYSIN parameter. In CICS, the command file name and type are specified in the DataInterchange Utility control information. For detailed information, see "DataInterchange Utility control information" on page 507.
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.

DataInterchange DB2 command file (EDITSIN)

This optional file is used to contain keywords that determine whether DataInterchange/MVS should attach and/or detach DB2. For more information, see "DataInterchange/MVS and DB2 attachment" on page 294. This file applies only to MVS DB2 installations; it does not apply to CICS installations. File specifications are:

Use	Input file containing information that tells the DataInterchange Utility whether or not to attach and/or detach DB2.
Specified	Does not apply.
ddname	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 DataInterchange Utility using EXEC PGM=EDIFFUT or EXEC PGM=IKJEFT01. For the most part, if EXEC PGM=EDIFFUT is used, EDITSIN will contain the DB2 control information. If EXEC PGM=IKJEFT01 is used, SYSTSIN will contain the DB2 control information.

Example 1

The following example uses DB2 command file EXEC PGM=EDIFFUT.

```
//RUNDI EXEC PGM=EDIFFUT,DYNAMNBR=20,REGION=6144K
//      PARM='SYSID=DIENU APPLID=EDIFFS LANGID=ENU'
//EDITSIN DD *
SYSTEM(DB93) PLAN(EDIENU41) 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=EDIENU41') PLAN(EDIENU41) END
/*
```

Network commands file (NETOP)

The network commands file contains the commands that you want DataInterchange to pass to the network. DataInterchange 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 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 DataInterchange to pass to the network.
Specified	Specific members are specified in the Network cmds file field of the trading partner or mailbox (requestor) profiles.
ddname	EDINTCMD.
Suggested format	Record format: Fixed or variable. Record length: 80 bytes.
Remarks	Network commands can contain variables that are resolved by DataInterchange before the commands are passed to the network. If the records exceed the file record length, they will be truncated.

Application file

The application file contains the input records that DataInterchange uses to translate application data into an EDI standard format during the send process, and the output records DataInterchange 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 DataInterchange Utility might return. If you request raw data records, the DataInterchange Utility writes the information and other optional records to the exception file (FFSEXP). The DataInterchange 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.
ddname	User-defined.
Suggested format	Record format: Fixed or variable. Record length: <ul style="list-style-type: none"> • For raw data, use maximum structure size. • For C and D records, use maximum structure size plus 17 bytes, or 1024 bytes, whichever is greater. • For I records, use 483 bytes.
Remarks	Records are truncated if record length is not large enough.

Envelope file

For outbound documents, the envelope file is either the file specified as the **Trans data queue** in the network profile member, or an override file specified in a command that requests enveloping. For inbound documents, this is either the **Receive file name** specified in the mailbox (requestor) profile member, or an override file specified in a command that requests receiving. File specifications are:

Use	Sending: Holds complete envelopes when enveloping takes place. Receiving: Contains envelopes to be deenveloped and/or translated.
Specified	Sending: TD queue in network profile or overridden by the FILEID specified on the command. Receiving: Receive file in mailbox (requestor) profile or overridden by the FILEID specified on the command.
ddname	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	<p>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.</p> <p>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 “DataInterchange Utility records format” on page 257 for a description of C and D records and the RAWDATA format.</p>



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, DataInterchange will write translated transactions to the exception file if it cannot open the file intended to receive them, or if a file name is not provided. When translating to EDI standard format, DataInterchange will write the transactions to this file that were not translated successfully. Optional records are also written to this file if the tracking file (FFSTRAK) does not exist.

This file must be large enough to contain the largest data record you are sending or receiving, and the largest information record the translator might return. DataInterchange 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	<p>Sending: Holds transactions that were not translated successfully and the unidentified and optional records. For more information, see “Tracking file (FFSTRAK)” on page 178.</p> <p>Receiving: Holds transactions that could not be written to the application file, and optional records if the application file contains raw data. For more information, see “Optional records” on page 271.</p>
Specified	Does not apply.

ddname	<p>FFSEXCP. FFSEXCP takes precedence and is used if allocated. You can override this ddname and use an MQSeries queue instead. You can use the MQEXCP parameter to specify a DataInterchange MQSeries queue profile member to use instead of a sequential file.</p> <p>In CICS, the exception file name and type are specified in the DataInterchange Utility control information. For more information, see “DataInterchange Utility control information field descriptions” on page 509.</p>
Suggested format	<p>Record format: Fixed or variable. Record length:</p> <ul style="list-style-type: none"> • 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, DataInterchange writes the optional records to the tracking file if it exists. This file must be large enough to contain the largest information record that DataInterchange might return. File specifications are:

Use	<p>Sending: Holds optional records. For more information, see “Optional records” on page 271.</p> <p>Receiving: Does not apply.</p>
Specified	Does not apply.
ddname	<p>FFSTRAK (optional). FFSTRAK takes precedence and is used if allocated. You can override this ddname and use an MQSeries queue instead. You can specify a DataInterchange MQSeries queue profile member to use instead of a sequential file with the MQTRAK parameter.</p> <p>In CICS, the tracking file name and type are specified in the DataInterchange Utility control information. See “DataInterchange Utility records format” on page 257.</p>
Suggested format	<p>Record format: Fixed or variable. Record length:</p> <ul style="list-style-type: none"> • 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, DataInterchange 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, DataInterchange does not write the optional records.

Print file (PRTFILE)

The print file must allow for a minimum record size of 132 bytes. DataInterchange 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 DataInterchange Utility showing the results of processing.
Specified	Does not apply.
ddname	PRTFILE. PRTFILE takes precedence and is used if allocated. You can override this ddname and use an MQSeries Queue instead. You can specify a DataInterchange MQSeries Queue profile member to use instead of a sequential file with the MQPRT parameter. In CICS, the print file name and type are specified in the DataInterchange Utility control information. For more information, see “DataInterchange Utility records format” on page 257.
Suggested format	Record format: FBA or VBA. Record length: 132 bytes.
Remarks	Status of processing and any errors DataInterchange encounters are written to this file during DataInterchange Utility processing.

The following is a sample PRTFILE.

```
Audit Trail Report  -DataInterchange Utility-  Date: 01/10/13   Time: 11:10:22   Page:
0001
Interchange Control Number = 000000000000057
FF0010 Transaction number 1 to 1 translated successfully
FF0013 Transactions with Interchange Control Number 000000000000057 were successfully
queued
Message: TR0004 Severity: 04
Code in ID type field not found in validation table. Internal Trading Partner ID and
Application Format = DLGTYPES - DLGTYPES. Transaction handle, code, mode, and function =
20011013111038000000 - DL1 - PRODUCTION - SEND. Interchange, group, and transaction
control numbers = 000000056 - 56 - 0155. Current Loop-ID and repetitions = 110000 - 1 - 1.
Standard segment and field ID = SEG3(003) - 3 - 1 - 11. Application field ID = SEG3 - 61.
Data type and value = Z -123456. Validation table name = DLGP2V.
Interchange Control Number = 000000000000056
FF0011 Transaction number 2 translated with errors
Message: TR0004 Severity: 04
Code in ID type field not found in validation table. Internal Trading Partner ID and
Application Format = DLGTYPES - DLGTYPES. Transaction handle, code, mode, and function =
20011013111039000000 - DL1 - PRODUCTION - SEND. Interchange, group, and transaction
control numbers = 000000056 - 56 - 0156. Current Loop-ID and repetitions = 110000 - 1 - 1.
Standard segment and field ID = SEG3(003) - 3 - 1 - 11. Application field ID = SEG3 - 61.
Data type and value = Z -123456. Validation table name = DLGP2V.
```

Report file (RPTFILE)

DataInterchange opens the report file for output for the first report and as EXTEND for successive reports. Use the JCL DISP options to control whether the file is cleared or appended to during the first use. File specifications are:

Use	Contains reports requested during Transaction Store processing.
Specified	Does not apply.
ddname	RPTFILE. RPTFILE takes precedence and is used if allocated. You can override this ddname and use an MQSeries Queue instead. You can specify a DataInterchange MQSeries Queue profile member to use instead of a sequential file with the MQRPT parameter. In CICS, the report file name and type are specified in the DataInterchange Utility control information. For more information, see “DataInterchange Utility records format” on page 257.
Suggested format	Record format: FBA or VBA. Record length: 132 bytes.
Remarks	Reports requested from the Transaction Store Facility or DataInterchange 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 DataInterchange 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:

PERFORM command	Record layout on page:
QUERY	62
ENVELOPE DATA EXTRACT	282
NETWORK ACTIVITY DATA EXTRACT	276
TRADING PARTNER CAPABILITY DATA EXTRACT	276
TRADING PARTNER PROFILE DATA EXTRACT	276
TRANSACTION ACTIVITY DATA EXTRACT	276
TRANSACTION DATA EXTRACT	282

File specifications are:

Use	Output file for the QUERY command and the data extract commands listed above.
Specified	Does not apply.
ddname	<p>EDIQUERY (required for commands that generate output). You can override this ddname and use an MQSeries queue instead. You can use the MQQUERY parameter specify a DataInterchange MQSeries Queue profile member to be used instead of a sequential file.</p> <p>In CICS, the query file name and type are specified in the DataInterchange Utility control information. For more information, see “DataInterchange Utility records format” on page 257.</p>
Suggested format	<p>Record format: Fixed or variable.</p> <p>Record length: 32756 to ensure the largest expected record is not truncated.</p>
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 DataInterchange during send and translation processing. DataInterchange opens this file for output only, and it should always be empty.



NOTE: In the CICS environment, this file is handled internally by DataInterchange.

File specifications are:

Use	<p>Sending: Holds the current transaction for transfer to the exception file if translation is not successful.</p> <p>Receiving: Does not apply.</p>
Specified	Does not apply.
ddname	<p>FFSWORK (required).</p> <p>External specification is not made in CICS. DataInterchange takes care of this file internally.</p>
Suggested format	<p>Record format: Variable blocked (VB).</p> <p>Record length: 32756 bytes.</p>
Remarks	A temporary file used only during send and translation processing. This file is used extensively as a temporary file during translate-to-EDI-standard operations and is a prime candidate for a virtual input/output data set.

Pageable translation work file (EDIVAX)

The pageable translation work file is a temporary work file used by DataInterchange when pageable translation is enabled and virtual storage usage for EDI or application data reaches 28 MB. Pageable translation is enabled by using the **PAGE** keyword on Utility PERFORM commands and, for API applications, by setting the **VAXFLAG** field in the TRCB to **X**. Enabling pageable translation will ensure that the virtual storage used for EDI and application data does not exceed 28 MB by paging any excess data to the EDIVAX file.

The amount of space allocated to this file depends on the maximum amount of data to be translated. You can calculate the amount of needed space by adding the following components:

Number of bytes in largest interchange	+	Number of bytes in largest application transaction image	+	4 MB overhead	+	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 DataInterchange can page with pageable translation is approximately one gigabyte (specifically, 32,000 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.
ddname	EDIVAX (required for pageable translation).
Suggested format	DCB statement should not be specified. (This is under DataInterchange control.)
Remarks	Omit the data set name to allow MVS to assign a temporary data set name.

Enveloping options file for functional acknowledgments (FAENV)

The enveloping options file is an optional QSAM file that you can use if you need more flexibility in the enveloping of functional acknowledgments.



NOTE: For this file, a functional acknowledgment is an ANSI X12 997 transaction, a UCS 999 transaction, or an EDIFACT CONTRL message.

With this file you can specify:

- Interchange and group envelope overrides for functional acknowledgments
- A standard profile member to fill envelope data elements other than the sender and receiver IDs

To use the file, specify ddname 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.
ddname	FAENV (optional) for MVS. 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 deenvolving to control the data used to build the enveloping (service) segments, such as ISA and GS segments, for the functional acknowledgment being returned to your trading partner.

The search key for an entry is ISID, IRID, GSID, and GRID, corresponding to the inbound envelope that is being received or deenvolved. 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, DataInterchange 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 *DataInterchange Administrator's Guide*.

FAENV field descriptions

The following fields are optional.

Field	Maximum length	Description
ISID	35	Interchange sender ID from the inbound envelope
IRID	35	Interchange receiver ID from the inbound envelope
GSID	35	Group sender ID from the inbound envelope
GRID	35	Group receiver ID from the inbound envelope
ISIDFA	35	Interchange sender ID override for outbound functional acknowledgment envelope
IRIDFA	35	Interchange receiver ID override for outbound functional acknowledgment envelope
GSIDFA	35	Group sender ID override for outbound functional acknowledgment envelope
GRIDFA	35	Group receiver ID override for outbound functional acknowledgment envelope
EPM	8	Standard envelope profile member name

FAENV file format

The following is the format of an entry in the FAENV file:

```
ISID,IRID,GSID,GRID = ISIDFA,IRIDFA,GSIDFA,GRIDFA>EPM
```

Where:

This Character: Separates:

,	Envelope fields and indicates that a value is not listed
=	Key fields from override fields
>	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 with the following envelope values:

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
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 following values:

FA interchange:	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 following separate envelopes:

FA interchange 1:	ISA06	MYDUNS#DIV1
	ISA08	TPDUNS#
	GS02	MYDUNS#DIV1
	GS03	TPDUNS#
FA interchange 2:	ISA06	MYDUNS#DIV2
	ISA08	TPDUNS#
	GS02	MYDUNS#DIV2
	GS03	TPDUNS#
FA interchange 3:	UNB03	ACCY ACCY03
	UNB06	ACCX ACCX01
	UNG02	BOOKING
	UNGO4	ACCOUNTING

Using the FAENV file allows you to separate the functional acknowledgments into interchange and group envelopes with the values you specify.

Export/Import utility function

DataInterchange provides an Export/Import utility function for updating DataInterchange's databases in a batch environment. The Export/Import batch function uses three files:

1. Batch Control File (CTLFILE) containing the control information that describes the data being exported or imported.
2. Export/Import Files (E/I File) containing the data that is being imported or exported (in tagged or fixed format).
3. Print File (PRTFILE) containing a report on Export/Import activity.

The Export/Import utility allows you to choose one, any combination of, or all the following categories of DataInterchange data for extracting (exporting) or loading (importing) EDI data:

1. EDI standards/standard transactions
2. Data formats
3. Maps
4. Control strings
5. Profiles
6. Tables

Each category is a complete set with or without the associated objects, depending on your choice. For example, for export/import of an EDI standard, the complete set includes:

- Standard definition
- Transaction definitions
- Transaction details
- Segment definitions
- Segment details
- Data element definitions

The associated objects are:

- Envelope standards
- Validation tables (if validation is required)

The categories and their associated objects are described on the following pages.

Sample programs included in the DataInterchange product enable you to convert a fixed format (flat) file into a tagged import file for importing trading partner profile (TPPROF) members, send transaction usages/rules, and receive transaction usages/rules.

These sample programs and JCL include extensive documentation that describes how they work and how they can be used. These programs are:

- A COBOL program named EDIXF2T that is the main program executed.
- An Assembler program named EDIXTAGF, that is link edited with EDIXTAGF, and is a formatting service to create the tags.

- Sample JCL named EDIXF2T used to define the fixed flat file and to execute the conversion program EDIXF2T.

These programs and JCL are provided as source and can be modified, compiled, and link edited to suit your individual needs.

Export/Import control file (CTLFILE)

The export/import control file describes what data is being exported or imported. The control file can process multiple export or import requests.

For example, to export seven different data formats would require seven records in the control file. The control file provides the same information to the DataInterchange Utility that you enter when using the export and import panels, subject to certain member selection exceptions. For more information about the exceptions, refer to the *DataInterchange Administrator's Guide*.

The control file is specified with the **CTLFILE** keyword and is used with EXPORT and IMPORT commands. The suggested record format can be fixed or variable, but the record length should not exceed 120 bytes. You can use the **CTLTYPE** keyword to specify the type of control file. The following table describes the labels in the export/import control file.

Label	Position	Length	Type	Description
CATEGORY	1	1	Char	Transaction category
REPLACE	2	1	Char	Replace named object
KEYID	3-32	30	Char	Object ID
ASSOBJ	33-48	16	Char	Associated objects
USAGETID	51-66	16	Char	Map name for usage import
MBRNAME	81-120	30	Char	Member name for profile import, or data format name for data format import

Export/Import control file label descriptions

CATEGORY

Specifies the category of the transaction and is required for both export and import. Valid values are:

- 1 EDI standards/transaction sets
- 2 Data format dictionaries/data formats
- 3 Maps
- 4 Control strings
- 7 Profiles
- 8 Tables

- 9** All categories (import only)
- A** XML dictionaries/DTDs
- B** Global variables

REPLACE

For import only. Indicates whether to replace the same named object in the database. This field does not apply to usages/rules or to profile members. It does apply to the parent categories of entire maps and profiles. This field applies only to the primary object identified by the category code. All associated objects that are imported will overwrite duplicate entries currently in DataInterchange without issuing a warning message. Valid values are:

- 1** Replaces the named object
- (other)** Does not replace the named object (default)

KEYID

Specifies the ID of the object to be exported or imported. The value must be left-justified. For exporting EDI standard transactions, the standard dictionary name is left-justified in the first 8 bytes. For importing specific EDI standard transaction sets, the import file must contain an EDI complete standard with all transaction sets included. If no key value is present for data format dictionaries, maps, or standards, then all records are exported for the specified category.

This keyword is required for importing usages/rules. For importing other items, if you do not specify this keyword, all records for the category (specified in position 1) are processed.

ASSOBJ

For export only. Identifies an array specifying the associated objects. The array must contain either **1** (Yes) or any other value (No) for each associated object in the following order:

For EDI standards:

1. Validation tables
2. Envelope profiles
3. Envelope standards

For maps and control strings:

1. Usages/Rules
2. Control strings
3. Standard transactions
4. Data formats
5. Validation tables
6. Translation tables
7. User exit routines
8. Trading partner profiles
9. Translation exit routines
10. Network security profiles
11. Network profiles
12. Network command profiles

- 13. Envelope profiles
- 14. Envelope standards
- 15. Maps
- 16. Conversion of prior-release objects



NOTE: There are no associated objects for XML DTDs or mapping global variables.

To export a usage/rule without its map, specify **1** for usages/rules (item 1) and **0** for maps (item 15).

For import only. The array must contain a **1** (Yes) and any other value (No) for each associated object in the following order:

For maps:

- 1. Usages/Rules only (the **KEYID** value must not be blank.)
- 16. Conversion of prior-release objects

For data formats:

- 16. Conversion of prior-release objects

For EDI standards:

- 16. Conversion of prior-release objects

USAGETID

For import only. Specifies the map name under which to import the usages/rules. If blank, the imported map name is used.

MBRNAME

Specifies the profile member name to export or import, or a transaction ID when exporting a specific EDI standard transaction. For data format dictionaries, specifies the dictionary name to be used.

Export/Import files

The export/import files are used for output when you export DataInterchange data, or are used for input when you import DataInterchange data. Export/Import files are sequential, contain variable length records, and must be allocated with a record format of V, LRECL=8152, and BLKSIZE=8156. There are several export/import files, each associated with the category of data they hold. The ddnames are:

ddname	Category
EDIEISTD	EDI standards and standard transactions
EDIEIADF	Data formats
EDIEITBL	Tables
EDIEITPT	Maps
EDIEICST	Control strings
EDIEIPRF	Profiles

You can maintain export files and the data in those files by using the following MVS/TSO ISPF utility functions:

Use:	To:
DATASET	Allocate, rename, delete, and display export data set information
COPY	Copy export data set
DSLIST	Print, rename, delete, browse, and display export data set information



NOTE: In CICS, if you want to maintain the export files, define them as QSAM extrapartitioned TD queues at DataInterchange system generation. To perform maintenance, close the queues in CICS, and use the ISPF utility functions in the MVS/TSO environment. Then run batch JCL to execute the utility functions in the MVS/TSO environment. CICS has the export/import files allocated and the data set names cannot be shared with ISPF utilities. When export queue maintenance is complete, open the queues in CICS. If the ddname associated with the queue specifies **DISP=OLD**, the queue is cleared when opened. If the ddname associated with the queue specifies **DISP=MOD**, the queue is not cleared when opened, and new export data is appended to the end of the queue.

DataInterchange requires that the records contained in each Export/Import file follow a certain sequence:

- The first record of the Export/Import file must be the Common Control Record (**0C1** or **0C2**). This record contains information such as the date, time and system release level.
- The last record of each group must be the Common End of Group Record (**000**).
- The data records are placed between the **0C1/0C2** record and the first **000** record, and also between each **000** record and any subsequent **000** records. These records vary in purpose and length, and they must be written in the format described in the tables on page 191 and page 192.

The following example shows the set of records written to the EDIEIPRF ddname by this export.

Export/Import common control record	0C1
First trading partner profile header record	7P1
First trading partner profile detail record	7P2
Second trading partner profile detail record	7P2
Third trading partner profile detail record	7P2
Fourth trading partner profile detail record	7P2
Fifth trading partner profile detail record	7P2
Sixth trading partner profile detail record	7P2
Last trading partner profile detail record	7P2
Export/Import common end of group record	000

The user exported seven trading partner profiles in tagged format. This resulted in DataInterchange writing out the Common Control Record (0C1) and then a Trading Partner Profile Header Record (7P1) to specify that the following records are part of a trading partner profile export. This is followed by seven Trading Partner Profile Detail Records (7P2), one for each trading partner. The file ends with a Common End of Group Record (000).

Export/Import common control record (0C1/0C2)

The Common Control Record defines the format of the records (tagged or fixed), the user ID, date, time, and DataInterchange version/release number. This record is required and it must be the first record in the Export/Import record data set.



NOTE: The first 3 bytes of this record are **0C1** or **0C2**, starting in column 1. The 0C1/0C2 record occurs only once in an export/import data set and is always the first record.

Field Name	Position	Length	Type	Field Description
Category	1-1	1	Char	Export/Import record category = 0
Rectype	2-3	2	Char	Export/Import record type code Record type = C1 (tagged format) Record type = C2 (fixed format)
User ID	4-11	8	Char	User ID of person creating the export/import record data set
Date	12-17	6	Char	Creation date of export/import data set (YYMMDD)

Field Name	Position	Length	Type	Field Description
Time	18-23	6	Char	Creation time of export/import data set (HHMMSS)
Language	24-26	3	Char	DataInterchange language English (United States) = ENU
DIVersRel	27-38	12	Char	DataInterchange version and release Vers 3 Rel 1 = 010301000000 Vers 4 Rel 1 = 010401000000

The following example of a completed 0C1 record depicts the user ID as SMITH, a creation date of **010419**, a creation time of **161616**, English as the language, and Version 4 Release 1 (**010401000000**) as the DataInterchange version and release.

```
0C1 SMITH 010419161616ENU010401000000
```

Export/Import common end of group record (000)

The Common End of Group Record indicates the end of an export/import group of records. This record is required and must be the last record of an export/import group. The following table describes the fields of the common end of group record.

Field Name	Position	Length	Type	Field Description
Category	1-1	1	Char	Export/Import record category = 0
Rectype	2-3	2	Char	Export/Import record type = 00

The Common End of Group Record contains only **000**, starting in column 1.

Export/Import file data area

The content data area varies depending upon which category of data you are importing or exporting. The following categories are supported:

- EDI standards and standard transactions
- Data formats
- Maps
- Control strings
- Profiles
- Tables

Regardless of the category chosen, the data portion (tagged or fixed) of an export/import file starts in column 4 of the record. Columns 1-3 are reserved for the Export/Import record ID (**RECID**). As discussed earlier, the Common Control Record (0C1/0C2) and Common End of Group Record (000) are fixed records. All records contained in the export/import file data area may be either tagged or fixed format records, except for the control string (G1) and document layout (L1), which are native database format.



NOTE: When fixed format record layouts are changed due to design changes in DataInterchange, new fields are added to the end of the records.

The following tables define the order in which the fields are exported (position of fixed format fields), the length of fixed format fields (maximum length for tagged fields), and the type of data. Data with type **Char** can have any valid characters. Data with type **DEC** must have only decimal characters (0-9). When exported, data with type **HEX** is expanded to two hex representation characters (0-9, A-F) for each hex byte.

Exporting and importing trading partner profiles

The trading partner profile contains information about the companies that exchange business transactions with you. There must be one record for each trading partner that you wish to define.

The Export/Import format for defining trading partners includes the following record types:

One trading partner profile header record	7P1
One or more trading partner profile records	7P2
Zero or more contact records	7Z1
Zero or more trading partner contact records	7P3
One or more trading partner control number records	7P4
At most, one trading partner comment record	7A1

The trading partner profile record group must be followed by the Export/Import Common End of Group Record (000). The following example illustrates a valid sequence of records written to ddname EDIEIPRF when exporting multiple trading partner profiles.

Record type	Record name
0C1	Export/Import common control record
7P1	Trading partner header record
7P2	Trading partner profile record
7P4	Trading partner control number record
7A1	Trading partner comment record
7P2	Trading partner profile record
7Z1	Contact record
7Z1	Contact record
7P3	Trading partner contact record
7P3	Trading partner contact record
7P3	Trading partner contact record
7P4	Trading partner control number record
7A1	Trading partner comment record
7P2	Trading partner profile record
7Z1	Contact record
7P3	Trading partner contact record

Record type	Record name
7P4	Trading partner control number record
7P2	Trading partner profile record
7P4	Trading partner control number record
7P2	Trading partner profile record
7P4	Trading partner control number record
000	Export/Import common end of group record

Importing new definitions to DataInterchange

After the Export/Import file data set has been updated, you can submit the updates using the batch utility or with the DataInterchange Import function.

Export/Import trading partner profile header record (7P1)

The trading partner profile header record is required and must be the first record of a trading partner profile record group. A trading partner profile group is defined as one trading partner profile header record (7P1) followed by one or more trading partner profile records (7P2).

The trading partner profile header record contains the trading partner profile indicator and profile description.

Profile definition header record (7P1)

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = TPPROF
PROFDESC	12-46	35	Char	Profile/table description

Export/Import trading partner profile record (7P2)

The trading partner profile record defines the descriptive and control data that identify an individual trading partner and its processing options. Multiple 7P2 records can be created (one record per trading partner) to support the creation of multiple trading partner profiles.

For fixed format records, the fields must be in the position specified and for the length specified in the table.

For tagged files, the tags for the trading partner profile record (7P2) can be placed in any column or any order you choose. When importing data, all incoming variables must be delimited by parentheses. When exporting, outgoing variable data is placed between parentheses. Profile member tags and their associated maximum record lengths are listed in "Trading partner profile member (TPPROF-P2)" on page 195.

Trading partner profile member (TPPROF-P2)

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID
TPNICKN	12-27	16	Char	Trading partner nickname
NETID	28-35	8	Char	Network ID
SYSQUAL	36-36	1	Char	System qualifier
SYSID	37-44	8	Char	System ID
ACCTID	45-76	32	Char	Account number
USERID	77-108	32	Char	User ID
INTQUAL	109-112	4	Char	Interchange ID qualifier
INTID	113-147	35	Char	Interchange ID
CONAME	148-187	40	Char	Company name
ADDR1	188-227	40	Char	Company address line 1
ADDR2	228-267	40	Char	Company address line 2
PHONE	268-292	25	Char	Contact phone
CONTACT	293-322	30	Char	Contact name
SNDPASS	323-336	14	Char	Interchange send password
RCVPASS	337-350	14	Char	Interchange receive password
SECPROF	351-358	8	Char	Network security profile ID
NETCLS	359-359	1	Char	Network message class
NETCHG	360-360	1	Char	Network charges code
NETACK	361-361	1	Char	Network acknowledgment
NETVCHK	362-362	1	Char	Destination verification code
NETRETN	363-365	3	Char	Retention period
NETEDIO	366-366	1	Char	EDI receive option
NETEDIP	367-367	1	Char	EDI processing override
STGFMTOV	368-368	1	Char	Storage format override
MACHTYPE	369-369	1	Char	Machine type
STGFMT	370-370	1	Char	Storage format
EOTID	371-371	1	Char	End of text/message delimiter
LOGENV	372-372	1	Char	Log standard data
FNCGRP	373-373	1	Char	Functional group code
SEDLM	374-375	2	Char	Subelement delimiter
DEDLM	376-377	2	Char	Data element delimiter

Tag	Position	Length	Type	Description
SEGDLM	378-379	2	Char	Segment delimiter
SEGSEP	380-381	2	Char	Segment ID separator
DECNOT	382-382	1	Char	Decimal notation
RLSCHAR	383-384	2	Char	Release character
INTCTLNO	385-393	9	Char	Interchange mask
GRPCTLNO	394-402	9	Char	Group mask
TRXCTLNO	403-411	9	Char	Transaction mask
COMMENT1	412-451	40	Char	Comment line 1
COMMENT2	452-491	40	Char	Comment line 2
NETCMDS	492-499	8	Char	Network commands file
TPDATAINE	500-531	32	Char	Trading partner data phone number
TIMEOUT	532-535	4	Dec	Data line communication timeout
SEGMENTED	536-536	1	Char	Segmented output option
SUFFIX	537-538	2	Char	File suffix
TPENVSUF	539-540	2	Char	Envelope profile suffix
TPGENRCV	541-541	1	Char	Allow generic receive
TPCMPRES	542-542	1	Char	Compression
TPSUPAD3	543-582	40	Char	Address line 3
TPSUPCTY	583-612	30	Char	City name
TPSUPST	613-614	2	Char	State code
TPSUPPST	615-629	15	Char	Postal code
TPSUPCON	630-659	30	Char	Country
TPSUPFAX	660-684	25	Char	Fax number
TPSUPU3	685-724	40	Char	Comment line 3
TPSUPU4	725-764	40	Char	Comment line 4
TPSUPU5	765-804	40	Char	Comment line 5
TPSUPU6	805-844	40	Char	Comment line 6
TPSUPU7	845-884	40	Char	Comment line 7
TPSUPU8	885-924	40	Char	Comment line 8
TPSUPU9	925-964	40	Char	Comment line 9
TPSUPU10	965-1004	40	Char	Comment line 10
DESCRIPT	1005-1034	30	Char	Description
TPTYPE	1035-1035	1	Char	Trading partner type

Tag	Position	Length	Type	Description
PRIORITY	1036-1036	1	Char	Expedite priority
PROCESS	1037-1076	40	Char	Process ID
DESEP	1077-1078	2	Char	Repeating data element separator

The following example shows part of an export/import file that contains records for exporting/importing three trading partner profiles.

```

0C1SMITH 0140419161616ENU010103000000
7P1PROFID(TPPROF) PROFDESC(Trading partner profile)
7P2PROFID(TPPROF) TPNICKN(IBM) NETID(IINR4) SYSQUAL(4) NETACK(N)PASSWORD(N
WPASS)
7P4TPNICKN(IBM) APPRECID(JONES) APPRECQ(01) INTCTLNO(000010000)
GRPCTLNO(000015000) TRXCTLNO(000020000)
7P2PROFID(TPPROF) TPNICKN(TPT40) NETACK(Y) SYSQUAL(4) NETID(IINR5) PHONE(555-
1234)
7P2PROFID(TPPROF) TPNICKN(VAN04PR) CONTACT(John Doe) SNDPASS(Y) RCVPASS(Y)
7P4TPNICKN(VAN04PR) APPRECID(SMITH) APPRECQ(02) APPDOCID(850) INTCTLNO(000030001)
GRPCTLNO(000040000) TRXCTLNO(000045000)
000

```

The above example shows how the 0C1 record is the first record in the Export/Import file, followed by profile data records with a 000 record as the last record. The 7P1 and 7P2 fields are the record IDs (RECIDs). They must be the first 3 bytes of the data record. In this example, the **7** indicates the category code, the **P** indicates the object being exported/imported (in this case, Profile), and the **1**, **2**, and **4** are internal DataInterchange indicators.

Trading partner profile member field descriptions

Category code

Must contain **7**.

Record type

Must contain **P2**.

Profile ID (PROFID)

Must contain **TPPROF**.

Trading partner nickname (TPNICKN)

The name you use to refer to this trading partner. Use the same name throughout DataInterchange to refer to this trading partner by nickname.

Network ID (NETID)

The name that identifies the network used to communicate with this trading partner. It must match the name of a member in the network profile.

System qualifier (SYSQUAL)

For AT&T Global Network users, enter an **I** if intersystem addressing is required for this trading partner (Network reference: DTBLTYP). Enter the ID of the other system in the next field.

System ID (SYSID)

For intersystem addressing, the ID of the system responsible for the receiver's account. For AT&T Global Network users, the ID is limited to three characters (Network reference: DTBLID).

Account number (ACCTID)

The account number assigned to your trading partner by the network. The entry must be left-justified. For sending and receiving EDI in ISA/IEA envelopes, the last position must be blank. The combination of this field and the User ID field must be unique for all members in the profile.

If the interchange ID is blank, the account number and user ID make up the receiver ID in the interchange envelope. UCS (BG/EG) envelopes are an exception, where the phone number field contains the receiver ID.

User ID (USERID)

The user ID assigned to your trading partner by the network. The entry must be left-justified. The combination of this field and the account number field must be unique for all members in the profile.

If the interchange ID field is blank, the account number and user ID make up the receiver ID in the interchange envelope. UCS (BG/EG) envelopes are an exception, where the phone number field contains the receiver ID.

Interchange ID qualifier (INTQUAL)

The type of interchange ID entered in the interchange ID field such as a Dun and Bradstreet number (DUNS). The EDI standard defines these codes. If the interchange ID is blank, the enveloper takes the qualifier from the envelope profile member. The combination of this field and the interchange ID field must be unique for each member unless the fields are left blank.

Interchange ID (INTID)

The ID used as the interchange receiver ID (recipient) when you send to this partner and as the interchange sender ID when you receive from this partner. If this field is blank, the enveloper uses the account number and user ID (or phone number for BG/EG interchanges). The combination of this field and the interchange ID qualifier field must be unique for each member unless the fields are left blank.

Company name (CONAME)

The name of the trading partner's company.

Company address line 1 (ADDR1)

The first line of the trading partner's address.

Company address line 2 (ADDR2)

The second line of the trading partner's address.

Contact phone (PHONE)

The trading partner's phone number (free-form). For UCS (BG/EG) enveloping, if the interchange ID is blank, the enveloper uses the phone number as the interchange ID.

Contact name (CONTACT)

The name of the person you communicate with when dealing with this trading partner.

Interchange send password (SNDPASS)

The password agreed upon by you and your trading partner for sending to this trading partner. This value maps to the **password data type** in the interchange envelope.

Interchange receive password (RCVPASS)

The password agreed upon by you and your trading partner for receiving from this trading partner. If this value matches the interchange password (password data type) that was received, translation occurs.

Network security profile ID (SECPROF)

The ID of the default network security profile member that specifies the encryption and authentication processes that apply to EDI data for this trading partner. This profile member is always used when receiving from this partner. It is used by default when sending to this partner unless the send usage/rule for the map specifies a different member in the **Group security profile name** field or the **Trans security profile name** field.

Network message class (NETCLS)

Applies only to sending. Indicates any special status (for example, test) of the data being sent. For AT&T Global Network users (Network reference: MSGNCLS), **T** indicates test status, and a blank indicates normal status.

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Network charges code (NETCHG)

Indicates how charges are shared between sender and receiver. For AT&T Global Network users (Network reference: MSGCHRG), valid values are:

- 1** Receiver pays all charges.
- 2** Receiver pays all charges if agreed to; otherwise, charges are split between sender and receiver.
- 3** Receiver pays all charges if agreed to; if receiver does not agree to pay all charges, charges are split between sender and receiver if agreed to; otherwise, the sender pays all charges.
- 4** Charges are split between sender and receiver if agreed to; otherwise, the sender pays all charges.
- 5** Charges are split between sender and receiver.
- 6** Sender pays all charges.

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Network acknowledgment (NETACK)

Indicates which network acknowledgments (receipt, delivery, purge) you want to receive when sending to this partner. For AT&T Global Network users (Network reference: MSGRCPT), valid values are:

(blank) No acknowledgments

- R** Receipt only
- D** Delivery only
- B** Both receipt and delivery
- A** Purge only
- C** Both receipt and purge
- E** Either purge or delivery
- F** Receipt and either delivery or purge

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Destination verification code (NETVCHK)

Indicates whether the destination is verified before the data is sent. For AT&T Global Network users (Network reference: MSGVCHK), valid values are:

- N** Does not request verification (default)
- Y** Requires verification
- F** Requests verification, but sends even if the destination is not verified (useful for inter-system addressing)

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Retention period (NETRETN)

The number of days that the data is kept in the network mailbox before being purged, if it is not received.

If you are using more than one network, contact a representative from the network you are using or refer to an appropriate manual to determine the correct values to use for this field. The following pertains to the AT&T Global Network and to Expedite Base 4.2 and higher.

Valid values are **0** through **180**. For installations in the U.S., the default retention period is **30** days. For installations outside the U.S., contact your marketing representative for your default value.

If you specify **0** or blank, Information Exchange retains the data for the default period. In addition, if you specify a value larger than the maximum for your system, Information Exchange retains the data for the default period.

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.



NOTE: The size of this field for Expedite/CICS is two digits, meaning the valid values for Expedite/CICS are **0-99**. The two left-most digits entered in this field are used by Expedite/CICS as the retention period. For example: if you enter 180, the retention period value sent to Expedite/CICS will be **18**.

EDI receive option (NETEDIO)

Indicates whether you want EDI segments stored in the file as separate records. For AT&T Global Network users (Network reference: EDIOPT), valid values are:

- Y** Ends records at the segment delimiter (default)
- N** Does not end records at the segment delimiter

EDI processing override (NETEDIP)

Indicates whether you want the EDI data you receive to have special EDI processing, which breaks records by the segment delimiter. For AT&T Global Network users (Network reference: EDIPROC), valid values are:

- Y** Performs EDI processing if the common data header indicates that the data is in EDI standard format (default)
- N** Does not perform EDI processing, regardless of the values in the common data header

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

Storage format override (STGFMTOV)

If you are using more than one network, contact a representative from the network you are using to determine the available options.

Information Exchange uses common data header control information that allows users to send or receive information electronically. The control header contains information such as the type of record, original record format, sending system type, type of data being sent, whether the data is in an EDI format, and a unique record number for tracking.

If you are using IBM Expedite Base/MVS (IEBASE), refer to the **DLMOVERRIDE** command option keyword for an up-to-date list of acceptable values. Valid values are:

- Y** Formats the data according to the **DELIMITED** parameter, even if the common data header (CDH) indicates a delimiter type
- N** Ignores the value in the **DELIMITED** parameter, and formats the data according to the record type (if any) in the CDH (default)

If there is no common data header, the format indicated in the **Storage format** field is used. If your request does not specify a trading partner, the information is taken from the mailbox (requestor) profile.

Machine type (MACHTYPE)

This field is not currently supported.

Storage format (STGFMT)

Indicates to the network how data is stored for free-form messages and files. When determining what option to select, you should consider the type of data you want to send and how the file will be received. If you are using more than one network, contact a representative from the network you are using to determine the available options. Valid values are:

- C** Stores each record with a carriage return and line-feed character (CRLF) and uses the end of file (EOF) character to mark the end of file. These characters are represented and stored as hex values 0D0A and 1A respectively. This option is generally used to send files containing program source code with variable length records. Output records will include data up to the carriage return and line feed characters (CRLF).
- L** Precedes each record with a 2-byte hexadecimal record length. Select this option for sending data in fixed format or for sending binary data. The output record length is determined by the value in the 2 bytes containing the record length.
- N** Stores the data as it is received (default). Output records are built based on the record length of the data set allocated to receive the data.

If you are using Expedite/CICS, refer to the DTYPE field for an up-to-date list of valid values:

- A** Stores each record with a carriage return and line-feed character (CRLF) and uses the end of file (EOF) character to mark the end of file. These characters are represented and stored as hex values 0D0A and 1A respectively. This option is generally used to send files containing program source code with variable length records. Output records will include data up to the carriage return and line feed characters (CRLF).

- B** Precedes each record with a 2-byte hexadecimal record length. Select this option for sending data in fixed format or for sending binary data. The output record length is determined by the value in the 2 bytes containing the record length.
- E** Stores each record based on the delimiters found in the EDI standard data.
- O** Stores data as it is received (free-form).

If your request does not specify a trading partner, this information is taken from the mailbox (requestor) profile.

End of text or message delimiter (EOTID)

The character that signifies to the network the end of the data. It applies to free-form messages and data files, not to EDI standard transactions. (Network reference: EORCHAR, EOMIND).

Log standard data (LOGENV)

Indicates whether the translator should include EDI standard (untranslated) data when logging an image of a transaction. For EDI data with errors, the EDI standard data is logged even if you enter **N**.

This field can be overridden by the **Log standard data** field of the APPDEFS profile. Valid values are:

- Y** Includes EDI standard (untranslated) data when logging transaction image
- N** Does not include EDI standard (untranslated) data when logging transaction image

Functional group code (FNCGRP)

Indicates whether the envelope should create functional groups for transactions with type **E** envelopes. (Functional groups are always created for types **I**, **U**, and **X**; they are never created for type **T**). If you choose not to have functional groups, the different message types are enveloped together. Valid values are:

- Y** Creates functional groups
- N** Does not create functional groups

Subelement delimiter (SEDLM)

The character (in hex notation) that separates subelements (also called component data elements) in a transaction set. An entry here (other than 00 or 40) overrides the character specified by the EDI standard.

If entered, this character must be different from the characters specified for the data element delimiter, segment delimiter, segment ID separator, decimal notation, and release character fields, with this exception: the subelement delimiter and the data element delimiter can be the same for envelopes with types **I**, **U**, and **X**.

Data element delimiter (DEDLM)

The character (in hex notation) that separates the data elements in a transaction set. An entry here (other than 00 or 40) overrides the character specified by the EDI standard.

If entered, this character must be different from the characters specified for the subelement delimiter, segment delimiter, segment ID separator, decimal notation, and release character fields, with this exception: the subelement delimiter and the data element delimiter can be the same for envelopes with types **I**, **U**, and **X**.

Segment delimiter (SEGDLM)

The character (in hex notation) that marks the end of each segment in a transaction set. An entry here (other than 00 or 40) overrides the character specified by the EDI standard.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment ID separator, decimal notation, and release character fields.

Segment ID separator (SEGSEP)

The character (in hex notation) that separates the segment ID and the first data element in a segment. An entry here (other than 00 or 40) overrides the character specified by the EDI standard.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, decimal notation, and release character fields.

Decimal notation (DECNOT)

The character that represents the decimal point in a transaction set. For type **E** envelopes, an entry here overrides the character specified by the EDI standard. For all other types, a period represents the decimal point.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, segment ID separator, and release character fields.

Release character (RLSCHAR)

For type **E** and **T** envelopes, the character (in hex notation) that is used to indicate where a delimiter is being used as part of the data. An entry here (other than 00 or 40) overrides the character specified by the EDI standard.

If entered, this character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, segment ID separator, and decimal notation fields.

Interchange mask (INTCTLNO)

The initial reference number that the enveloper places in the **CN data type** of the interchange header and trailer. This value is used as the base value for each trading partner/receiver ID combination. This field does not represent the current control number for this trading partner. Use the **Control Number** option on the Member List panel to request control number information.

Group mask (GRPCTLNO)

The initial reference number or special codes that the enveloper places in the **CN data type** of the functional group header and trailer. This value is used as the base value for each trading partner/receiver ID combination. Use caution when updating this field. This field does not represent the current control number for this trading partner. Use the **Control Number** option on the Member List panel to request control number information.

Transaction mask (TRXCTLNO)

The initial reference number or special codes that the enveloper places in the **CN data type** of the transaction set header and trailer. This value is used as the base value for each trading partner/receiver ID combination. Use caution when updating this field. This field does not represent the current control number for this trading partner. Use the **Control Number** option on the Member List panel to request control number information.

Comment line 1 (COMMENT1)

Free-form notes about the trading partner.

Comment line 2 (COMMENT2)

Free-form notes about the trading partner.

Network commands file (NETCMD5)

The name of a PDS member to be allocated the MVS ddname EDINTCMD. This member will contain the commands that you want to pass to the network.

DataInterchange reads the commands from the PDS member and writes the commands to the network input file specified in the network profile member after all substitutable variable tags are resolved by DataInterchange.

TP data phone number (TPDATALINE)

The phone number to dial to connect to your trading partner directly. This is the number used by your computer to talk directly to your trading partner's computer. The Contact phone field contains the voice phone number to call a human (not a computer) at your trading partner's site.

Data line communication timeout (TIMEOUT)

Specifies the maximum allowable time that the communications line can be idle without dropping the connection. If the data line is idle for a time greater than the value entered, then the line is dropped. If you specify a trading partner when requesting network activity, the value for this field is taken from the trading partner profile. Otherwise, the value for this field is taken from the mailbox (requestor) profile.

Segmented output option (SEGMENTED)

Indicates whether you want EDI segments stored in the output file as separate records. Valid values are:

- Y** Ends records at the segment delimiter
- N** Does not end records at the segment delimiter (default)

File suffix (SUFFIX)

A 2-character suffix for the ddname used to store the results of a fixed-to-fixed translation. The ddname is taken from the **Application file name** field of the target data format.

This 2-character suffix is appended to the **Application file name** or, if the file name is longer than 6 characters, overwrites the last two characters of the **Application file name**. Use this suffix to separate the results of a fixed-to-fixed translation by trading partner. If you do not want to separate results by trading partner, then either leave the suffix blank or assign all the ddnames to the same physical file.

For CICS, you can use the suffix to identify a unique TS queue for each trading partner.

Envelope profile suffix (TPENVSUF)

A 2-character suffix for a generic envelope profile member name. A generic envelope profile name has an ampersand (&) as the first character of the name followed by a 1 to 6-character common name.

This 2-character suffix is used to generate the envelope profile member name for enveloping transactions. Use this suffix to use different envelope profile members for different trading partners. To use the same envelope profile member for several trading partners, enter the same suffix in each of the trading partner profiles, or leave the suffix blank to use a common envelope profile member (with the same name as the common name in the usage/rule). You must define an envelope profile member for each combination of common name/suffix value specified in a trading partner profile.

The envelope profile member name is specified on the Send and Receive Trading Partner Usage panels. The generic form of the name is most useful for generic usages/rules but can be used for any usage/rule.

Allow generic receive (TPGENRCV)

Indicates whether transactions received from this trading partner can be translated using generic receive trading partner usages/rules. Valid values are:

- Y** Allows translation via generic receive trading partner usages/rules
- N** Does not allow translation via generic usages/rules (default)

Compression (TPCMPRES)

Indicates whether Expedite Base/MVS or Expedite/CICS should compress your data prior to transmission. Valid values are:

- N** Does not compress data (default).
- Y** Compresses data prior to transmission. If the file sent contains multiple EDI envelopes, all envelopes are sent compressed.
- T** Conditionally compresses data prior to transmission. For Expedite Base/MVS, compression will occur if the **SENDER**, **RECEIVER**, and **COMPRESS** parameters are listed in the CPLOOKUP EXPFILE file. This file defines a series of paired receivers and senders, and for each pair, indicates whether compression should be performed. For Expedite/CICS, compression will occur if the sender/receiver pair is found in the lookup table and **COMPRESS** equals **Y** in the lookup table.

For more information, refer to the *Expedite Base/MVS Programming Guide* or to *Customizing and Developing Applications with Expedite/CICS*.

If a trading partner profile member is specified on the send request, the compression value is taken from the trading partner profile. Otherwise, this information is taken from the mailbox (requestor) profile.

Address line 3 (TPSUPAD3)

The third line of the trading partner's address.

City name (TPSUPCTY)

The trading partner's city.

State code (TPSUPST)

The trading partner's state.

Postal code (TPSUPPST)

The trading partner's zip code.

Country (TPSUPCON)

The trading partner's country.

Fax number (TPSUPFAX)

The trading partner's fax number.

Comment line 3 (TPSUPU3)

Free-form remarks.

Comment line 4 (TPSUPU4)

Free-form remarks.

Comment line 5 (TPSUPU5)

Free-form remarks.

Comment line 6 (TPSUPU6)

Free-form remarks.

Comment line 7 (TPSUPU7)

Free-form remarks.

Comment line 8 (TPSUPU8)

Free-form remarks.

Comment line 9 (TPSUPU9)

Free-form remarks.

Comment line 10 (TPSUPU10)

Free-form remarks.

Description (DESCRIPT)

A free-form description of the member.

Trading partner type (TPTYPE)

Indicates the type of trading partner this definition represents. Valid values are:

- E** EDI trading partner (default). Considered to be external to your business organization. The interchange control numbers are generated using the EDI trading partner ID.
- A** Application trading partner. Considered to be internal to your business organization. For example, divisions or departments may be internal trading partners. Use this setting when no specific EDI trading partner is defined, or in combination with an EDI trading partner. Interchange control numbers are generated using the application and EDI trading partner combination. Use this trading partner type with centralized EDI when multiple application trading partners do business with the same EDI trading partner.
- B** Both EDI trading partner and application trading partner. Considered to be both external (EDI) and internal (application). Use this setting when your organization provides EDI translation services to customers who are trading partners. The value in this field is directly related to the usage/rule setup.

For more information, refer to the *DataInterchange Administrator's Guide*.

Priority (PRIORITY)

Indicates whether messages are sent through Expedite Base/MVS or Expedite/CICS with normal delivery or priority delivery. Valid values are:

- (blank)** Normal delivery
- P** High priority

For more information, refer to the SEND commands in the *Expedite Base/MVS Programming Guide* or to *Customizing and Developing Applications with Expedite/CICS*.

If a trading partner profile member is specified on the send request, this value is taken from the trading partner profile. Otherwise, this information is taken from the mailbox (requestor) profile.

Process

The process name used with data transformation to associate with this trading partner, such as PRODUCTION_PURCHSG_V1R1. You can create data transformation rules for a group of maps (creating a process) and then associate trading partners with the process. If you change the map rules for the process, all trading partners associated with the process use the changed maps automatically.

In the same manner, you can use the process name to create classes of trading partners such as production, non-production, financial institutions, XML trading partners, or EDI trading partners. You can create data transformation map rules to associate with each trading partner class.

Repeating data element separator (DESEP)

The hex character separator placed between repeating data elements for all transactions sent to this trading partner. A value in this field (other than 00 or 40) overrides the separator character defined in the EDI standard. This character must be different from the characters specified for the subelement delimiter, data element delimiter, segment delimiter, segment ID separator, decimal notation, and release characters. This hex character should not be found within the EDI standard data.

Trading partner contact definition (7P3)

Trading partner contact information is exported or imported automatically when a trading partner record is exported or imported. Each trading partner contact record specifies a contact that is related to a trading partner. The fields are described below.

Tag	Position	Length	Type	Description
TPNICKN	4-19	16	Char	Trading partner nickname
TPCONNM	20-59	40	Char	Trading partner contact name

Trading partner control numbers (7P4)

Trading partner control numbers are exported or imported automatically when a trading partner is exported or imported. Each trading partner control number specifies a control number that is related to a trading partner. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPNICKN	4-19	16	Char	Trading partner nickname
APPRECID	20-54	35	Char	Receiver ID
APPRECQ	55-58	4	Char	Receiver qualifier
APPDOCID	59-66	8	Char	Transaction ID, message ID, or document definition name
INTCTLNO	67-75	9	Char	Interchange control number
GRPCTLNO	76-84	9	Char	Group control number
TRXCTLNO	85-93	9	Char	Transaction control number

Comments definition (7A1)

Comments are exported or imported automatically when a trading partner is exported or imported. The comment record specifies textual data about a trading partner. The fields are described in the table below.

Tag	Position	Length	Type	Description
CMTTYPE	4-11	8	Char	Comments entity type
CMTKEY	12-46	35	Char	Comments entity type key
CMTTEXT	47-2094	2048	Char	Trading partner contact comments

Contact definition (7Z1)

Contact information is exported or imported automatically when a trading partner record is exported or imported. The contact record specifies data about a contact. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPCONNM	4-43	40	Char	Trading partner contact name
TPCONTLE	44-83	40	Char	Trading partner contact job title
TPCONAD1	84-123	40	Char	Trading partner contact address line 1
TPCONAD2	124-163	40	Char	Trading partner contact address line 2
TPCONAD3	164-203	40	Char	Trading partner contact address line 3
TPCONCTY	204-233	30	Char	Trading partner contact city
TPCONST	234-235	2	Char	Trading partner contact state
TPCONPST	236-250	15	Char	Trading partner contact postal code
TPCONCON	251-280	30	Char	Trading partner contact country
TPCONPHN	281-305	25	Char	Trading partner contact phone
TPCONFAX	306-330	25	Char	Trading partner contact fax number
TPCONOTH	331-355	25	Char	Trading partner contact other phone
TPCONEML	356-395	40	Char	Trading partner contact e-mail address
CMTTEXT	396-2443	2048	Char	Trading partner contact comments

Export and importing EDI standard records

EDI standards can be exported/imported using tagged or fixed format files. The EDI standard/transaction set includes:

1Y1	Dictionary Record
1Y2	Transaction Header Record
1Y3	Transaction Detail Record
1Y4	Segment Header Record
1Y5	Segment Detail Record
1Y6	Data Element Header Record
1Y7	Data Element Detail Record
1Y8	Segment Notes Record
1Y9	Transaction Notes Record
1YA	Composite Data Element Notes Record
1YB	Envelope Detail Record

Three types of associated objects can be exported using tags:

1. Validation tables
2. Envelope profile
3. Envelope standard

In the example below, you can export all of the data elements in a particular segment. Since a segment is made up of many data element usages/rules and definitions, multiple Y6 and Y7 records are shown here.

Record Type	Record Name
0C1	Export/Import common control record
1Y1	EDI standard dictionary record
1Y2	EDI standard transaction header record
1Y3	EDI standard transaction detail record
1Y4	EDI standard segment header record
1Y5	EDI standard segment detail record
1Y6	EDI standard data element header record
1Y7	EDI standard data element detail record
1Y6	EDI standard data element header record
1Y7	EDI standard data element detail record
1Y8	EDI standard segment notes record
1Y9	EDI standard transaction notes record

Record Type	Record Name
1YA	EDI standard composite data element notes record
1YB	EDI standard envelope details record
000	Export/Import common end of group record

The following example shows part of an export/import file data set that contains records for exporting/importing one transaction of an EDI standard. The transaction has one segment with three data elements.

```

0C1ADMIN  011116151224ENU010401000000
1Y1 STDID(AARV4R4) AGENCY(X) VERSION(40) RELEASE(40) INDUSTRY(RAIL)
DICTENVFLAG(D) INTENV(X) STDDESC(Rail Carrier 4040) ALLTRXS(N)
1Y2 STDID(AARV4R4) STDTRID(426) FNGRPID(SR) STDDESC(Rail Revenue Waybill)
STDPURP(Purpose)
1Y3 STDID(AARV4R4) STDTRID(426) TABLENO(1) POSNO(20) SEGID(ZR) SEGREQ(M)
SEGMAX(1) UNLIMMAX(0) UNLPREP(0)
1Y4 STDID(AARV4R4) SEGID(ZR) STDDESC(Waybill Reference Identification)
STDPURP(To transmit identity and reference information of the waybill)
1Y5 STDID(AARV4R4) SEGID(ZR) POSNO(1) DEID(762) DEREQ(M)
1Y6 STDID(AARV4R4) DEID(762) DETYPE(ID) DEMIN(1) DEMAX(1)
CODELIST(762R440) STDDESC(Waybill Response Code) STDPURP(Code indicating
a waybill response)
1Y9 STDID(AARV4R4) STDTRID(426) TABLENO(1) POSNO(241) NOTETYPE(N)
NOTEPARA(1) RELATION(CM) STDNOTE(One R2B segment is needed for each line
haul carrier)
1Y8 STDID(AARV4R4) SEGID(ZR) POSNO(4) NOTETYPE(N) NOTEPARA(1) RELATION(P)
REL1(4) REL2(5)
000

```

EDI standard dictionary record (1Y1)

This record defines the EDI standard being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
AGENCY	34-41	8	Char	Agency issuing the standard
VERSION	42-49	8	Char	Version of the standard
RELEASE	50-57	8	Char	Release level of the standard
INDUSTRY	58-65	8	Char	Industry code of the standard
DICTENVFLAG	66-66	1	Char	Standard type flag (dictionary or envelope)
INTENV	67-96	30	Char	Interchange envelope name
STDDESC	97-176	80	Char	Description of the standard
STDPURP	177-688	512	Char	Purpose of the standard
CMTTEXT	689-2736	2048	Char	Comments
ALLTRXS	2737-2737	1	Char	All transactions flag

EDI standard transaction header record (1Y2)

This record defines the transaction header fields being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
STDTRID	34-41	8	Char	Transaction ID
FNGRPID	42-49	8	Char	Function group ID
STDDESC	50-129	80	Char	Description of the standard
STDPURP	130-2177	2048	Char	Purpose of the standard
CMTTEXT	2178-3977	1800	Char	Comments

EDI standard transaction detail record (1Y3)

This record defines the detail fields being used. If you are importing a transaction set, you can define multiple transaction detail records. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
STDTRID	34-41	8	Char	Transaction ID
TABLENO	42-47	6	Char	Table ID
POSNO	48-53	6	Char	Position number
SEGID	54-61	8	Char	Segment ID
SEGREQ	62-62	1	Char	Segment required flag
SEGMAX	63-73	11	Char	Number of times segment may be repeated
UNLIMMAX	74-79	6	Char	No limit on the number of times segment may be repeated
NLOOPID	80-85	6	Char	Loop ID
LPREP	86-96	11	Char	Number of times loop may be repeated
UNLPREP	97-102	6	Char	No limit on the number of times loop may be repeated
LPLEV	103-108	6	Char	Loop level

EDI standard segment header record (1Y4)

This record defines the segment header fields being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
SEGID	34-41	8	Char	Segment ID
STDDDESC	42-121	80	Char	Description of the standard
STDPURP	122-2169	2048	Char	Purpose of the standard
CMTTEXT	2170-3969	1800	Char	Comments

EDI standard segment detail record (1Y5)

This record defines the segment detail fields being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
SEGID	34-41	8	Char	Segment ID
POSNO	42-47	6	Char	Position number
DEID	48-55	8	Char	Data element ID
DEREQ	56-56	1	Char	Data element required flag
SEGMAX	57-67	11	Char	Number of times segment may be repeated

EDI standard data element header record (1Y6)

This record defines the data element header fields being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
DEID	34-41	8	Char	Data element ID
DETYPE	42-43	2	Char	Type of data element
DEMIN	44-54	11	Char	Minimum length of data element
DEMAX	55-65	11	Char	Maximum length of data element
CODELIST	66-73	8	Char	Code list name
STDDDESC	74-123	80	Char	Description of the standard
STDPURP	124-2171	2048	Char	Purpose of the standard
CMTTEXT	2172-3971	1800	Char	Comments

EDI standard data element detail record (1Y7)

This record defines the data element detail fields being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
COMPID	34-41	8	Char	Composite data element ID
POSNO	42-47	6	Char	Position number
DEID	48-55	8	Char	Data element ID
DEREQ	56-56	1	Char	Data element required flag
SEGMAX	57-67	11	Char	Number of times segment may be repeated

EDI standard segment notes record (1Y8)

This record defines the segment note fields being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
SEGID	34-41	8	Char	Segment ID
POSNO	42-47	6	Char	Position number
NOTETYPE	48-48	1	Char	Type of note
NOTEPARA	49-54	6	Char	Paragraph number
RELATION	55-56	2	Char	Relationship of one data element to another
REL1	57-62	6	Char	Related data element reference number
REL2	63-68	6	Char	Related data element reference number
REL3	69-74	6	Char	Related data element reference number
REL4	75-80	6	Char	Related data element reference number
REL5	81-86	6	Char	Related data element reference number
REL6	87-92	6	Char	Related data element reference number
REL7	92-98	6	Char	Related data element reference number
REL8	99-104	6	Char	Related data element reference number
REL9	105-110	6	Char	Related data element reference number
REL10	111-116	6	Char	Related data element reference number
STDNOTE	117-1140	1024	Char	Text of the note

EDI standard transaction notes record (1Y9)

This record defines the transaction note fields being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
STDTRID	34-41	8	Char	Transaction ID
TABLENO	42-47	6	Char	Table number
POSNO	48-53	6	Char	Position number
NOTETYPE	54-54	1	Char	Type of note
NOTEPARA	55-60	6	Char	Paragraph number
RELATION	61-62	2	Char	Relationship of one data element to another
REL1	63-68	6	Char	Related data element reference number
REL2	69-74	6	Char	Related data element reference number
REL3	75-80	6	Char	Related data element reference number
REL4	81-86	6	Char	Related data element reference number
REL5	87-92	6	Char	Related data element reference number
STDNOTE	93-1116	1024	Char	Text of the note

EDI standard composite data element notes record (1YA)

This record defines the composite data element note fields being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
STDID	4-33	30	Char	EDI standard dictionary name
COMPID	34-41	8	Char	Composite data element ID
POSNO	42-47	6	Char	Position number
NOTETYPE	48-48	1	Char	Type of note
NOTEPARA	49-54	6	Char	Paragraph number
RELATION	55-56	2	Char	Relationship of one data element to another
REL1	57-62	6	Char	Related data element reference number
REL2	63-68	6	Char	Related data element reference number
REL3	69-74	6	Char	Related data element reference number
REL4	75-80	6	Char	Related data element reference number
REL5	81-86	6	Char	Related data element reference number
STDNOTE	87-1110	1024	Char	Text of the note

Exporting and importing data formats

Data formats can be exported/imported using either tagged or fixed format files. The data format set includes:

2W1	Data format dictionary
2W2	Data format record ID information
2W3	Data format leader record
2W4	Data format loop record
2W5	Data format record record
2W6	Data format structure record
2W7	Data format field record
2W8	Data format header details record
2W9	Data format loop details record
2WA	Data format record details record
2WB	Data format structure details record

There are no associated objects for data formats.

The following example shows part of an export/import file data set that contains records for exporting/importing one data format.

```

0C1DD5TST1 011025134023ENU010401000000
2W1  DICTIONARYNAME(MMTHL1_DICTIONARY)  DESC(MAUI MAP TEST - HL1 - SUW)
ALLADFS(N)
2W2  RECORDIDINFONAME(MMTHL1_RECORDID)  DESC(MAUI MAP TEST - HL1 - SUW)
ADFTYPE(0)  RIDPOS(1)  RIDLENG(3)  RIDTYPE(AN)
2W3  DICTIONARYNAME(MMTHL1_DICTIONARY)  RECORDIDINFONAME(MMTHL1_RECORDID)
FORMAT(MMTHL1)  DESC(MAUI MAP TEST - HL1 - SUW)  ADFTPIDFLD(TRADINGPART)
ADFBEGINRECID(MMTHL1REC)
2W5  DICTIONARYNAME(MMTHL1_DICTIONARY)  RECORDIDINFONAME(MMTHL1_RECORDID)
ADFRECORDNAME(MMTHL1REC)  DESC(MAUI MAP TEST - HL1 - SUW)  ADFRECORDID(001)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(MMTHL1REC)  DESC(MAUI MAP
TEST - HL1 - SUW)  DATATYPE(AN)  FLDLEN(3)  ADFSSENDMAP(0)  ADFRECEIVEMAP(0)
2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC)
ADF_RMEM_SEQNUM(1)  ADF_RMEM_MEMBERNAME(MMTHL1REC)  ADF_RMEM_MEMBERTYPE(FIELD)
ADF_RMEM_MAXUSE(1)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(ACFIELD)  DESC(MAUI MAP
TEST - HL1 - SUW)  DATATYPE(AN)  FLDLEN(18)  ADFSSENDMAP(0)  ADFRECEIVEMAP(0)
2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC)
ADF_RMEM_SEQNUM(2)  ADF_RMEM_MEMBERNAME(ACFIELD)  ADF_RMEM_MEMBERTYPE(FIELD)
ADF_RMEM_MAXUSE(1)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(ACFIELD)  DESC(MAUI MAP
TEST - HL1 - SUW)  DATATYPE(AC)  FLDLEN(6)  ADFSSENDMAP(0)  ADFRECEIVEMAP(0)
2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC)
ADF_RMEM_SEQNUM(3)  ADF_RMEM_MEMBERNAME(ACFIELD)  ADF_RMEM_MEMBERTYPE(FIELD)
ADF_RMEM_MAXUSE(1)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(FILLER)  DESC(MAUI MAP
TEST - HL1 - SUW)  DATATYPE(AN)  FLDLEN(1)  ADFSSENDMAP(0)  ADFRECEIVEMAP(0)
2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC)
ADF_RMEM_SEQNUM(4)  ADF_RMEM_MEMBERNAME(FILLER)  ADF_RMEM_MEMBERTYPE(FIELD)
ADF_RMEM_MAXUSE(1)

```

```

2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(ELIGBEGDATE)  DESC(MAUI MAP
TEST - HL1 - SUW)  DATATYPE(AN)  FLDLEN(8)  ADFSENDMAP(0)  ADFRECEIVEMAP(0)
2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC)
ADF_RMEM_SEQNUM(5)  ADF_RMEM_MEMBERNAME(ELIGBEGDATE)  ADF_RMEM_MEMBERTYPE(FIELD)
ADF_RMEM_MAXUSE(1)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(ELIGENDDATE)  DESC(MAUI MAP
TEST - HL1 - SUW)  DATATYPE(AN)  FLDLEN(8)  ADFSENDMAP(0)  ADFRECEIVEMAP(0)
2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC)
ADF_RMEM_SEQNUM(6)  ADF_RMEM_MEMBERNAME(ELIGENDDATE)  ADF_RMEM_MEMBERTYPE(FIELD)
ADF_RMEM_MAXUSE(1)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(SUBSCRIBER)  DESC(MAUI MAP
TEST - HL1 - SUW)  DATATYPE(AN)  FLDLEN(18)  ADFSENDMAP(0)  ADFRECEIVEMAP(0)
2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC)
ADF_RMEM_SEQNUM(7)  ADF_RMEM_MEMBERNAME(SUBSCRIBER)  ADF_RMEM_MEMBERTYPE(FIELD)
ADF_RMEM_MAXUSE(1)
2W6  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFSTRUCTNAME(FAMILY)  DESC(MAUI MAP TEST
- HL1 - SUW)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(DEPFIRSTNAME)  DESC(MAUI MAP
TEST - HL1 - SUW)  DATATYPE(AN)  FLDLEN(9)  ADFSENDMAP(0)  ADFRECEIVEMAP(0)
2WB  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_SMEM_STRUCTNAME(FAMILY)
ADF_SMEM_SEQNUM(1)  ADF_SMEM_MEMBERNAME(DEPFIRSTNAME)  ADF_SMEM_MEMBERTYPE(FIELD)
ADF_SMEM_MAXUSE(1)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(DEPLASTNAME)  DESC(MAUI MAP
TEST - HL1 - SUW)  DATATYPE(AN)  FLDLEN(9)  ADFSENDMAP(0)  ADFRECEIVEMAP(0)
2WB  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_SMEM_STRUCTNAME(FAMILY)
ADF_SMEM_SEQNUM(2)  ADF_SMEM_MEMBERNAME(DEPLASTNAME)  ADF_SMEM_MEMBERTYPE(FIELD)
ADF_SMEM_MAXUSE(1)
2W7  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADFFIELDNAME(DEPDOB)  DESC(MAUI MAP TEST -
HL1 - SUW)  DATATYPE(AN)  FLDLEN(8)  ADFSENDMAP(0)  ADFRECEIVEMAP(0)
2WB  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_SMEM_STRUCTNAME(FAMILY)
ADF_SMEM_SEQNUM(3)  ADF_SMEM_MEMBERNAME(DEPDOB)  ADF_SMEM_MEMBERTYPE(FIELD)
ADF_SMEM_MAXUSE(1)
2WA  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_RMEM_RECORDNAME(MMTHL1REC)
ADF_RMEM_SEQNUM(8)  ADF_RMEM_MEMBERNAME(FAMILY)  ADF_RMEM_MEMBERTYPE(STRUCT)
ADF_RMEM_MAXUSE(15)
2W8  DICTIONARYNAME(MMTHL1_DICTIONARY)  ADF_HMEM_ADFFNAME(MMTHL1)
ADF_HMEM_SEQNUM(1)  ADF_HMEM_MEMBERNAME(MMTHL1REC)  ADF_HMEM_MEMBERTYPE(REC)
ADF_HMEM_AREA(Detail)  ADF_HMEM_MAXUSE(1)  ADF_HMEM_INFLOOP(0)
000

```

Data format dictionary record (2W1)

This record defines the data format dictionary being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
DESC	34-83	50	Char	Description
CMTTEXT	84-2131	2048	Char	Comments
ALLADFS	2132-2132	1	Char	Include all ADFs

Data format record ID record (2W2)

This record defines the data format record being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
RECORDIDINFONAME	4-33	30	Char	Record ID information name
DESC	34-83	50	Char	Description
ADFTYPE	84-84	1	Char	Data format type
RIDPOS	85-90	6	Char	Record ID position
RIDLENG	91-101	11	Char	Record ID length
RIDTYPE	102-103	2	Char	Record ID type
CMTTEXT	104-2151	2048	Char	Comments

Data format header record (2W3)

This record defines the data format header being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
RECORDIDINFONAME	34-63	30	Char	Record ID information name
FORMAT	64-79	16	Char	Data format name
DESC	80-129	50	Char	Description
FILEID	130-137	8	Char	ddname
FILETYPE	138-139	2	Char	File type
ADFTPIDFLD	140-169	30	Char	Field containing trading partner ID
ADFBEGINRECID	170-199	30	Char	Field containing beginning record ID
ADFENDRECID	200-229	30	Char	Field containing ending record ID
ADFGENRTCDE	230-259	30	Char	Field containing generic routing code
CMTTEXT	260-2307	2048	Char	Comments
INTSNDQUALFLD	2308-2337	30	Char	Field containing interchange sender ID qualifier
INTSNDIDFLD	2338-2367	30	Char	Field containing interchange sender ID
INTRCVQUALFLD	2368-2397	30	Char	Field containing interchange receiver ID qualifier

Tag	Position	Length	Type	Description
INTRCVFLD	2398-2427	30	Char	Field containing inter-change receiver ID
APPLTPIDFLD	2428-2457	30	Char	Field containing application trading partner nickname
EDITPIDFLD	2458-2487	30	Char	Field containing EDI trading partner nickname
ADFCMDELIM	2488-2488	1	Char	Comma-separated value delimiter
ADFRECDELIM	2489-2490	2	Char	Record delimiter
ADFUNICODE	2491-2506	16	Char	Code page
ADFTXTDELIM	2507-2508	2	Char	Reserved for future use
ADFTXTQUAL	2509-2538	30	Char	Reserved for future use

Data format loop record (2W4)

This record defines the data format loop being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
RECORDIDINFONAME	34-63	30	Char	Record ID information name
ADFLOOPNAME	64-93	30	Char	Loop name
DESC	94-143	50	Char	Description
CMTTEXT	144-2191	2048	Char	Comments

Data format record record (2W5)

This record defines the data format record being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
RECORDIDINFONAME	34-63	30	Char	Record ID information name
ADFRECORDNAME	64-93	30	Char	Record name
DESC	94-143	50	Char	Description
ADFRECORDID	144-159	16	Char	Record ID
CMTTEXT	160-2207	2048	Char	Comments

Data format structure record (2W6)

This record defines the data format structure being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADSTRUCTNAME	34-63	30	Char	Structure name
DESC	64-113	50	Char	Description
CMTTEXT	114-2161	2048	Char	Comments

Data format field record (2W7)

This record defines the data format fields being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADFFIELDNAME	34-63	30	Char	Field name
DESC	64-113	50	Char	Description
DATATYPE	114-115	2	Char	Data type
FLDLEN	116-121	6	Char	Field length
ADFMAPCOMMANDS	122-201	80	Char	Mapping command line
ADFSENDMAP	202-202	1	Char	Send map
ADFRECEIVEMAP	203-203	1	Char	Receive map
ADFVALTBLNAME	204-211	8	Char	Code list name
CMTTEXT	212-2259	2048	Char	Comments

Data format header details record (2W8)

This record defines the data format header details being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADF_HMEM_ADFNAME	34-49	16	Char	Data format name
ADF_HMEM_SEQNUM	50-59	10	Char	Sequence number
ADF_HMEM_MEMBERNAME	60-89	30	Char	Member name
ADF_HMEM_MEMBERTYPE	90-97	8	Char	Member type

Tag	Position	Length	Type	Description
ADF_HMEM_AREA	98-113	16	Char	Area value
ADF_HMEM_MAXUSE	114-118	5	Char	Maximum number of occurrences
ADF_HMEM_INFLOOP	119-119	1	Char	Infinite loop flag

Data format loop details record (2W9)

This record defines the data format loop details being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADF_LMEM_LOOPNAME	34-63	30	Char	Loop name
ADF_LMEM_SEQNUM	64-73	10	Char	Sequence number
ADF_LMEM_MEMBERNAME	74-103	30	Char	Member name
ADF_LMEM_MEMBERTYPE	104-111	8	Char	Member type
ADF_LMEM_MAXUSE	112-116	5	Char	Maximum number of occurrences
ADF_LMEM_INFLOOP	117-117	1	Char	Infinite loop flag

Data format record details record (2WA)

This record defines the data format record details being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADF_RMEM_RECORDNAME	34-63	30	Char	Record name
ADF_RMEM_SEQNUM	64-73	10	Char	Sequence number
ADF_RMEM_MEMBERNAME	74-103	30	Char	Member name
ADF_RMEM_MEMBERTYPE	104-111	8	Char	Member type
ADF_RMEM_MAXUSE	112-116	5	Char	Maximum number of occurrences
ADF_RMEM_OCCURSDEPNAME	117-146	30	Char	The field that determines whether this structure should occur

Data format structure details record (2WB)

This record defines the data format structure details being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
ADF_SMEM_STRUCTNAME	34-63	30	Char	Structure name
ADF_SMEM_SEQNUM	64-73	10	Char	Sequence number
ADF_SMEM_MEMBERNAME	74-103	30	Char	Member name
ADF_SMEM_MEMBERTYPE	104-111	8	Char	Member type
ADF_SMEM_MAXUSE	112-116	5	Char	Maximum number of occurrences
ADF_SMEM_OCCURSDEPNAME	117-146	30	Char	The field that determines whether this structure should occur

Exporting and importing maps

Maps can be exported/imported using tagged or fixed format files. The map set includes:

- 3T4** Trading partner
- 3T5** Trading partner send usages/rules
- 3T6** Trading partner receive usages/rules
- 3T7** Data transformation map usage/rule
- 3V1** Map header record
- 3V2** Map segment record
- 3V3** Map element record
- 3V4** Map application control record
- 3VA** Map syntax record
- 3VB** Map local variables record
- 3VC** Map reference record
- 3VD** Map nodes record
- 3VE** Map comments record
- BK2** Global variables

The sixteen associated objects for exporting maps are listed below. Either maps or usages/rules must be selected:

1. Transaction usages/rules
2. Control string
3. EDI standard transaction
4. Data format
5. Validation tables
6. Translation tables
7. User exit routines
8. Trading partner profile
9. Translation exit routines
10. Network security profile
11. Network profile
12. Network commands profile
13. Envelope profile
14. Envelope standard
15. Maps
16. Conversion of prior-release objects



NOTE: In trading partner-to-trading partner translations, DataInterchange stores control numbers by ISA Sender Qualifier, ISA Receiver Qualifier, and ISA Receiver, ensuring that the receiver gets a contiguous set of control numbers from every sender.

The following example shows an export/import file that contains tags for a send map and the associated send usage/rule.

```

0C1DD5TST1 011025135724ENU010401000000
3V1 TPTID(MMTHL1) STDID(X12V3R7) STDTRID(271) FORMAT(MMTHL1) DIR(S)
GENREQD(N) DESC(MAUI MAP TEST - HL1 - SUW Outbound)
DICTIONARYNAME(MMTHL1_DICTIONARY) CMPCHK(N) BASE(T)
3V2 TPTID(MMTHL1) TABLE(1) POS(20) OCCUR(1) GENFLAG(N)
3V2 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(1) DEPTHSEQ(1) PATH(MMTHL1REC)
HLNODE(1) HLPARNODE(1) HL03(20) GENFLAG(N)
3V2 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(2) DEPTHSEQ(3) HLNODE(2)
HLPARNODE(1) HL03(21) PARHL03(20) GENFLAG(N)
3V2 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(3) DEPTHSEQ(5) HLNODE(3)
HLPARNODE(1) HL03(22) PARHL03(21) GENFLAG(N)
3V2 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(4) DEPTHSEQ(6)
PATH(MMTHL1REC\FAMILY) HLNODE(4) HLPARNODE(1) HL03(23) PARHL03(22) GENFLAG(N)
3V2 TPTID(MMTHL1) TABLE(2) POS(20) OCCUR(1) PARTABLE(2) PARPOS(10)
PAROCCUR(2) DEPTHSEQ(4) QUALOCCUR(1) GENFLAG(N)
3V2 TPTID(MMTHL1) TABLE(2) POS(30) OCCUR(1) PARTABLE(2) PARPOS(10)
PAROCCUR(4) DEPTHSEQ(7) QUALOCCUR(1) GENFLAG(N)
3V2 TPTID(MMTHL1) TABLE(2) POS(30) OCCUR(2) PARTABLE(2) PARPOS(10)
PAROCCUR(1) DEPTHSEQ(2) QUALOCCUR(1) GENFLAG(N)
3V3 TPTID(MMTHL1) TABLE(1) POS(20) OCCUR(1) ELEPOS(1) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(0022)
3V3 TPTID(MMTHL1) TABLE(1) POS(20) OCCUR(1) ELEPOS(2) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(11)
3V3 TPTID(MMTHL1) TABLE(1) POS(20) OCCUR(1) ELEPOS(3) ELEOCCUR(1)
PATH(MMTHL1REC) ADFFIELDNAME(ACFIELD) DATEEDIT(-4096)
3V3 TPTID(MMTHL1) TABLE(1) POS(20) OCCUR(1) ELEPOS(4) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&DATE)
3V3 TPTID(MMTHL1) TABLE(1) POS(20) OCCUR(1) ELEPOS(5) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&TIME)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(1) ELEPOS(1) ELEOCCUR(2)
DATEEDIT(-4096) LITVAL(&SET MAPIT 0)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(1) ELEPOS(1) ELEOCCUR(3)
PATH(MMTHL1REC) ADFFIELDNAME(SUBSCRIBER) DATEEDIT(-4096) LITVAL(&IFDATA &SET
MAPIT 1)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(1) ELEPOS(1) ELEOCCUR(4)
DATEEDIT(-4096) LITVAL(&IF(MAPIT = 1?) &HLID)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(1) ELEPOS(2) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&IF(MAPIT = 1?) &HLPID)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(1) ELEPOS(3) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&IF(MAPIT = 1?) &HCODE)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(1) ELEPOS(4) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&IF(MAPIT = 1?) &HCHILD)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(2) ELEPOS(1) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&HLID)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(2) ELEPOS(2) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&HLPID)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(2) ELEPOS(3) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&HCODE)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(2) ELEPOS(4) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&HCHILD)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(3) ELEPOS(1) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&HLID)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(3) ELEPOS(2) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&HLPID)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(3) ELEPOS(3) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&HCODE)
3V3 TPTID(MMTHL1) TABLE(2) POS(10) OCCUR(3) ELEPOS(4) ELEOCCUR(1)
DATEEDIT(-4096) LITVAL(&HCHILD)

```

```

3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(1)  ELEOCCUR(2)
DATEEDIT(-4096)  LITVAL(&SET  MAPIT 0)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(1)  ELEOCCUR(3)
PATH(MMTHL1REC\FAMILY)  ADFFIELDNAME(DEPDOB)  DATEEDIT(-4096)  LITVAL(&IFDATA  &SET
MAPIT 1)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(1)  ELEOCCUR(4)
DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HLID)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(2)  ELEOCCUR(1)
DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HLPID)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(3)  ELEOCCUR(1)
DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HCODE)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(10)  OCCUR(4)  ELEPOS(4)  ELEOCCUR(1)
DATEEDIT(-4096)  LITVAL(&IF(MAPIT = 1?) &HCHILD)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(20)  OCCUR(1)  ELEPOS(1)  ELEOCCUR(1)
DATEEDIT(-4096)  LITVAL(AC)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(20)  OCCUR(1)  ELEPOS(2)  ELEOCCUR(1)
DATEEDIT(-4096)  LITVAL(&ACFIELD)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(1)  ELEPOS(1)  ELEOCCUR(1)
DATEEDIT(-4096)  LITVAL(DD)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(1)  ELEPOS(2)  ELEOCCUR(1)
DATEEDIT(-4096)  LITVAL(D)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(1)  ELEPOS(3)  ELEOCCUR(1)
PATH(MMTHL1REC\FAMILY)  ADFFIELDNAME(DEPLASTNAME)  DATEEDIT(-4096)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(1)  ELEPOS(4)  ELEOCCUR(1)
PATH(MMTHL1REC\FAMILY)  ADFFIELDNAME(DEPFIRSTNAME)  DATEEDIT(-4096)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(2)  ELEPOS(1)  ELEOCCUR(1)
DATEEDIT(-4096)  LITVAL(SS)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(2)  ELEPOS(2)  ELEOCCUR(1)
DATEEDIT(-4096)  LITVAL(S)
3V3  TPTID(MMTHL1)  TABLE(2)  POS(30)  OCCUR(2)  ELEPOS(3)  ELEOCCUR(1)
PATH(MMTHL1REC)  ADFFIELDNAME(SUBSCRIBER)  DATEEDIT(-4096)
000

```

The following tables define the fields for exporting/importing maps.

Map header record (3V1)

This record defines the map header being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
STDID	20-49	30	Char	EDI standard dictionary name
STDTRID	50-57	8	Char	EDI standard transaction ID
FORMAT	58-73	16	Char	Data format ID
DIR	74-74	1	Char	Map type (send/receive/data transformation) flag
GENREQD	75-75	1	Char	Compile required
DESC	76-125	50	Char	Description
DICTIONARYNAME	126-155	30	Char	Data format dictionary name
CMPCHK	156-156	1	Char	Compliance check flag
BASE	157-157	1	Char	Future use

Map segment record (3V2)

This record defines the map segment being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
TABLE	20-20	1	Char	Table
POS	21-25	5	Char	Position in table
OCCUR	26-30	5	Char	Occurrence
PARTABLE	31-31	1	Char	Parent table
PARPOS	32-36	5	Char	Parent position
PAROCCUR	37-41	5	Char	Parent occurrence
DEPTHSEQ	42-46	5	Char	Depth sequence number
QUALOCCUR	47-52	6	Char	Qualification occurrence number
QUALELEPOS	53-56	4	Char	Qualified element position
QUALELEREPEATNO	57-61	5	Char	Qualified element repeat number
QUALSUBELEPOS	62-65	4	Char	Qualified sub-element position
PATH	66-320	255	Char	Qualified path
QUALTEXT	321-365	45	Char	Element loop value
HLNODE	366-370	5	Char	Hierarchical loop node number
HLPARNODE	371-375	5	Char	Hierarchical loop parent node number
HL03	376-377	2	Char	HL03 value
PARHL03	378-379	2	Char	HL03 parent value
FLDHL03	380-409	30	Char	HL03 field value
PARFLDHL03	410-439	30	Char	HL03 field parent value
GENFLAG	440-440	1	Char	Generate segment
DOSNIPET	441-441	1	Char	Snipet text
BSNIPET	442-521	80	Char	Before snipet flag
ASNIPET	522-601	80	Char	After snipet flag
CMTTEXT	602-2649	2048	Char	Segment mapping comments

Map element record (3V3)

This record defines the map element being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
TABLE	20-20	1	Char	Table ID
POS	21-25	5	Char	Position in table
OCCUR	26-30	5	Char	Table occurrence number
ELEPOS	31-35	5	Char	Element position number
SUBELEPOS	36-40	5	Char	Sub-element position number
ELEOCCUR	41-45	5	Char	Element occurrence number
PATH	46-300	255	Char	Path name
ADFFIELDNAME	301-330	30	Char	Data format field name
ELEQUALPOS	331-335	5	Char	Qualifying element's position number
SUBELEQUALPOS	336-340	5	Char	Qualifying sub-element's position number
ELEQUALOCCUR	341-345	5	Char	Qualifying element's occurrence number
QUALTEXT	346-386	45	Char	Element qualifier
TRNTAB	391-398	8	Char	Translation table
USEREXIT	699-406	8	Char	User exit ID
VALTAB	407-414	8	Char	Code list ID
DATEEDIT	415-416	2	Char	Date edit field
SUBPOS	417-422	6	Char	Substring or concatenation position
SUBLEN	423-428	6	Char	Substring or concatenation length
ACCUM1	429-430	2	Char	Accumulator1
ACTION1	431-432	2	Char	Accumulator1 action
ACCUM2	433-434	2	Char	Accumulator2
ACTION2	435-436	2	Char	Accumulator2 action
ACCUM3	437-438	2	Char	Accumulator3
ACTION3	439-440	2	Char	Accumulator3 action
ACCUM4	441-442	2	Char	Accumulator4
ACTION4	443-444	2	Char	Accumulator4 action
LITVAL	445-524	80	Char	Code Snippet

Tag	Position	Length	Type	Description
CMTTEXT	525-2572	2048	Char	Element comments
ELEREPEATNO	2573-2577	5	Char	Element repeat occurrence number
QUALPATH	2578-2832	255	Char	Qualifying path
UNIQUEID	2833-2838	6	Char	Mapping unique ID

Map application control record (3V9)

This record defines the map application control being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-20	1	Char	Unique ID
SEQUENCE	21-21	1	Char	Sequence
PATH	22-276	255	Char	Path name
ADFFIELDNAME	277-306	30	Char	Data format field name
LENGTH	307-311	5	Char	Field length

Map syntax record (3VA)

This record defines the map syntax being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-24	5	Char	Unique ID
BASE	25-25	1	Char	Future use
SEQNO	26-30	6	Char	Sequence number
SYNTAX	31-31	1	Char	Syntax type
DICTIONARYNAME	32-61	30	Char	Dictionary name
DOCNAME	62-91	30	Char	Document definition name

Map local variables record (3VB)

This record defines the map local variables being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
VARNAME	20-49	30	Char	Variable name
DISPNAME	50-79	30	Char	Displayed variable name
DESC	80-129	50	Char	Description
DATATYPE	130-130	1	Char	Data type
SCOPE	131-131	1	Char	Scope
DEMAX	132-136	5	Char	Maximum length
INITVAL	137-168	32	Char	Initial value

Map reference record (3VC)

This record defines the map reference being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-25	6	Char	Unique ID
TYPE	26-26	1	Char	Type
PARENTID	27-37	11	Char	Syntax ID
SYNTAX	38-38	1	Char	Syntax type
DICTIONARYNAME	39-68	30	Char	Dictionary name
FORMAT	69-98	30	Char	Document definition name
PATH	99-353	255	Char	Path

Map nodes record (3VD)

This record defines the map nodes being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-25	6	Char	Unique ID
PARENTID	26-36	11	Char	Parent's unique ID
SEQNO	37-47	11	Char	Sequence
TYPE	48-48	1	Char	Map type
SUBTYPE	49-49	1	Char	Map sub type

Tag	Position	Length	Type	Description
ASSOCFWD	50-55	6	Char	Associated forward
ASSOCBKD	56-61	6	Char	Associated backward
POSNO	62-70	9	Char	Position number
EDIT	71-71	1	Char	Map edit
REFID	72-77	6	Char	Map reference ID
REPEATS	78-78	1	Char	Node repeats
QUALIFIED	79-79	1	Char	Qualified flag
CMTTEXT	80-2128	2048	Char	Comment

Map commands record (3VE)

This record defines the map commands being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPTID	4-19	16	Char	Map name
UNIQUEID	20-25	6	Char	Unique ID
PARENTID	26-36	11	Char	Node's unique ID
SEQNO	37-47	11	Char	Sequence
VERSION	48-53	6	Char	Version
COMMAND	54-2101	2048	Char	Map command

Global Variables Details Record (BK2)

This record defines the global variables being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
VARNAME	4-33	30	Char	Variable name
DISPNAME	34-63	30	Char	Displayed variable name
DESC	64-113	50	Char	Description
DATATYPE	114-114	1	Char	Data type
SCOPE	115-115	1	Char	Scope
DEMAX	116-120	5	Char	Maximum length
INITVAL	121-152	32	Char	Initial value

Send usage record (3T5)

This record defines the send usage/rule being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
INTPID	4-38	35	Char	Internal trading partner ID
FORMAT	39-54	16	Char	Data format ID
TPTID	55-70	16	Char	Trading partner mapping transaction ID
TPNICKN	71-86	16	Char	Trading partner nickname
ENVTYPE	87-87	1	Char	EDI standard envelope type
STDPROF	88-95	8	Char	EDI standard profile name
APPSNDID	96-130	35	Char	Application sender ID
APPRECID	131-165	35	Char	Application receiver ID
APPASSWD	166-179	14	Char	Application password
PSTTRXIT	180-187	8	Char	Post-translation exit routine name
ACTIVE	188-188	1	Char	Active usage for this INTPID/FORMAT combination
ACKREQ	189-189	1	Char	Acknowledgment expected flag
TESTIND	190-190	1	Char	Test indicator
LOGAPP	191-191	1	Char	Log application data
ENFORCEHIER	192-192	1	Char	Enforce structure hierarchy
STRUCTDATACHK	193-193	1	Char	Structure must produce data
TRVALVL	194-194	1	Char	Validation level
TRERLVL	195-195	1	Char	Acceptable error level
GRPSECPR	196-203	8	Char	Group network security profile name
GRPENCKY	204-219	16	Char	Group encryption key name
GRPAUTKY	220-235	16	Char	Group authentication key name
TRNSECPR	236-243	8	Char	Trans network security profile name
TRNENCKY	244-259	16	Char	Trans encryption key name
TRNAUTKY	260-275	16	Char	Trans authentication key name
APPLTPID	276-291	16	Char	Application trading partner nickname

Tag	Position	Length	Type	Description
ALPHANUM	292-299	8	Char	Alternate table for ALPHANUM
CHARSET	300-307	8	Char	Alternate table for CHARSET
CNTRX	308-308	1	Char	Control numbers utilization flag

Receive usage record (3T6)

This record defines the receive usage/rule being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
TPNICKN	4-19	16	Char	Trading partner nickname
STDTRID	20-27	8	Char	EDI standard transaction ID
TPTID	28-43	16	Char	Map name
APSNDRCV	44-78	35	Char	Application sender/receiver ID
AGENCY	79-86	8	Char	Responsible agency code
VER	87-94	8	Char	Version
REL	95-102	8	Char	Release
SNDRCVFL	103-103	1	Char	Sender or receiver flag
INTPID	104-138	35	Char	Internal trading partner ID
STDPROF	139-146	8	Char	Standard profile name
APPASSWD	147-160	14	Char	Application password
PRETRXIT	161-168	8	Char	Pre-translation exit routine
FILEID	169-176	8	Char	Logical receiving data set name
FILETYPE	177-178	2	Char	Type of receive file
ACTIVE	179-179	1	Char	Active usage flag for this trading partner
ACKTRN	180-187	8	Char	Acknowledgment transaction ID
TESTIND	188-188	1	Char	Test indicator
LOGAPP	189-189	1	Char	Log application data
OVERLAYCHK	190-190	1	Char	Consider data overlay an error
UNEXPFLDCHK	191-191	1	Char	Consider unexpected field an error
UNEXPSEGCHK	192-192	1	Char	Consider unexpected segment an error
SWITCHROUTING	193-193	1	Char	Switch application routing IDs on functional acknowledgment
TRVALVL	194-194	1	Char	Validation level

Tag	Position	Length	Type	Description
TRERLVL	195-195	1	Char	Acceptable error level
INTFA	196-196	1	Char	Internal functional acknowledgment switch
APPLTPID	197-212	16	Char	Application trading partner nickname
ALPHANUM	213-220	8	Char	Alternate table for ALPHANUM
CHARSET	221-228	8	Char	Alternate table for CHARSET
CNTRX	229-229	1	Char	Control numbers utilization flag

Data transformation map rule record (3T7)

This record defines the data transformation map being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	Dictionary name
DOCID	34-63	30	Char	Document definition name
MAPID	64-79	16	Char	Map name
RECVTPID	80-95	16	Char	Receiving trading partner nickname
SENDTPID	96-111	16	Char	Sending trading partner nickname
TESTIND	112-112	1	Char	Test indicator
PROCESS	113-152	40	Char	Process ID
DESCRIPT	153-182	30	Char	Description
ENVTYPE	183-183	1	Char	EDI standard envelope type
STDPROF	184-191	8	Char	EDI standard profile name
APPSNDID	192-226	35	Char	Application sender ID
APPRECID	227-261	35	Char	Application receiver ID
APPASSWD	262-275	14	Char	Application password
SYNTAX	276-278	3	Char	Syntax type
PRETRANEXIT	279-286	8	Char	Pre-translation exit routine name
POSTTRANEXIT	287-294	8	Char	Post-translation exit routine name
ACTIVE	295-295	1	Char	Active usage for this INTPID/FORMAT combination
ACKREQ	296-296	1	Char	Acknowledgment expected flag
LOGINBIMG	297-297	1	Char	Log inbound application data
LOGOUTBIMG	298-298	1	Char	Log outbound application data

Tag	Position	Length	Type	Description
INBVALLVL	299-299	1	Char	Inbound validation level
OUTBVALLVL	300-300	1	Char	Outbound validation level
DATOVRCCHK	301-201	1	Char	Consider data overlay an error
UNEXPDECHK	302-302	1	Char	Unexpected element check
UNEXPSEGCHK	303-303	1	Char	Unexpected segment check
FAENVTYPE	304-304	1	Char	Functional acknowledgment envelope type
FAENVPROF	305-312	8	Char	Functional acknowledgment envelope profile name
GRPLVLFA	313-313	1	Char	Group level functional acknowledgment
ENFSTRHIER	314-314	1	Char	Enforce structure hierarchy
STRMUSTPRD	315-315	1	Char	Structure must produce data
TRERLVL	316-316	1	Char	Acceptable error level
INBGRPSECPR	317-324		Char	Inbound group network security profile name
INBGRPENCKY	325-340	16	Char	Inbound group encryption key name
INBGRPAUTKY	341-356	16	Char	Inbound group authentication key name
INBTRNSECPR	357-364	8	Char	Inbound trans network security profile name
INBTRNENCKY	365-380	16	Char	Inbound trans encryption key name
INBTRNAUTKY	381-396	16	Char	Inbound trans authentication key name
INBALPHANUM	397-404	8	Char	Inbound alternate table for ALPHANUM
INBCHARSET	405-412	8	Char	Inbound alternate table for CHARSET
OUTBGRPSECPR	413-420	8	Char	Outbound group network security profile name
OUTBGRPENCKY	421-436	16	Char	Outbound group encryption key name
OUTBGRPAUTKY	437-452	16	Char	Outbound group authentication key name
OUTBTRNSECPR	453-460	8	Char	Outbound trans network security profile name

Tag	Position	Length	Type	Description
OUTBTRNENCKY	461-476	16	Char	Outbound trans encryption key name
OUTBTRNAUTKY	477-492	16	Char	Outbound trans authentication key name
OUTBALPHANUM	493-500	8	Char	Outbound alternate table for ALPHANUM
OUTBCCHARSET	501-508	8	Char	Outbound alternate table for CHARSET
CNTRX	509-509	1	Char	Control numbers utilization flag
FILEID	510-517	8	Char	Logical application data set name
FILETYPE	518-519	2	Char	Type of application file
ACKDOCID	520-549	30	Char	Acknowledgment document ID

Exporting and importing table definitions

Table definitions can be exported/imported using either tagged or fixed format files. The table definitions set includes:

B1 Table definition

B2 Table entry

The following example shows an export/import file which contains tags for table definitions.

```
0C1SMITH 940419161616ENU010103000000
8B1TBLID(AAAAA) DATECRT(940325) TBLDSC(dd) PUBAUTH(ALL) FLAGS(00)
8B2TBLID(AAAAA) VAR1(A)
000
```

The following tables define the fields for exporting/importing tables.

Export/Import table definition (B1)

The following table defines the fields for exporting and importing table definitions.

Tag	Position	Length	Type	Description
TBLID	4-11	8	Char	Table ID
DATECRT	12-17	6	Char	Date created
TBLDSC	18-52	35	Char	Table description
PUBAUTH	53-58	6	Char	Authority
OWNER	59-66	8	Char	Owner
FLAGS	67-68	2	Char	Flags (always 00)
KEYID	69-86	18	Char	Key field ID
KEYLN	87-92	6	Char	Key field length
RECLN	93-98	6	Char	Record length
NUMFLDS	99-104	6	Char	Number of fields in a table row
TBLTYPE	105-105	1	Char	Table type
FLD1ID	106-123	18	Char	Field 1 ID
FLD1OFF	124-129	6	Char	Field 1 offset
FLD1LEN	130-132	3	Char	Field 1 length
FLD1TYPE	133-134	2	Char	Field 1 data type
FLD1DSC	135-169	35	Char	Field 1 description
FLD2ID	170-187	18	Char	Field 2 ID
FLD2OFF	188-193	6	Char	Field 2 offset
FLD2LEN	194-196	3	Char	Field 2 length

Tag	Position	Length	Type	Description
FLD2TYPE	197-198	2	Char	Field 2 data type
FLD2DSC	199-233	35	Char	Field 2 description
TAG	234-241	8	Char	Future index tag

Export/Import table definition (B1) field descriptions

Table ID (TBLID)

A name for referring to the table.

Date created (DATECRT)

The date this table was created.

Table description (TBLDSC)

A brief description of this table.

Authority (PUBAUTH)

For DataInterchange use only.

Owner (OWNER)

The user ID of the owner of this table.

Flags (FLAGS)

Always set to 00.

Key ID (KEYID)

The ID of the field used as the key for performing lookups. Validation and type **T** translation tables use the first (or only) field as the key field. Type **R** translation tables use the second field.

Key field length (KEYLN)

The length of the key field.

Record length (RECLN)

The length of a row in this table.

Number of fields (NUMFLDS)

The total number of fields in a row in this table. Validation tables have one field. Translation tables have two.

Table type (TBLTYPE)

The type of table. Valid values are:

- T** Translation table. Contains pairs of values: the value to be translated and the value to be substituted for it. For example, you can translate a purchaser's part number to a supplier's part number before sending a purchase order. The table is organized with the application value as the key value and the EDI standard or trading partner value as data. With this type of table, multiple application values can translate to the same EDI standard value, but each EDI standard value will translate to one and only one application value. This table gives the best performance when translating from an application to an EDI standard.
- R** Translation table. Serves the same purpose as the type **T** table but is organized with the EDI standard value as the key value and the application value as data. With this type of table, multiple EDI standard values can translate to the same application value, but each application value will translate to one and only one EDI standard value. This table gives the best performance when translating from an EDI standard to an application.
- V** Validation table. This table contains a list of acceptable values for a field. It can contain several values or only one value.

Field 1 ID (FLD1ID)

The ID of the first column in a row of this table.

Field 1 offset (FLD1OFF)

The offset into a row of this table where the first field begins.

Field 1 length (FLD1LEN)

The maximum number of characters one entry in this column will contain. The maximum is **35**. For translation tables (data type **T**) the maximum length for Field 1 and Field 2 combined is 68 characters. Include decimal points and integer signs as part of the data length. For example, the value -12.34 requires a length of 6.

Field 1 data type (FLD1TYPE)

The data type for entries in this column. Valid values are:

- CH** Character. Any combination of characters.
- R** Real. A numeric field that requires a decimal point for fractional values. The decimal point is optional for integers. A sign (+ or -) is optional.

Field 1 description (FLD1DSC)

A brief description of this field.

Field 2 ID (FLD2ID)

The ID of the second column in a row of this table (if there is one).

Field 2 offset (FLD2OFF)

The offset into a row of this table where the second field begins.

Field 2 length (FLD2LEN)

The maximum number of characters one entry in this column will contain. The maximum is **35**. For translation tables (data type T) the maximum length for Field 1 and Field 2 combined is 68 characters. Include decimal points and integer signs as part of the data length. For example, the value -12.34 requires a length of 6.

Field 2 data type (FLD2TYPE)

The data type for entries in this column. Valid values are:

- CH** Character. Any combination of characters.
- R** Real. A numeric field that requires a decimal point for fractional values. The decimal point is optional for integers. A sign (+ or -) is optional.

Field 2 description (FLD2DSC)

A brief description of this field.

Tag (TAG)

Reserved for future use.

Export/Import table entry (B2)

The following table defines the tags and lengths for exporting/importing table entries.

Tag	Position	Length	Type	Description
TBLID	4-11	8	Char	Table ID
VAR1	12-Variable	Variable	Char	Translate FROM field value
VAR2	Variable	Variable	Char	Translate TO field value

Table entry (B2) field descriptions

Table ID (TBLID)

A name for referring to the table.

VAR1

For validation tables, the valid value to be entered into the table. For translation tables, the application value (user-defined) to be associated with a value from the EDI standard.

VAR2

Applies only for translation tables. The EDI standard value to be associated with an application value (user-defined).

Exporting and importing XML records

XML data definitions can be exported/imported using either tagged or fixed format files. The XML definitions set includes:

- AJ1** XML dictionary record
- AJ2** XML DTD header record
- AJ3** XML DTD detail record

The following example shows an export/import file which contains tags for XML definitions.

```

OC1DD5TST1 011025135847ENU010401000000
AJ1  DICTIONARYNAME(TESTS)  DESC(Test DTDs)  ALLDTDS(Y)
AJ2  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  DESC(Order with Sender/Receiver)
DTDROOTELEM(OrderSR)  DTDSENDQUALELEM(\OrderSR\Header\Sender\Qualifier\\)
DTDSENDIDELEM(\OrderSR\Header\Sender\Id\\)
DTDRECVQUALELEM(\OrderSR\Header\Receiver\Qualifier\\)
DTDRECVIDELEM(\OrderSR\Header\Receiver\Id\\)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(1)  DTDDATA(<?xml version="1.0"
encoding="ISO-8859-1"?>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(2)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(3)  DTDDATA(<!-- @version:  -->)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(4)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(5)  DTDDATA(<!ELEMENT OrderSR
(Header, DetailLoop*, Trailer?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(6)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(7)  DTDDATA(<!ELEMENT Header
(PONum, PODate, Sender, Receiver?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(8)  DTDDATA(<!ATTLIST Header
typecode CDATA #REQUIRED>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(9)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(10)  DTDDATA(<!ELEMENT PONum
(#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(11)  DTDDATA(<!ELEMENT PODate
(#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(12)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(13)  DTDDATA(<!ELEMENT Sender
(Id, Qualifier?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(14)  DTDDATA(<!ELEMENT Receiver
(Id, Qualifier?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(15)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(16)  DTDDATA(<!ELEMENT Id
(#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(17)  DTDDATA(<!ELEMENT Qualifier
(#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(18)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(19)  DTDDATA(<!ELEMENT
DetailLoop (ItemNumber, SubDetail+?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(20)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(21)  DTDDATA(<!ELEMENT
ItemNumber (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(22)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(23)  DTDDATA(<!ELEMENT SubDetail
(Description, Quantity, UnitPrice?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(24)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(25)  DTDDATA(<!ELEMENT
Description (#PCDATA?)>)
AJ3  DICTIONARYNAME(TESTS)  DOCNAME(POXML5SR)  SEQNO(26)  DTDDATA(<!ELEMENT Quantity
(#PCDATA?)>)

```

```

AJ3  DICTIONARYNAME( TESTS)  DOCNAME(POXML5SR)  SEQNO(27)  DTDDATA(<!ELEMENT UnitPrice
(#PCDATA?)>)
AJ3  DICTIONARYNAME( TESTS)  DOCNAME(POXML5SR)  SEQNO(28)
AJ3  DICTIONARYNAME( TESTS)  DOCNAME(POXML5SR)  SEQNO(29)  DTDDATA(<!ELEMENT Trailer
(ItemCount, TotalBucks?)>)
AJ3  DICTIONARYNAME( TESTS)  DOCNAME(POXML5SR)  SEQNO(30)
AJ3  DICTIONARYNAME( TESTS)  DOCNAME(POXML5SR)  SEQNO(31)  DTDDATA(<!ELEMENT ItemCount
(#PCDATA?)>)
AJ3  DICTIONARYNAME( TESTS)  DOCNAME(POXML5SR)  SEQNO(32)  DTDDATA(<!ELEMENT
TotalBucks (#PCDATA?)>)
AJ3  DICTIONARYNAME( TESTS)  DOCNAME(POXML5SR)  SEQNO(33)
000

```

The following tables define the fields for exporting/importing XML definitions.

XML dictionary record (AJ1)

This record defines the XML dictionary being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	XML dictionary name
DESC	34-83	50	Char	Description
ALLDTDS	84-84	1	Char	All XML DTDs flag

XML DTD header record (AJ2)

This record defines the XML DTD header being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	XML dictionary name
DOCNAME	34-63	30	Char	XML DTD Name
DESC	64-113	50	Char	Description
DTDROOTELEM	114-177	64	Char	Root element
DTSENDQUALELEM	178-432	255	Char	Element containing sender qualifier
DTSENDIDELEM	433-687	255	Char	Element containing sender ID
DTSENDALIASTBL	688-695	8	Char	Sender alias table
DTDRECVQUALELEM	696-950	255	Char	Element containing receiver qualifier
DTDRECVIDELEM	951-1205	255	Char	Element containing receiver ID
DTDRECVALIASTBL	1206-1213	8	Char	Receiver alias table
DTDLOC	1214-1363	150	Char	Source location of DTD

XML DTD details record (AJ3)

This record defines the XML DTD details being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
DICTIONARYNAME	4-33	30	Char	XML dictionary name
DOCNAME	34-63	30	Char	XML DTD Name
SEQNO	64-74	11	Char	Data sequence number
DTDCONT	75-75	1	Char	Continuation flag
DTDDATA	76-330	255	Char	DTD data

Additional profile layouts

The sixteen additional profile layouts defined in the following tables can be exported or imported using either tagged or fixed format files. These profile layouts are:

- Mailbox (requestor) profile
- Network security profile
- Network profile
- Network commands profile
- Activity log profile
- Application defaults profile
- User exit profile
- Language profile
- EDIFACT profile
- ICS profile
- UN/TDI profile
- UCS profile
- X12 profile
- Continuous receive profile
- CICS performance profile
- MQSeries queue profile

Mailbox (requestor) profile (REQPROF-P2)

This record defines the mailbox (requestor) settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = REQPROF
REQID	12-27	16	Char	Mailbox (requestor) ID
NETID	28-35	8	Char	Network ID
ACCTID	36-67	32	Char	Network account number
USERID	68-99	32	Char	Network user ID
NETPASS	100-115	16	Char	Network password
MSGCLS	116-123	8	Char	Network message user class
FILEID	124-131	8	Char	Transaction receive ddname
NETCLS	132-132	1	Char	Network classification
NETCHG	133-133	1	Char	Network charges code
NETACK	134-134	1	Char	Network acknowledgment code
NETVCHK	135-135	1	Char	Validate destination
NETRETN	136-138	3	Char	Message retention
NETEDIO	139-139	1	Char	EDI processing option
NETEDIP	140-140	1	Char	EDI processing override
STGFMT OV	141-141	1	Char	Storage format override
STGFMT	142-142	1	Char	Storage format

Tag	Position	Length	Type	Description
NETCMDS	143-150	8	Char	Member name of network commands file
TIMEOUT	151-154	4	Dec	Command line timeout value
NETACKPG	155-162	8	Char	Program to handle network acknowledgments from remote network
ALTNETPH	163-194	32	Char	Alternate dial connection phone number
COMPRESS	195-195	1	Char	Compression
PRIORITY	196-196	1	Char	Delivery priority
DESCRIPT	197-226	30	Char	Member description

Network security profile (SECUPROF-P2)

This record defines the network security settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = SECUPROF
SECID	12-19	8	Char	Security ID
SECORIGN	20-35	16	Char	Security originator
SECRECIP	36-51	16	Char	Security recipient
AUTTYPE	52-52	1	Char	Authorization type
AUTCODE	53-53	1	Char	Authorization code
ENCTYPE	54-54	1	Char	Encryption code
FILTYPE	55-55	1	Char	Filtering type
ENCPROG	56-63	8	Char	Encryption program
AUTPROG	64-71	8	Char	Authentication program
COMPROG	72-79	8	Char	Compression program
FILPROG	80-87	8	Char	Filtering program
BUFSIZE	88-92	5	Dec	Buffer size
DESCRIPT	93-122	30	Char	Description

Network profile (NETPROF-P2)

This record defines the network settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = NETPROF
NETID	12-19	8	Char	Network ID
NETNAME	20-49	30	Char	Network name
COMMRTN	50-57	8	Char	Communication routine name
NETPROG	58-65	8	Char	Network send/receive program name
NETPARM	66-122	57	Char	Network program parameters
CMDINDD	123-130	8	Char	Network program command input file name
CMDRECLN	131-134	4	Dec	Network command record length
QFILEDD	135-142	8	Char	TD queue file name
QRECLN	143-146	4	Dec	Transaction data record length
TIMEZONE	147-151	5	Char	Time zone
SYSTYPE	152-159	8	Char	System type
SYSLVL	160-163	4	Char	System level
MSGTXTH	164-164	1	Char	Message text header character
CMDOUTDD	165-172	8	Char	Network program command output file name
MSGHNDLR	173-180	8	Char	Program to process messages
NETSEQ	181-185	5	Char	Sequence number for network
NETAKFILE	186-193	8	Char	File for network acknowledgements
NETPHONE	194-225	32	Char	Dial connection phone number
SCRIPT	226-233	8	Char	Communication script name
DESCRIPT	234-263	30	Char	Description

Network commands profile (NETOP-P2)

This record defines the network commands being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = NETOP
NETID	12-19	8	Char	Network ID
NETOP	20-27	8	Char	Network command
NETFS	28-35	8	Char	Operation sequence number

Tag	Position	Length	Type	Description
BLKNM	36-43	8	Char	Block name
BLKPOS	44-47	4	Dec	Block position
CMDSEQ	48-51	4	Dec	Command line sequence number
CMDPOS	52-55	4	Dec	Command field position
CMDLEN	56-59	4	Dec	Command field length
CMDVAL	60-117	58	Char	Command field value
DESCRIPT	118-147	30	Char	Description

Activity log profile (ACTLOGS-P2)

This record defines the activity log settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = ACTLOGS
APPLID	12-19	8	Char	Application ID
LOGFLNM	20-27	8	Char	Logical name of file
PRGNAME	28-35	8	Char	File handler program name (EDIELAD)
LANG	36-36	1	Char	Language type of log handler (always C)
LOGFLAG	37-38	2	Char	Log profile activity
DESCRIPT	39-68	30	Char	Description

Application defaults profile (APPDEFS-P2)

This record defines the application default settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = APPDEFS
APPLICID	12-19	8	Char	Application ID
MEMBER	20-27	8	Char	ACTLOGS profile name
MRFLAG	28-28	1	Char	Management reporting active
TSFLAG	29-29	1	Char	Transaction Store active
TIFLAG	30-30	1	Char	Transaction image wanted
MONITOR	31-38	8	Char	CICS performance monitor exit name
USFLAG	39-39	1	Char	Test transaction with production usage
LOGENV	40-40	1	Char	Log EDI standard data

Tag	Position	Length	Type	Description
FAIFLAG	41-41	1	Char	Functional acknowledgment image
ELFLAG	42-42	1	Char	Envelope log active
DESCRIPT	43-72	30	Char	Description
ALPHANUM	73-80	8	Char	Name of override ALPHANUM table
CHARSET	81-88	8	Char	Name of override CHARSET table
CTRLYY	89-90	2	Char	Century control year

User exit profile (ADAMCTL-P2)

This record defines the user exits being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = ADAMCTL
LOGPGNM	12-19	8	Char	Logical name of program
LOADMOD	20-27	8	Char	Load module name
LANG	28-28	1	Char	Language used in user program
ADUEFLD	29-29	1	Char	Field exit type
ADUEPST	30-30	1	Char	Post-translate exit type
ADUEPRE	31-31	1	Char	Pre-translate exit type
ADUEENC	32-32	1	Char	Encryption exit type
ADUEAUT	33-33	1	Char	Authentication exit type
ADUECMP	34-34	1	Char	Compression exit type
ADUEFLT	35-35	1	Char	Filtering exit type
ADUEMNT	36-36	1	Char	Monitor exit type
ADUECOM	37-37	1	Char	Communication exit type
ADUEMSG	38-38	1	Char	Message process exit type
ADUEPTP	39-39	1	Char	Point-to-point exit type
ADUEENV	40-40	1	Char	Envelope exit type
DESCRIPT	41-70	30	Char	Description

Language profile (LANGPROF-P2)

This record defines the language settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = LANGPROF
LANGID	12-17	6	Char	Language ID
CPAGE	18-22	5	Char	Code page ID
DMASK	23-32	10	Char	Date edit/display mask
TMASK	33-40	8	Char	Time edit/display mask
DECNOT	41-41	1	Char	Edit/display decimal notation
SIGN	42-43	2	Char	Preferred negative sign (display)
FOLD	44-44	1	Char	Substitute for non-displayed character
DESCRIPT	45-74	30	Char	Description

EDIFACT (E envelope) profile (E-P2)

This record defines the EDIFACT envelope settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = E
EDIFKEY	12-19	8	Char	Profile name
UNB01	20-23	4	Char	Syntax identifier
UNB02	24-24	1	Char	Syntax version number
UNB03	25-59	35	Char	Sender ID
UNB04	60-63	4	Char	Sender ID qualifier
UNB05	64-77	14	Char	Reverse routing address
UNB06	78-112	35	Char	Recipient ID
UNB07	113-116	4	Char	Recipient ID qualifier
UNB08	117-130	14	Char	Routing address
UNB09	131-136	6	Char	Interchange date
UNB10	137-140	4	Char	Interchange time
UNB11	141-154	14	Char	Interchange control reference
UNB12	155-168	14	Char	Password
UNB13	169-170	2	Char	Password qualifier
UNB14	171-184	14	Char	Application reference
UNB15	185-185	1	Char	Processing priority

Tag	Position	Length	Type	Description
UNB16	186-186	1	Char	Acknowledgment request
UNB17	187-221	35	Char	Communications agreement ID
UNB18	222-222	1	Char	Test indicator
UNZ01	223-228	6	Char	Interchange control count
UNZ02	229-242	14	Char	Interchange control reference
UNG01	243-248	6	Char	Functional group ID
UNG02	249-283	35	Char	Sender ID
UNG03	284-287	4	Char	Sender ID qualifier
UNG04	288-322	35	Char	Recipient ID
UNG05	323-326	4	Char	Recipient ID qualifier
UNG06	327-332	6	Char	Functional group date
UNG07	333-336	4	Char	Functional group time
UNG08	337-350	14	Char	Functional group reference number
UNG09	351-352	2	Char	Controlling agency
UNG10	353-355	3	Char	Message version
UNG11	356-358	3	Char	Message release
UNG12	359-364	6	Char	Association assigned
UNG13	365-378	14	Char	Application password
UNE01	379-384	6	Char	Number of messages
UNE02	385-398	14	Char	Functional group reference number
UNH01	399-412	14	Char	Message reference number
UNH02	413-418	6	Char	Message type
UNH03	419-421	3	Char	Message version number
UNH04	422-424	3	Char	Message release number
UNH05	425-426	2	Char	Controlling agency
UNH06	427-432	6	Char	Association assigned
UNH07	433-467	35	Char	Common access reference
UNH08	468-469	2	Char	Sequence of transfer
UNH09	470-470	1	Char	First and last message in sequence
UNT01	471-476	6	Char	Number of segments in message
UNT02	477-490	14	Char	Message reference number
DESCRIPT	491-520	30	Char	Description

ICS (I envelope) profile (I-P2)

This record defines the ICS envelope settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = I
ICSKEY	12-19	8	Char	Profile name
ICS01	20-21	2	Char	Subelement separator
ICS02	22-25	4	Char	Control standards ID
ICS03	26-30	5	Char	Control version number
ICS04	31-32	2	Char	Sender ID qualifier
ICS05	33-47	15	Char	Information sender ID
ICS06	48-49	2	Char	Receiver ID qualifier
ICS07	50-64	15	Char	Information receiver ID
ICS08	65-70	6	Char	Interchange date
ICS09	71-74	4	Char	Interchange time
ICS10	75-83	9	Char	Interchange control number
ICE01	84-89	6	Char	Number of groups
ICE02	90-98	9	Char	Interchange control number
GS01	99-100	2	Char	Functional group ID
GS02	101-115	15	Char	Application sender code
GS03	116-130	15	Char	Application receiver code
GS04	131-136	6	Char	Functional group date
GS05	137-144	8	Char	Functional group time
GS06	145-153	9	Char	Functional group control number
GS07	154-155	2	Char	Responsible agency code
GS08	156-167	12	Char	Version/release/industry ID
GE01	168-173	6	Char	Number of included sets
GE02	174-182	9	Char	Functional group control number
ST01	183-185	3	Char	Transaction set ID
ST02	186-194	9	Char	Transaction set control number
SE01	195-204	10	Char	Number of included segments
SE02	205-213	9	Char	Transaction set control number
DESCRIPT	214-243	30	Char	Description

UN/TDI (T envelope) profile (T-P2)

This record defines the UN/TDI envelope settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = T
TDIKEY	12-19	8	Char	Profile name
STX01	20-23	4	Char	Syntax identifier
STX02	24-24	1	Char	Syntax version number
STX03	25-38	14	Char	Sender code
STX04	39-73	35	Char	Sender name
STX05	74-87	14	Char	Recipient code
STX06	88-122	35	Char	Recipient name
STX07	123-128	6	Char	Interchange date
STX08	129-134	6	Char	Interchange time
STX09	135-148	14	Char	Interchange control reference
STX10	149-162	14	Char	Recipient's reference/password
STX11	163-176	14	Char	Application reference
STX12	177-177	1	Char	Transmission priority code
END01	178-182	5	Char	Total number of messages
MHD01	183-194	12	Char	Message reference
MHD02	195-200	6	Char	Message type
MHD03	201-201	1	Char	Message version
MHD04	202-236	35	Char	Common access reference
MHD05	237-238	2	Char	Sequence of transfers
MHD06	239-239	1	Char	First and last transfers
MTR01	240-243	4	Char	Number of segments in message
DESCRIPT	244-273	30	Char	Description

UCS (U envelope) profile (U-P2)

This record defines the UCS envelope settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = U
UCSKEY	12-19	8	Char	Profile name
BG01	20-29	10	Char	Communication ID
BG02	30-39	10	Char	Communication password
BG03	40-54	15	Char	Application sender ID
BG04	55-69	15	Char	Application receiver ID
BG05	70-75	6	Char	Interchange date
BG06	76-81	6	Char	Interchange time
BG07	82-86	5	Char	Transmission control number
EG01	87-91	5	Char	Transmission control number
EG02	92-96	5	Char	Number of included groups
EG03	97-102	6	Char	Number of included transaction sets
EG04	103-112	10	Char	Number of included segments
GS01	113-114	2	Char	Functional group ID
GS02	115-129	15	Char	Application sender's code
GS03	130-144	15	Char	Application receiver's code
GS04	145-150	6	Char	Functional group date
GS05	151-158	8	Char	Functional group time
GS06	159-167	9	Char	Functional group control number
GS07	168-169	2	Char	Responsible agency code
GS08	170-181	12	Char	Version/release/industry ID
GE01	182-187	6	Char	Number of included sets
GE02	188-196	9	Char	Functional group control number
ST01	197-199	3	Char	Transaction set ID
ST02	200-208	9	Char	Transaction set control number
SE01	209-218	10	Char	Number of included segments
SE02	219-227	9	Char	Transaction set control number
DESCRIPT	228-257	30	Char	Description

X12 (X envelope) profile (X-P2)

This record defines the X12 envelope settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = X
X12KEY	12-19	8	Char	Profile name
ISA01	20-21	2	Char	Authorization information qualifier
ISA02	22-31	10	Char	Authorization information
ISA03	32-33	2	Char	Security information qualifier
ISA04	34-43	10	Char	Security information
ISA05	44-45	2	Char	Interchange sender ID qualifier
ISA06	46-60	15	Char	Interchange sender ID
ISA07	61-62	2	Char	Interchange receiver ID qualifier
ISA08	63-77	15	Char	Interchange receiver ID
ISA09	78-83	6	Char	Interchange date
ISA10	84-87	4	Char	Interchange time
ISA11	88-98	1	Char	Interchange standard ID
ISA12	89-93	5	Char	Interchange version ID
ISA13	94-102	9	Char	Interchange control number
ISA14	103-103	1	Char	Acknowledgment requested
ISA15	104-104	1	Char	Test indicator
ISA16	105-106	2	Char	Subelement separator
IEA01	107-111	5	Char	Number of included groups
IEA02	112-120	9	Char	Interchange control number
GS01	121-122	2	Char	Functional group ID
GS02	123-137	15	Char	Application sender's code
GS03	138-152	15	Char	Application receiver's code
GS04	153-158	6	Char	Group date
GS05	159-166	8	Char	Group time
GS06	167-175	9	Char	Functional group control number
GS07	176-177	2	Char	Responsible agency code
GS08	178-189	12	Char	Version/release/industry ID
GE01	190-195	6	Char	Number of included transaction sets
GE02	196-204	9	Char	Functional group control number
ST01	205-207	3	Char	Transaction set ID

Tag	Position	Length	Type	Description
ST02	208-216	9	Char	Transaction set control number
SE01	217-226	10	Char	Number of included segments
SE02	227-235	9	Char	Transaction set control number
DESCRIPT	236-265	30	Char	Description

Continuous receive profile (CONTRECV-P2 for CICS only)

This record defines the continuous receive settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = CONTRECV
CONTRCV	12-27	16	Char	Profile name
ACTIVE	28-28	1	Char	Profile active
REQID	29-44	16	Char	Mailbox (requestor) ID
TPNICKN	45-60	16	Char	Trading Partner nickname
MSGCLS	61-68	8	Char	Network message user class
TRANSLATE	69-69	1	Char	Deenvelope and translate
RAWDATA	70-70	1	Char	Generate raw data
PRINTNM	71-78	8	Char	Print file name
PRINTTYP	79-80	2	Char	Print file type
EXCPNM	81-88	8	Char	Exception file name
EXCPTP	89-90	2	Char	Exception file type
ADDLRECS	91-95	5	Char	Additional records
DENVONLY	96-96	1	Char	Deenvelope only
DELAYFA	97-97	1	Char	Delay functional acknowledgment enveloping
FATSQ	98-105	8	Char	Enveloped functional acknowledgment TS queue name
RESPNM	106-113	8	Char	Name of response application TD queue
RESPTP	114-115	2	Char	Type of response
USERFLD	116-131	16	Char	User field passed to response
APPLID	132-139	8	Char	Application ID
NLSID	140-145	6	Char	National language ID
SYNCPTS	146-146	1	Char	Allow syncpoints

Tag	Position	Length	Type	Description
DUPENVL	147-147	1	Char	Allow duplicate envelopes
NETAKONLY	148-148	1	Char	Process network acknowledgments
PURGEINT	149-152	4	Dec	Transaction purge interval
DESCRIPT	153-182	30	Char	Description
SAPUPDT	183-183	1	Char	SAP update
PAGE	184-184	1	Char	Pageable translation active flag

CICS performance profile (SYSPROF-P2 for CICS only)

This record defines the CICS performance settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-11	8	Char	Profile ID = SYSPROF
SYSID	12-19	8	Char	System ID
PERSACTV	20-20	1	Char	Persistent environment active
PERSIZE	21-24	4	Dec	Persistent environment size
PERSTHDS	25-26	2	Dec	Number persistent environment threads
DESCRIPT	27-56	30	Char	Description

MQSeries queue profile (MQSERIES-P2)

This record defines the MQSeries queue settings being used. The fields are described in the table below.

Tag	Position	Length	Type	Description
PROFID	4-011	8	Char	Profile ID = MQSERIES
QUEUEID	012-019	8	Char	Logical name associated with the queue name
REALNAME	20-67	48	Char	Actual MQSeries queue name
MGRNAME	68-115	48	Char	Override queue manager name
READFLAG	116-116	1	Char	Destructive read flag indicator
MQSYNCFLAG	117-117	1	Char	Syncpoint control
MAXMSGLN	118-125	8	Char	Maximum message length
DESCRIPT	126-155	30	Char	Description

DataInterchange Utility records format

This section describes the format of the DataInterchange Utility records, which are:

- Control (C)
- Data (D)
- End transaction and interchange (Z)
- Raw data
- Optional

Control (C) records

The C record format is used for both send and receive transactions. All fields are left-justified and case-sensitive. The expanded fields (position 67 and greater) are optional. Use these fields to provide the override values you want the translator or enveloper to use. The following table describes the record layout for translating C record data to EDI standard format.

Label	Position	Length	Type	Description
RECID	1-1	1	Char	Record ID = C
INTPID	2-36	35	Char	Internal trading partner ID
FORMATID	37-52	16	Char	Data format ID
TRANRC	53-56	4	Integer	Translator return code
TRANXRC	57-60	4	Integer	Translator extended return code
UTILRC	61-64	4	Integer	DataInterchange Utility return code
TESTIND	65-65	1	Char	Usage indicator
MUWIND	66-66	1	Char	Multiple D records indicator
XPANDED	67-67	1	Char	For send only. Expanded control block indicator
ITYPE	68-68	1	Char	For send only. Return information record indicator
ETYPE	69-69	1	Char	For send only. Return envelope header record indicator
GTYPE	70-70	1	Char	For send only. Return group header record indicator
TTYPE	71-71	1	Char	For send only. Return transaction set header record indicator
QTYPE	72-72	1	Char	For send only. Return queuing totals record indicator
ISID	73-107	35	Char	Interchange sender ID
IRID	108-142	35	Char	Interchange receiver ID
IVERREL	143-147	5	Char	Interchange version and release number
ISPW	148-161	14	Char	Interchange password

Label	Position	Length	Type	Description
IAPREF	162-175	14	Char	Interchange application reference
GSID	176-210	35	Char	Group application sender ID
GRID	211-245	35	Char	Group application receiver ID
GVER	246-257	12	Char	Group version
GREL	258-269	12	Char	Group release number
GAPW	270-283	14	Char	Group password
TVER	284-289	6	Char	Transaction version
TREL	290-295	6	Char	Transaction release number
HOLDFLAG	296-296	1	Char	Held status indicator
BNDLFLAG	297-297	1	Char	Bundle indicator
ROUTCODE	298-300	3	Char	Generic routing code
ISYNTAXID	301-304	4	Char	Interchange syntax ID for E and T envelopes
ISYNTAXVER	305-305	1	Char	Interchange syntax version for E and T envelopes
ISIDQUAL	306-309	4	Char	Interchange sender ID qualifier
ISENDNAME	310-323	14	Char	Interchange sender name for T envelope) Application sender code for U envelope
IREVROUT	324-337	14	Char	Interchange reverse routing for E envelope
IRIDQUAL	338-341	4	Char	Interchange receiver ID qualifier
IRECVNAME	342-355	14	Char	Interchange receiver name for T envelope) Application receiver code for U envelope
IROUTEADDR	356-369	14	Char	Interchange routing address for E envelope
ISTDID	370-373	4	Char	Interchange standard ID for I and X envelopes
IPRIOR	374-374	1	Char	Interchange priority for E and T envelopes
ICOMMAGREE	375-409	35	Char	Interchange communication agreement for E envelope
GSIDQ	410-413	4	Char	Group application sender ID qualifier
GRIDQ	414-417	4	Char	Group application receiver ID qualifier
GRESAG	418-419	2	Char	Group responsible agency code

Label	Position	Length	Type	Description
DUPTRANS	420-420	1	Char	For receive only. Duplicate transaction indicator
FORCEC	421-421	1	Char	For send only. Force C record format on send translation
APPLTPID	422-437	16	Char	Application trading partner
EDITPID	438-453	16	Char	EDI trading partner
RSRVD1	454-512	59	Char	Reserved
CUSERDATA	513-768	256	Char	User data area
RSRVD2	769-1024	256	Char	Reserved

Control record label descriptions

RECID

The transaction set control record with a value of **C**.

INTPID

The internal trading partner ID used to define the map.

FORMATID

The data format ID used to describe the application data.

TRANRC

The translation status of the transaction. The code is in binary format.

TRANXRC

Defines the translation status of the transaction. The code is in binary format.

UTILRC

For send translation, indicates whether a transaction is written to the exception file in the UTILRC field. The DataInterchange Utility returns the binary value **8** in UTILRC if it writes the transaction to the exception file.

For receive translation, UTILRC is always a binary **0**.

TESTIND

Indicates the usage/rule used for the transaction. Valid values are:

- I** Information transaction. Use an information usage/rule. If an information usage/rule is not found, use a production usage/rule. Even when a production usage/rule is used, the transaction is flagged as an information transaction.
- P** Production transaction. Use only a production usage/rule (default).
- T** Test transaction. Use a test usage/rule. If a test usage/rule is not found, use a production usage/rule. Even when a production usage/rule is used, the transaction is flagged as a test transaction.
- U** The translator should determine if the transaction is test, information, or production based on the usage/rule found. If a test usage/rule is found, the transaction is a test transaction, and value of **T** is returned. If an information usage/rule is found, the transaction is an information transaction, and a value of **I** is returned. If only a production usage/rule is found, the transaction is a production transaction, and a value of **P** is returned.

MUWIND

Indicates whether the transaction contains a single D record or multiple D records are involved in the transaction. Valid values are:

- Y** Multiple D records are involved in the transaction.
- N** A single D record is involved in the translation.

XPANDED

For send translation only. Indicates the version of the C record format being used (which fields are contained in the record). Receive translation always writes all fields through column 1024. Valid values are:

This value: Specifies the C record contains fields through:

- Y** Column 297
- N** Column 67
- 1** Column 432
- 2** Column 1024

ITYPE

Indicates whether an information record is written to the exception/tracking file. Generally used for send translation. On receive translation, the field will contain the same values as shown in the discussion of "DUPTRANS" on page 267. Valid values are:

- Y** Writes an information record to the exception/tracking file for each translated transaction. For more information, see "Information (I) records" on page 273.
- N** Does not write an information record.

ETYPE

For send translation only. Indicates whether an interchange header image is written to the exception/tracking file. Valid values are:

- Y** Writes an image of the interchange header segment to the exception/tracking file for each interchange processed. For more information, see “Interchange header (E) records” on page 274.
- N** Does not write an image of the interchange header segment.

GTYPE

For send translation only. Indicates whether a group header image is written to the exception/tracking file. Valid values are:

- Y** Writes an image of the group header segment to the exception/tracking file for each group processed. For more information, see “Group header (G) records” on page 275.
- N** Does not write an image of the group header segment.

TTYE

For send translation only. Indicates whether a transaction header image is written to the exception/tracking file. Valid values are:

- Y** Writes an image of the transaction header segment to the exception/tracking file for each transaction processed. For more information, see “Transaction set header (T) records” on page 275.
- N** Does not write an image of the transaction header segment.

QTYPE

For send translation only. Indicates whether a record containing totals relative to an interchange is written to the exception/tracking file. Valid values are:

- Y** Writes information relative to totals in an interchange to the exception/tracking file for each interchange processed. For more information see “Queuing totals (Q) records” on page 275.
- N** Does not write interchange totals information.

ISID

Overrides the ID provided in the envelope profile member or the trading partner profile member. It maps to a type **IS** standard data element. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB03	E
ICS05	I

This value: **Overrides this envelope type:**

STX03 or STX04 T

BG03 U

ISA06 X

IRID

Overrides the ID provided in the trading partner profile or envelope profile member. It maps to a type **IR** standard data element. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value: **Overrides this envelope type:**

UNB06 E

ICS07 I

STX05 or STX06 T

BG04 U

ISA08 X

IVERREL

Overrides the version and release provided in the envelope profile member. It maps to types **VR** and **LV** standard data elements. DataInterchange provides the equivalent data on receive translation.

ISPW

Overrides the password that the trading partner profile member provides. It maps to a type **PW** standard data element. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value: **Overrides this envelope type:**

UNB12 E

STX10 T

ISA04 X

IAPREF

Overrides the application reference ID provided in the envelope profile member. DataInterchange uses this reference as the message user class for EDIFACT and UN/TDI. This reference points to a type **AP** standard data element. As distributed by DataInterchange, no fields contain the **AP** data type. However, the following fields are frequently customized to have a data type of **AP**. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value: **Overrides this envelope type:**

UNB14 E

STX11 T

GSID

Overrides the data format ID provided in the map. If the map provides the name of an envelope profile member, this entry overrides the sender ID that the envelope profile member provides. The group application sender ID maps to a type **AS** standard data element. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNG02	E
GS02	X
GS02	I
GS02	U

GRID

Overrides the trading partner application name provided in the map. If the map provides the name of an envelope profile member, this entry overrides the receiver ID that the envelope profile member provides. The group application ID maps to a type **AR** standard data element. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNG04	E
GS03	X
GS03	I
GS03	U

GVER

Overrides the group version provided in the envelope profile member. The value in the group version maps to a type **VR** standard data element. DataInterchange provides the equivalent data on receive translation.

GREL

Overrides the group release number provided in the envelope profile member. The group release number maps to type **IV** standard data element. DataInterchange provides the equivalent data on receive translation.

GAPW

Overrides the group password provided in the trading partner usages/rules. This usage/rule overrides the password in the envelope profile member. The group password maps to a type **PW** standard data element. DataInterchange provides the equivalent data on receive translation. Valid value is:

This value:	Overrides this envelope type:
UNG13	E

TVER

Overrides the transaction version provided in the envelope profile member. The transaction version maps to a type **VR** standard data element. DataInterchange provides the equivalent data on receive translation.

TREL

Overrides the transaction release number provided in the envelope profile member. The transaction release number maps to a type **IV** standard data element. DataInterchange provides the equivalent data on receive translation.

HOLDFLAG

Specifies if the transaction is in hold status. DataInterchange provides the equivalent data on receive translation. Valid values are:

- | | |
|----------|--|
| Y | Places the transaction in held status. |
| N | Does not hold the transaction. Transactions that are not on hold and do not have a date specified for enveloping are available immediately for enveloping and sending. |

BNDLFLAG

Specifies if the transaction starts a bundle. DataInterchange provides the equivalent data on receive translation. Valid values are:

- | | |
|----------------|---|
| Y | Starts a group of related transactions (a bundle). All transactions that follow are part of the bundle until the translator encounters another transaction with this field set to Y or N . |
| N | Ends the bundle without starting a new one. |
| (blank) | Continues processing as usual. The translator ends a bundle automatically if the data forces the start of a new group or interchange envelope. Any action that changes the transaction or store status of one member of the bundle (such as envelope, send, or hold) is applied to all members of the bundle. |



NOTE: If the current trading partner is not using functional groups (value of **N** in the **Functional group** field of the trading partner profile), DataInterchange ignores the changes in the data. A new group is created but does not end the bundle.

ROUTCODE

For send translation only. A three-character generic routing code provided by the application and used by DataInterchange to select a generic send usage/rule. A blank specifies a default generic send usage/rule. DataInterchange provides the equivalent data on receive translation.

ISYNTAXID

Overrides the interchange syntax ID provided in the envelope profile member. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB01	E
STX01	T

ISYNTAXVER

Overrides the interchange syntax version provided in the envelope profile member. Valid values are:

This value:	Overrides this envelope type:
UNB02	E
STX02	T

ISIDQ

Overrides the interchange sender ID qualifier provided in the envelope profile member or trading partner profile member. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB04	E
ICS04	I
ISA05	X

ISENDNAME

Overrides the interchange sender name provided in the envelope profile member. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
STX04	T
UCS03	U (If not IS data type)

IREVROUT

Overrides the interchange reverse routing provided in the envelope profile member. DataInterchange provides the equivalent data on receive translation. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
UNB05	E

IRIDQ

Overrides the interchange receiver ID qualifier provided in the trading partner profile or envelope profile member. Valid values are:

This value:	Overrides this envelope type:
UNB06	E
ICS06	I
ISA07	X

IRECVNAME

Overrides the interchange receiver name provided in the envelope profile member. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
STX06	T
UCS04	U (If not IR data type)

IROUTADDR

Overrides the interchange routing address provided in the envelope profile member. This entry must be left-justified. Valid value is:

This value:	Overrides this envelope type:
UNB08	E

ISTDID

Overrides the interchange standard ID provided in the envelope profile member. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
ICS02	I
ISA11	X

IPRIOR

Overrides the interchange processing priority defined in the envelope profile member. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB15	E
STX12	T

ICOMMAGREE

Overrides the interchange communication agreement provided in the envelope profile member. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNB17	E

GSIDQ

Overrides the group sender ID qualifier provided in the envelope profile member. DataInterchange provides the equivalent data on receive translation. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
UNG03	E

GRIDQ

Overrides the group receiver ID qualifier provided in the envelope profile member. This entry must be left-justified. Valid values are:

This value:	Overrides this envelope type:
UNG05	E

GRES PAG

Overrides the group responsible agency code and controlling agency provided in the envelope profile member. DataInterchange provides the equivalent data on receive translation. Valid values are:

This value:	Overrides this envelope type:
UNG09	E
GS07	I
GS07	U
GS07	X

DUPTRANS

For receive translation only. Specifies if a transaction is part of a duplicate envelope. Valid values are:

Y	Part of a duplicate envelope
N	Not part of a duplicate envelope

FORCEC

For send translation only. Forces a C record to be written. Valid values are:

Y	Always writes a C record
N or (blank)	Only writes a C record when an error occurs (default)

APPLTPID

The trading partner nickname of the application trading partner as defined in the trading partner profile.

EDITPID

The trading partner nickname of the EDI trading partner as defined in the trading partner profile.

RSVD1

Reserved for DataInterchange.

CUSERDATA

The value in this field is copied to the TRCB where it can be modified by user exits. Before any C record is output by DataInterchange, the value in this field is copied from the equivalent field in the TRCB. On receive translation, the value in the TRCB can be set using the DataInterchange reserved variable DICUSERDATA. DataInterchange does not use this field.

RSVD2

Reserved for DataInterchange.

Data (D) records

There are two formats for data (D) records. One format is used when all the data for a transaction is provided by a single structure. The other format is used when data for a transaction is provided by multiple structures. The format of D records is described in the following tables.

For Fixed-to-Fixed mapping, when there is no target data format, the STRUCTNAME is the segment ID value from the EDI standard transaction definition, and D records are always output in the format described for single structures on page 269.

When translating D records, use the name in the **ID field**, and populate ISA and GS segments with the values passed in the C record.



NOTE: The largest record your application or the translator can handle is 32000 bytes, which is the largest LRECL allowed for QSAM files. During translate-to-application processing, DataInterchange divides any records that are larger than 32000 bytes into one or more X records, and uses a D record as the last record of the structure. For example, a structure that is 80000 bytes in length requires two X records (64000 bytes) followed by one D record (16000 bytes). During translate-to-EDI-standard processing, the program that creates the application file must create X and D records for structures exceeding 32000 bytes in length. Define the application file as variable-blocked (VB) if the application file will contain X records.

Data record format - single structure

Label	Position	Length	Type	Description
RECID	1-1	1	Char	Record ID
DATARCD	2-32750	32749	Char	Application transaction data

Data record label descriptions (single structure)

The following are descriptions of the labels for the data record when data structures are passed together.

RECID

Specifies if a transaction exceeds 32000 bytes. Valid values are:

- D** An entire transaction, or the last record of a transaction that exceeds 32000 bytes
- X** The first and middle records of a transaction that exceed 32000 bytes

DATARCD

Specifies application transaction data.

Data record format - multiple structures

Label	Position	Length	Type	Description
RECID	1-1	1	Char	Record ID
STRUCNAM	2-17	16	Char	Structure name
DATARCD	18-32750	32733	Char	Application transaction data

Data record label descriptions (multiple structures)

The following are descriptions of the labels for the data record when data structures are passed separately.

RECID

Specifies if a transaction exceeds 32000 bytes. Valid values are:

- D** An entire transaction, or the last record of a transaction that exceeds 32000 bytes
- X** The first and middle records of a transaction that exceed 32000 bytes

STRUCNAME

The structure name defined in the data format for this transaction.

DATARCD

Specifies application transaction data.

End transaction and interchange (Z) records

The Z record is optional and is used in the MVS environment when the size of an interchange is being controlled by the application, or in the CICS environment when recoverable resources are being used. The Z record marks the end of:

- The application data for a transaction. If a Z record is not present, DataInterchange can only detect the end of a transaction from reading the next transaction's C record.

If your input data is in recoverable intrapartition TD queues, using the Z record keeps the reading of the C record for the next transaction from being part of the syncpoint interval for the current transaction. The Z record provides a formal end for a transaction rather than the implied transaction end achieved when the next C record is read.

- The application data for a transaction and indicates that the current transaction is the last transaction for an interchange when a Z record with the ENDINTERCH field set to 1 (Z1) is present. If a Z1 record is not present, DataInterchange can only detect the end of an interchange from reading the next transaction's C record.

If your input data is in recoverable intrapartition TD queues, using a Z1 record keeps the reading of the C record for the next transaction from being part of the syncpoint interval for the current interchange. The Z1 record provides a formal end for an interchange rather than the implied end achieved when the next C record is read. The Z1 record can be used to artificially limit the size of an interchange. A Z1 record causes the current interchange to be completed. A new interchange starts with the next transaction.

Z and Z1 records are only necessary when using the C and D record format, and when the conditions mentioned below are true. Otherwise, using Z and Z1 records is optional. The conditions for use are:

- The DataInterchange Utility is being executed in the CICS environment.
- Application data is given to the DataInterchange Utility in recoverable intrapartition TD queues.
- DataInterchange is allowed to issue CICS SYNCPOINT commands.

Z record format

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = Z
ENDINTERCH	2	1	Char	End of interchange indicator

Z record label descriptions

RECID

A value of **Z** identifies this record as a transaction or interchange terminator.

ENDINTERCH

A value of **1** indicates the preceding transaction is the last in the current interchange. The next C record begins a new transaction and new interchange.

Raw data records

The table below describes the format of raw data records. The data format must indicate the position, length, and type of record ID field within the data. The data format can also provide the field in the application data that contains the internal trading partner ID value. For TRANSLATE TO STANDARD processing, the data format ID is provided by the **RAWFMTID** keyword on the PERFORM command. During TRANSLATE TO APPLICATION processing, the translator automatically supplies the record ID values. The internal trading partner ID value is taken from the trading partner receive usage/rule when raw data is requested.

DataInterchange can create raw data output during TRANSLATE TO STANDARD processing with Fixed-to-Fixed mapping. Record ID values and the internal trading partner ID default values are used but may be overridden during mapping.

Raw data record format

Label	Position	Length	Type	Description
DATARCD	1-32750	32750	Char	Application transaction data

Optional records

The following optional record types are available:

- Envelope header (E)
- Group header (G)
- Information (I)
- Queuing totals (Q)
- Transaction set header (T)

You can request optional records using the **TYPE** fields in the control record (see “Control (C) records” on page 257), on the Additional Records panel, or by using the OPTRECS keyword (see “OPTRECS” on page 150) in one of the following commands:

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

Different optional records are available for different commands. The commands and the record types they support are:

Command	Record type				
	E	G	I	Q	T
DEENVELOPE DEENVELOPE AND SEND	X	X	X	X	X
ENVELOPE ENVELOPE AND SEND	X	X	X	X	X
RECEIVE AND DEENVELOPE RECEIVE AND TRANSLATE	X	X	X	X	X
REENVELOPE REENVELOPE AND SEND	X	X	X	X	X
RETRANSLATE TO APPLICATION					
TRANSLATE AND ENVELOPE	X	X	X	X	X
TRANSLATE AND SEND	X	X	X	X	X
TRANSLATE TO APPLICATION			X		X
TRANSLATE TO STANDARD			X		

For TRANSLATE TO STANDARD operations, the DataInterchange Utility writes the optional records to a file as described below:

- Writes the records to FFSTRAK if it exists.
- Writes the records to FFSEXCP if FFSTRAK does not exist.
- Writes no records if FFSTRAK does not exist and raw data (RAWFMTID keyword) is requested.

For TRANSLATE TO APPLICATION operations, the DataInterchange Utility writes the optional records to a file as described below:

- Writes the records to the application output file if C and D records are requested.
- Writes the records to FFSEXCP if the raw data records are requested (RAWDATA keyword).

For DEENVELOPE operations, the DataInterchange Utility writes the optional records to the FFSEXCP file.

Information (I) records

The table below describes the format and contents of the information (I) record. You can request an information record on the Optional Record Options panel, in the **ITYPE** field in the control record, or by using the **OPTRECS** keyword in a command. DataInterchange returns one record for each transaction with the values that were present at the time the transaction was translated.

Label	Position	Length	Type	Description
RECID	1-1	1	Char	Record ID = I
IHXCTL	2-15	14	Char	Interchange header control number
ISID	16-50	35	Char	Interchange sender ID
IRID	51-85	35	Char	Interchange receiver ID
IDATE	86-91	6	Char	Interchange date
ITIME	92-97	6	Char	Interchange time
IVERREL	98-102	5	Char	Interchange version/release
IGT	103-108	6	Char	For sending, the total number of groups in the interchange at the current time For receiving, the total number of groups processed so far in the interchange
ITT	109-114	6	Char	For sending, the total number of transactions in the interchange at the current time For receiving, the total number of transactions processed so far in the interchange
IST	115-124	10	Char	For sending, the total number of segments in the interchange at the current time For receiving, the total number of segments processed so far in the interchange
IBT	125-132	8	Char	For sending, the total number of bytes in the interchange at the current time For receiving, the total number of bytes processed so far in the interchange
ISPW	133-146	14	Char	Interchange password
IAPREF	147-160	14	Char	Interchange application reference
GHXCTL	161-174	14	Char	Control number of group header
GFGID	175-180	6	Char	Group functional group ID
GSID	181-215	35	Char	Group application sender ID

Label	Position	Length	Type	Description
GRID	216-250	35	Char	Group application receiver ID
GDATE	251-256	6	Char	Group date
GTIME	257-262	6	Char	Group time
GVER	263-274	12	Char	Group version
GREL	275-286	12	Char	Group release
GTT	287-292	6	Char	For sending, current group transaction total For receiving, total transactions in group
GAPW	293-306	14	Char	Group password
THXCTL	307-320	14	Char	Control number of transaction set header
TTC	321-326	6	Char	Transaction code
TVER	327-332	6	Char	Transaction version
TREL	333-338	6	Char	Transaction release
TST	339-348	10	Char	Transaction segment total
AC	349-383	35	Char	Application control field value
THANDLE	384-403	20	Char	Expanded value of the handle assigned to this transaction in the Transaction Store
TPNICKN	404-419	16	Char	Trading partner nickname
GDATE2	420-427	8	Char	Group date with century
RESERVED	428-483	56	Char	Reserved for future use

Interchange header (E) records

The table below shows the format and contents of the interchange (E) header record. You can request an interchange header record using the **ETYPE** field in the control record or the **OPTRECS** keyword in a command. DataInterchange returns one record for each interchange that is created or received.

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = E
EDATA	2-256	255	Char	Standard interchange header image

Group header (G) records

The table below shows the format and contents of the group header (G) record. You can request a group header record using the **GTYPE** field in the control record or the **OPTRECS** keyword in a command. DataInterchange returns one record for each group created or received.

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = G
GDATA	2-256	255	Char	Functional group header image

Transaction set header (T) records

The table below shows the format and contents of the transaction set header (T) record. You can request a transaction header record using the **TTYPE** field in the control record or the **OPTRECS** keyword in a command. DataInterchange returns one record for each transaction created or received.

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = T
TDATA	2-256	255	Char	Transaction set header image

Queuing totals (Q) records

The table below shows the format and contents of the queuing totals record. You can request a queuing totals record using the **QTYPE** field in the control record or the **OPTRECS** keyword in a command. DataInterchange returns one record to the exception file each time an interchange is queued or deenveloped.

Label	Position	Length	Type	Description
RECID	1	1	Char	Record ID = Q .
QBT	2-9	8	Char	Number of bytes queued for the envelope.
QST	10-19	10	Char	Number of segments in the envelope.
QTT	20-25	6	Char	Number of transactions in the envelope.
QGT	26-31	6	Char	Number of groups in the envelope.
QDSNAME	32-87	56	Char	Physical data set name from which transactions were read or to which transactions were written. The name is terminated with a NULL character (X'00').

Management reporting

The management reporting data extracts are formatted as sequential files containing fixed-length (1024 bytes) records. All data extracts are written to the EDIQUERY file. Since other commands use the EDIQUERY file, you should define the EDIQUERY file as variable block 32756 rather than fixed block 1024 to accommodate commands that output larger records, such as the image records created by the ENVELOPE DATA EXTRACT and TRANSACTION DATA EXTRACT commands.

The output is tabular with columns representing categories of information and rows containing the actual data entries. The formats of the data extracts are described in the following tables.

Trading partner profile data extract

This record defines the trading partner profile settings being used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = TPI
Trading Partner Nickname	4-19	16	Char	Trading partner ID in DataInterchange
Company Name	20-59	40	Char	Company name of the trading partner
Address line 1	60-99	40	Char	First line of the company's address
Address line 2	100-139	40	Char	Second line of the company's address
Comment line 1	140-179	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Comment line 2	180-219	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Contact Name	220-249	30	Char	Name of a contact at this trading partner
Contact Phone	250-274	25	Char	Phone number of the contact
Network ID	275-282	8	Char	ID of the network used to communicate with this trading partner
Interchange ID	283-317	35	Char	Receiver ID used in interchanges to this trading partner
Account ID	318-349	32	Char	Network account ID
User ID	350-381	32	Char	Network user ID
Direction	382-382	1	Char	Direction of the (inbound or outbound)
Date of Last Transmission	383-390	8	Char	Interchange date of last transmission to this trading partner

Label	Position	Length	Type	Description
Interchange control number	391-404	14	Char	Interchange control number of last transmission to this trading partner
Group control number	405-418	14	Char	Group control number of the last transmission to this trading partner
Transaction control number	419-432	14	Char	Transaction control number of the last transaction to this trading partner
Filler	433-1024	592	Char	Filler for expansion

Trading partner capability data extract

This record defines the trading partner capability settings being used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = TPC
Trading Partner Nickname	4-19	16	Char	Trading partner ID in DataInterchange
Internal Trading Partner ID	20-54	35	Char	ID used for this trading partner internally (customer number, supplier number)
Company Name	55-94	40	Char	Company name of the trading partner
Address line 1	95-134	40	Char	First line of the company's address
Address line 2	135-174	40	Char	Second line of the company's address
Comment line 1	175-214	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Comment line 2	215-254	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Direction	255-255	1	Char	Direction of the (inbound or outbound)
Standard ID	256-263	8	Char	ID of the EDI standard (X12, EDIFACT)
Version ID	264-265	2	Char	Version of the EDI standard
Release ID	266-267	2	Char	Release of the EDI standard
Description	268-317	50	Char	Description of the EDI standard/version/release
Transaction ID	318-325	8	Char	Transaction ID (such as ORDERS, DISPATCH, 850, 860)

Label	Position	Length	Type	Description
Map ID	326-341	16	Char	Trading partner map name
Measurement date	342-349	8	Char	Date testing or production started with this map/trading partner combination
Measurement ID	350-353	4	Char	Type of statistic
Total number of transactions	354-368	15	Char	Total number of test or production transactions exchanged with this map/trading partner combination
Total errors	369-383	15	Char	Total number of test or production transactions exchanged with this map/trading partner combination that had errors
Appl Trading Partner	384-399	16	Char	Application trading partner ID
Filler	400-1024	625	Char	Filler for expansion

Network activity data extract

This record defines the network activity settings being used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = NTA
Requestor ID	4-19	16	Char	Requestor ID
Network ID	20-27	8	Char	Network ID
Network name	28-57	30	Char	Network descriptive name
Account number	58-89	32	Char	Network account number
User ID	90-121	32	Char	Network user ID
Direction	122-122	1	Char	Direction of the transmission
Charge code	123-123	1	Char	Network charge code
Measurement ID	124-127	4	Char	Type of statistic
Day	128-135	8	Char	Measurement date
Interchange envelopes	136-150	15	Char	Total number of interchange envelopes
Total characters	151-165	15	Char	Total number of characters sent
Filler	166-1024	859	Char	Filler for expansion

Transaction activity data extract

This record defines the transaction activity settings being used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = TPA
Trading Partner Nickname	4-19	16	Char	Trading partner ID in DataInterchange
Internal Trading Partner ID	20-54	35	Char	ID used for this trading partner internally (such as customer number, supplier number)
Company Name	55-94	40	Char	Company name of the trading partner
Address line 1	95-134	40	Char	First line of the company's address
Address line 2	135-174	40	Char	Second line of the company's address
Comment line 1	175-214	40	Char	Can be used to further classify the trading partner (customer, supplier, division, subsidiary)
Comment line 2	215-254	40	Char	Can be used to further classify the trading partner
Direction	255-255	1	Char	Direction of the transmission
Standard ID	256-263	8	Char	ID of the EDI standard
Version ID	264-265	2	Char	Version of the EDI standard
Release ID	266-267	2	Char	Release of the EDI standard
Description	268-317	50	Char	Description of the EDI standard/version/release
Transaction ID	318-325	8	Char	Transaction ID
TPT ID	326-341	16	Char	Trading partner transaction ID
Data format ID	342-357	16	Char	Name of the data format
Measurement ID	358-361	4	Char	Type of statistic
Measurement Date	362-369	8	Char	Date of this statistic
Total transactions	370-384	15	Char	Total transactions for the indicated date
Total errors	385-399	15	Char	Total transactions in error for the indicated date
Appl Trading Partner	400-415	16	Char	Application trading partner ID
Filler	416-1024	609	Char	Filler for expansion

Transaction Store data extract information categories

You can extract information from the DataInterchange Transaction Store databases using the ENVELOPE DATA EXTRACT and the TRANSACTION DATA EXTRACT commands. See "Producing management reports from the Transaction Store" on page 9 for a description of these PERFORM commands. The categories of information that may be requested are described in the table below.

This value:	Requests:	Using this keyword:
E	Interchange data	INTERCHANGE(Y)
G	Group data	GROUP(Y)
T	Transaction data	TRANSACTION(Y)
A	Application data	APPLICATION(Y)
R	Transaction image	IMAGE(Y)
E, G, T	Send acknowledgment data	SENDACKDATA(Y)
E, G, T	Receive acknowledgment data	RECEIVEACKDATA(Y)
F, K	Send acknowledgment image	SENDACKIMAGE(Y)
F, K	Receive acknowledgment image	RECEIVEACKIMAGE(Y)

All data extracted from the Transaction Store is written to the EDIQUERY file. The information categories are either written as separate records by setting **CONCATENATE** to **N**, or are combined and written as a single record by setting **CONCATENATE** to **Y**.

The following rules apply when requesting information categories:

1. Images (**R**, **F**, and **K**) are not produced unless you also request the transaction.
2. Images (**R**, **F**, and **K**) are always written as separate records even when concatenation has been requested.
3. Send acknowledgment data (**E**, **G**, and **T**) is not produced unless you also request the group record. If you want transaction acknowledgment data, you must also request the transaction record.
4. Send acknowledgment data (**E**, **G**, and **T**) is always concatenated to the corresponding group or transaction record even when concatenation has not been requested.
5. Receive acknowledgment data (**E**, **G**, and **T**) is not produced unless you also request the group or transaction record.
6. Receive acknowledgment data (**E**, **G**, and **T**) is always concatenated to the corresponding group or transaction record even when concatenation has not been requested.

Transaction Store data extract common key

All records created by Transaction Store Data Extract command have a common 141-byte key field that begins with a 3-character record ID. Portions of the key that do not apply to a particular record are initialized with blanks. The table below shows which fields are used for which record types.

Common key format

This record defines the common key format used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1	3	Char	Record ID = TX Used for record types: A, E, G, K, R, T
Nickname	4	16	Char	Trading partner nickname Used for record types: A, E, G, K, R, T
Direction	20	1	Char	Direction of the transaction Used for record types: A, E, G, K, R, T
Control Number	21	14	Char	Interchange control number Used for record types: A, E, G, K, R, T
Receiver ID	35	35	Char	Interchange receiver ID Used for record types: A, E, G, K, R, T
Control Number	70	14	Char	Group control number Used for record types: A, G, K, R, T
Control Number	84	14	Char	Transaction control number Used for record types: A, K, R, T
Controlling handle	98	20	Char	Handle value of controlling transaction (YYYYMMDDHHMMSSnnnnnn) Used for record types: A, K, R, T
Transaction handle	118	20	Char	Handle value of transaction (YYYYMMD- DDHHMMSSnnnnnn) Used for record types: A, K, R, T
Sequence Number	138	4	Char	Sequence number Used only for record type A

Transaction Store data extract record formats

The following tables show the formats for the records created by the TRANSACTION STORE DATA EXTRACT command. All tables, except images, are padded to a length of 1024 bytes to leave room for expansion. If concatenation is requested in the DataInterchange Utility control statements, then the full records as described are concatenated into a single record. Images are written separately and use the full logical record length of the EDIQUERY data set.

Interchange data extract record layout

This record defines the interchange settings being used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = E . E1 = Functional acknowledgment E2 = Transaction acknowledgment
Nickname	4-19	16	Char	Trading partner nickname.
Direction	20-20	1	Char	Direction of the interchange.
Control Number	21-34	14	Char	Interchange control number.
Receiver ID	35-69	35	Char	Interchange receiver ID.
Filler	70-141	72	Char	Blanks.
Sender ID	142-176	35	Char	Interchange sender ID.
Fake flag	177-177	1	Char	No interchange header.
Sequence error flag	178-178	1	Char	Interchange out of sequence (ENVELOPE DATA EXTRACT only).
Usage indicator	179-179	1	Char	Type of usage.
Duplicate interchange flag	180-180	1	Char	Duplicate interchange received.
Envelope date	181-194	14	Char	Date/time envelope created (YYYYMMDDHHMMSS).
Send date	195-208	14	Char	Date/time envelope sent (YYYYMMDDHHMMSS).
TA1 acknowledgment	209-209	1	Char	TA1 acknowledgment received.
TA1 date	210-223	14	Char	Date/time TA1 received (YYYYMMDDHHMMSS).
Network status code	224-225	2	Char	Network status.
Network status text	226-245	20	Char	Network status code in text format.
Acknowledgment expected	246-246	1	Char	Network acknowledgment expected.

Label	Position	Length	Type	Description
Acknowledgment received	247-247	1	Char	Network acknowledgment received.
Acknowledged date	248-261	14	Char	Date/time envelope of network acknowledgment (YYYYMMDDH-HMMSS).
Message user class	262-269	8	Char	Message user class assigned when sent.
Message name	270-277	8	Char	Message name assigned when sent.
Sequence number	278-282	5	Char	Sequence number assigned when sent.
Message ID	283-290	8	Char	Message ID assigned when sent.
Physical data set name	291-346	56	Char	Physical data set name to which data was queued.
Group count	347-357	11	Char	Number of groups in interchange.
Transaction count	358-368	11	Char	Number of transactions in interchange.
Segment count	369-379	11	Char	Number of segments in interchange.
Interchange size	380-390	11	Char	Number of bytes in interchange.
Interchange header	391-640	250	Char	Interchange header image.
Interchange trailer	641-670	30	Char	Interchange trailer image.
Filler	671-1024	354	Char	Reserved for expansion.

Group data extract record layout

This record defines the group settings being used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = G . G1 = Functional acknowledgment G2 = Transaction acknowledgment
Nickname	4-19	16	Char	Trading partner nickname.
Direction	20-20	1	Char	Direction of the group.
Control Number	21-34	14	Char	Interchange control number.
Receiver ID	35-69	35	Char	Interchange receiver ID.
Control Number	70-83	14	Char	Group control number.

Label	Position	Length	Type	Description
Filler	84-141	58	Char	Blanks.
Fake flag	142-142	1	Char	Interchange did not have groups.
Sender ID	143-177	35	Char	Application sender ID.
Receiver ID	178-212	35	Char	Application receiver ID.
Acknowledgment expected	213-213	1	Char	Functional acknowledgment expected.
Acknowledgment received	214-214	1	Char	Functional acknowledgment received. Note: If an acknowledgment is not expected, then this field is blank.
Acknowledgment date	215-228	14	Char	Date/time group acknowledgment received (YYYYMMDDH-HMMSS).
Acknowledgment handle	229-248	20	Char	Handle value for the functional acknowledgment transaction (YYYYMMDDHHMMSSnnnnnn).
Transaction count	249-259	11	Char	Number of transactions in group.
Segment count	260-270	11	Char	Number of segments in group.
Group size	271-281	11	Char	Number of bytes in group.
Group header	282-434	153	Char	Image of the group header.
Group trailer	435-460	26	Char	Image of the group trailer.
Filler	461-1024	564	Char	Filler for expansion.

Transaction data extract record layout

This record defines the transaction settings being used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = T . T1 = Functional acknowledgment T2 = Transaction acknowledgment
Nickname	4-19	16	Char	Trading partner nickname.
Direction	20-20	1	Char	Direction.
Control Number	21-34	14	Char	Interchange control number.
Receiver ID	35-69	35	Char	Interchange receiver ID.
Control Number	70-83	14	Char	Group control number.
Control Number	84-97	14	Char	Transaction control number.

Label	Position	Length	Type	Description
Controlling handle	98-117	20	Char	Handle value of the controlling transaction (YYYYMMDDH-HMMSSnnnnnn).
Transaction handle	118-137	20	Char	Handle value of the transaction (YYYYMMDDHHMMSSnnnnnn).
Filler	138-141	4	Char	Blanks.
Enveloped date	142-155	14	Char	Date/time transaction was enveloped (YYYYMMDDHHMMSS).
Creation date	156-169	14	Char	Date/time transaction put into the store.
Transaction status code	170-171	2	Char	Current transaction status.
Transaction status text	172-191	20	Char	Transaction status in a text format.
Acknowledgment received	192-192	1	Char	Group acknowledgment received.
Acknowledgment received text	193-212	20	Char	Group acknowledgment received in a text format.
Acknowledgment date	213-226	14	Char	Date/time group acknowledgment was received (YYYYMMDDH-HMMSS).
Trx Acknowledgment expected	227-227	1	Char	Transaction acknowledgment expected.
Trx Acknowledgment received	228-228	1	Char	Transaction acknowledgment received.
Trx Acknowledgment received text	229-248	20	Char	Transaction acknowledgment received in a text format.
Trx Acknowledgment date	249-262	14	Char	Date/time transaction acknowledgment was received (YYYYMMD-DHHMMSS).
Acknowledgment handle	263-282	20	Char	Handle value of the transaction acknowledgment transaction (YYYYMMDDHHMMSSnnnnnn).
Segment count	283-293	11	Char	Number of segments in transaction.
Transaction size	294-304	11	Char	Number of bytes in transaction.
Enveloped segment count	305-315	11	Char	Number of segments in transaction when enveloped.
Enveloped transaction size	316-326	11	Char	Number of bytes in transaction when enveloped.

Label	Position	Length	Type	Description
Format ID	327-342	16	Char	Last data format ID.
Transaction ID	343-350	8	Char	Standard transaction.
Group ID	351-356	6	Char	Function Group ID associated with the transaction.
Envelope member	357-364	8	Char	Member name used for enveloping.
Envelope type	365-365	1	Char	Type of envelope associated with transaction.
Network ID	366-373	8	Char	Network associated with trading partner.
Standard ID	374-381	8	Char	EDI standard ID of the transaction.
Standard Version	382-383	2	Char	EDI standard version.
Standard Level	384-385	2	Char	EDI standard release level.
Internal Trading Partner ID	386-420	35	Char	Last internal trading partner ID.
Application control field	421-455	35	Char	Last application control field.
Delivery date	456-469	14	Char	Last date given/gotten from application (YYYYMMDDHHMMSS).
Earliest envelope date	470-477	8	Char	Earliest date that transaction will be enveloped (YYYYMMDD).
Earliest purge date	478-485	8	Char	Earliest date that transaction will be automatically purged (YYYYMMDD).
Error level	486-486	1	Char	Translation error level.
Store status	487-488	2	Char	Store status of the transaction.
Store status text	489-508	20	Char	Store status in a text format.
Override flag	509-509	1	Char	Envelope overrides provided for transaction.
Held flag	510-510	1	Char	Transaction held.
Test flag	511-511	1	Char	Test transaction.
Duplicate flag	512-512	1	Char	Transaction part of duplicate envelope.
Purge flag	513-513	1	Char	Purge transaction.
Translate flag	514-514	1	Char	Translated.
Detached flag	515-515	1	Char	Transaction detached.
Override handle	516-535	20	Char	Handle for enveloping overrides (YYYYMMDDHHMMSSnnnnnn).

Label	Position	Length	Type	Description
Network security profile name	536-543	8	Char	Network security profile name for group level encryption.
Encryption key	544-559	16	Char	Encryption key name for group level encryption.
Authentication key	560-575	16	Char	Authentication key name for group level encryption.
Application assigned control number	576-589	14	Char	Transaction control number if assigned by the application.
Data element delimiter	590-590	1	Char	Data element delimiter used when transaction was translated.
Sub element-delimiter	591-591	1	Char	Subelement delimiter used when transaction was translated.
Segment terminator	592-592	1	Char	Segment terminator used when transaction was translated.
Decimal notation	593-593	1	Char	Decimal notation used when transaction was translated.
Release character	594-594	1	Char	Release character used when transaction was translated.
Segment ID separator	595-595	1	Char	Segment ID separator used when transaction was translated.
Transaction header	596-680	85	Char	Image of the transaction header.
Transaction trailer	681-706	26	Char	Image of the transaction trailer.
Last ID	707-722	16	Char	Last used.
Filler	723-1024	302	Char	Reserved for expansion.

Application data extract record layout

This record defines the application settings being used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID = A
Nickname	4-19	16	Char	Trading partner nickname
Direction	20-20	1	Char	Direction of format
Control Number	21-34	14	Char	Interchange control number
Receiver ID	35-69	35	Char	Interchange receiver ID
Control Number	70-83	14	Char	Group control number

Label	Position	Length	Type	Description
Control Number	84-97	14	Char	Transaction control number
Controlling handle	98-117	20	Char	Handle value of controlling transaction (YYYYMMDDH-HMMSSnnnnnn)
Transaction handle	118-137	20	Char	Handle value of transaction (YYYYM-MDDHHMMSSnnnnnn)
Sequence Number	138-141	4	Char	Sequence number
Format ID	142-157	16	Char	Data format ID
Application ID	158-165	8	Char	Application ID
BATCH ID	166-173	8	Char	Batch ID
Delivery Date	174-187	14	Char	Date/time delivered to application (YYYYMMDDHHMMSS)
Error level	188-188	1	Char	Translation error level
Acceptable error level	189-189	1	Char	Acceptable error level
Error count	190-200	11	Char	Number of errors found
Filler	201-1024	824	Char	Reserved for expansion

Transaction/Acknowledgment image data extract record layout

This record defines the transaction and acknowledgment image settings being used to extract data. The fields are described in the table below.

Label	Position	Length	Type	Description
Record ID	1-3	3	Char	Record ID. Valid values are: RX = Transaction image continued RZ = Final transaction image record FX = Functional acknowledgment image continued FZ = Final functional acknowledgment image record KX = Transaction acknowledgment image continued KZ = Final transaction acknowledgment image record
Nickname	4-19	16	Char	Trading partner nickname.
Direction	20-20	1	Char	Direction of image data.
Control Number	21-34	14	Char	Interchange control number.
Receiver ID	35-69	35	Char	Interchange receiver ID.
Control Number	70-83	14	Char	Group control number.

Label	Position	Length	Type	Description
Control Number	84-97	14	Char	Transaction control number.
Controlling handle	98-117	20	Char	Handle value of controlling transaction (YYYYMMDDH-HMMSSnnnnnn).
Transaction handle	118-137	20	Char	Handle value of transaction (YYYYM-MDDHHMMSSnnnnnn).
Filler	138-141	4	Char	Blanks.
Total size	142-152	11	Char	Total size of the image.
Record size	153-163	11	Char	Size of image data in this record.
Image	164-end	Variable	Char	Transaction or acknowledgment image.

Using the DataInterchange application program interface

You can request DataInterchange services directly from an application program. There are application program interfaces (APIs) defined for the following services:

- Environmental services
- Translation services
- Enveloping services
- Communication services
- Data extraction services
- Update status services
- SYNCPOINT services

The API for each of these services is described in detail later in this chapter. The information common to all the APIs is described next.

The following control blocks are common to all service requests:

- Common control block (CCB)
DataInterchange 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 should 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 581.
- Service name block (SNB)
The service being requested. As 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 should define an SNB for each service requested. For a complete description of the SNB, see “Service Name Block (SNB)” on page 579.

- **Function control block (FCB)**
The particular service function being requested. As 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 584.

API languages

You can access DataInterchange services with a simple call statement to a DataInterchange-provided *stub* program. DataInterchange provides four stub programs, one for each application programming language directly supported by DataInterchange. These stub programs and associated languages are:

- FXXZCBL for COBOL programs
- FXXZPLI for PL/I programs
- FXXZC for IBM C/370 programs (MVS only)
- FXXZASM for Assembler programs

Although DataInterchange 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 DataInterchange services by using the FXXZASM stub program.

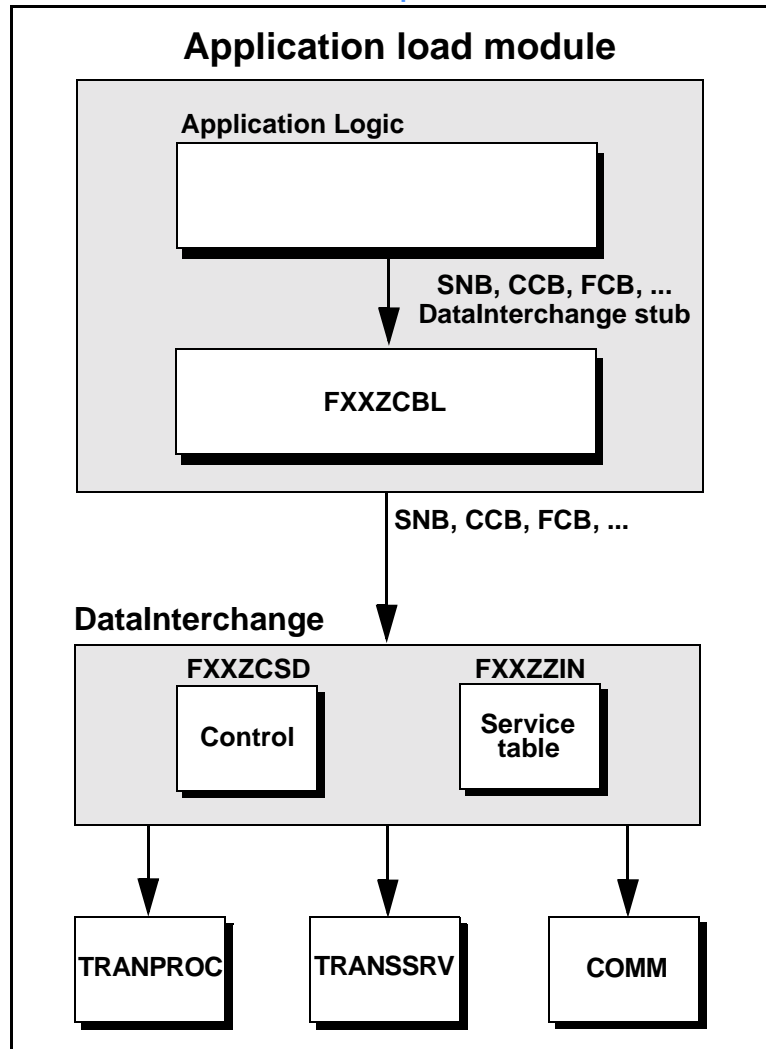
API link edit

The load library distributed with DataInterchange contains a load module for each stub program. These stub programs are linked with the application program requesting the services. DataInterchange is not physically part of the application load module. You must use the stub programs to access DataInterchange.

When linking an application program, the DataInterchange distribution load library should 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 DataInterchange 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 DataInterchange services. The application program can be above or below the 16 MB line, and all parameters passed to DataInterchange 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 DataInterchange is illustrated in the diagram on page 293.

Figure 1-1. Load Module Relationships



DataInterchange/MVS and DB2 attachment

For the most part, the DB2 processing mode that DataInterchange 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 DataInterchange does not attempt to attach or detach DB2 (an attachment is assumed to have taken place prior to DataInterchange initialization).
- If EDITSIN exists, **OPEN** is not **N**, **CAF** is not **Y**, and DataInterchange detects that DB2 is already attached while trying to make the DB2 attachment, then the DB2 processing mode becomes **DSN**. This exception is true of the DataInterchange facility CLIST.
- If **OPEN** is not **N** and values are supplied for **SYSTEM** and **PLAN**, DataInterchange attempts to attach DB2.
- If DataInterchange 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**, DataInterchange termination will detach DB2.

The table below describes the keywords used in EDITSIN.

Keyword	Description
SYSTEM	The name of the DB2 subsystem (or group, if data sharing). This value may be up to four characters long and is required if EDITSIN exists.
PLAN	The DB2 plan name. This value may be up to eight characters long and is required if EDITSIN exists.
OPEN	Indicates whether DataInterchange should attempt to attach to DB2 during initialization. The default is Y .
CLOSE	Indicates whether DataInterchange should detach from DB2 during DataInterchange termination. The default is Y .
CAF	If DataInterchange initialization detects while trying to make a DB2 attachment that DB2 is already attached, the DataInterchange DB2 processing mode becomes DSN . A value of Y for this keyword acts as an override and tells DataInterchange to keep the DB2 processing mode CAF . The default is N .

EDITSIN examples

Example 1: Your DB2 subsystem is named DB93, your DB2 plan is named **EDIENU41**, and you want DataInterchange to handle the DB2 attachment and detachment. Include the following keywords and values in EDITSIN. The **OPEN** and **CLOSE** keywords are not needed since the default action is for DataInterchange to handle DB2 attachment.

```
SYSTEM(DB93) PLAN(EDIENU41)
```


Example 2: Your DB2 subsystem is named DB93, your DB2 plan is named EDIENU41, and you do not want DataInterchange to attach or detach DB2. Include the following keywords and values in EDITSIN.

```
SYSTEM(DB93) PLAN(EDIENU41) OPEN(N) CLOSE(N)
```

Example 3: Your DB2 subsystem is named DB93, your DB2 plan is named EDIENU3, DB2 may be attached, but, regardless, you do not want DataInterchange to detach from DB2 on termination, and you want DataInterchange 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(EDIENU41) CLOSE(N) CAF(Y)
```

DataInterchange/MVS- CICS abend return codes

When an abend is detected, DataInterchange 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 DataInterchange 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]
```

DataInterchange assumes that all parameters are pointers to control blocks. COBOL passes control block pointers as part of normal processing.

Code fragments for initializing DataInterchange 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 DataInterchange 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)
```

DataInterchange 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 DataInterchange 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 DataInterchange 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),       /* MVS DB2 subsystem ID    */
    3 ZCCBDBPL        CHAR(8),       /* MVS 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 MVS environment. Other environments are not supported. When using C, you invoke DataInterchange as a normal function call. The call is made to FXXZC and is coded as follows.

```
FXXZC(&MYSNB,&MYCCB,&MYFCB,other parms)
```

DataInterchange 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 DataInterchange 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     zccbldpid[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];      /* MVS DB2 subsystem ID     */
    char     zccbdbpl[8];      /* MVS 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      zfcbl1;           /* 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 DataInterchange 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:

Register	Description
R0	Not defined
R1	Address of a parameter list
R2 - R12	Not defined
R13	Address of 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
              etc.  for as many parms as the service calls for
```

Code fragments for initializing DataInterchange 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
ZCCBRVS      DS      F           Reserved for future use
ZCCBCCXP     DS      F           CCB extension pointer
ZCCBCABP     DS      F           Common area block pointer
ZCCBDBID     DS      CL4        MVS DB2 subsystem ID
ZCCBDBPL     DS      CL8        MVS DB2 plan / AIX alias
ZCCBDBUI     DS      CL8        AIX DB2 user ID
ZCCBDBPW     DS      CL18       AIX DB2 password
ZCCBRVS1     DS      CL26       DI internal use only
ZCCBRVS2     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
```

```

*****
*   DataInterchange initialization successful   *
*****

INITERR EQU*

*****
*   DataInterchange 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 then sends the data to all trading partners, the following sequence of API service calls is required:

1. Call the environmental service to initialize DataInterchange. For detailed information, see “Initializing the environmental API” on page 304.
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 311.
3. Call the translation service to signal the end of translation. For detailed information, see “End translation/enveloping API” on page 384.
4. Call the communication service to send the transactions. For detailed information, see “Send transactions and restart send transactions API” on page 412.
5. Call the environmental service to terminate DataInterchange. For detailed information, see “Terminating the API” on page 306.

You can accomplish all of these calls using the TRANSLATE AND SEND command. The DataInterchange utility issues the same API requests that you would issue in an application program.

You should base the decision on whether to use the API or a DataInterchange Utility request (PERFORM command) on the control and availability of data rather than availability of DataInterchange Utility functions. The DataInterchange Utility provides most of the functions available in the API. However, the DataInterchange 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 MVS). 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 DataInterchange 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 should consider writing a program to request API services directly.

- You might want to synchronize the updating of your data with the results of DataInterchange 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 DataInterchange databases to verify that the business transaction has been processed.



NOTE: In CICS, you can synchronize application data with DataInterchange data without writing an API program. For more information, see Chapter 6, “Using DataInterchange in the CICS environment”.

- The application data might not meet the RAWDATA requirements of DataInterchange or might be formatted in C and D records. As a result, your data cannot be processed by DataInterchange. When you understand the data, you can instruct DataInterchange how to perform translation.
- Your application data might not meet the access type requirements of DataInterchange (QSAM in MVS, 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 DataInterchange utilities can be used.
- The sequence of transactions in interchanges and groups that is created by the DataInterchange 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 DataInterchange Utility. For more information, see Chapter 1, “Using The Datainterchange Utility”.

You might need a combination of application-written API programs and DataInterchange 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 DataInterchange
 - b. Translation service to translate data without enveloping
 - c. Translation service to signal the end of translation
 - d. Environmental service to terminate DataInterchange
2. A DataInterchange Utility step to use the ENVELOPE AND SEND command, or a two-step DataInterchange Utility process to use the ENVELOPE command followed by the SEND command.

Environmental services

Environmental services both establishes and removes the DataInterchange environment.

First, your application program should request initialization. Once 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 should 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 DataInterchange can release all resources that it has obtained.

The table below lists the functions provided by the environmental service.

Function	Code	Sample Call Statement for Function
Initialize DataInterchange	1	<code>FXXZccc(SNB,CCB,FCB,'applid','sysid')</code>
Terminate DataInterchange	2	<code>FXXZccc(SNB,CCB,FCB)</code>

Initializing the environmental API

You must request the initialization function to enable your program to use the DataInterchange environment. During request processing, DataInterchange confirms that:

- All necessary resources for a DataInterchange environment are available, and
- You are authorized to access the system (identified by the SYSID parameter)

DataInterchange 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 should identify a member in the application definition profile (APPDEFS) or in the activity log profile (ACTLOGS). If an APPDEFS member does not exist, the following assumptions are made:

- The **APPLID** parameter identifies an ACTLOGS member. If it does not, and logging is necessary for other service requests, errors occur.
- The management reporting functions must be enabled for the current DataInterchange session.

The syntax for the Initialization API request is.

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

The same CCB used to initialize the DataInterchange session is used for every call issued during the session. The unique parameters for this function request are:

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 DataInterchange session.
FCB	ZFCBFUNC 1
APPLID	The application ID. Must match either a member in the application definition profile or in the activity log profile. 8 bytes, left-justified, and padded with blanks.
SYSID	The system ID. Must match a resource name the security administrator assigned to DataInterchange. 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 should 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:

RC	ERC	Explanation
0		Initialization was successful.
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 may be caused by an incorrect ZCCBLPID value (one that cannot be matched with a LANGPROF member). In DB2 installations, this error may 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 DataInterchange DB2 load modules: EDIRPML and (for CICS users) EDICRIN, which could generate DB2 timestamp problems if out of sync.
	16	A DataInterchange 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:

Parameter	Description
SNB	ZSNBNAME UTILSRV ZSNBPC 4
CCB	The common control block that was used to initialize DataInterchange.
FCB	ZFCBFUNC 1 - Normal execution 2 - HOT DataInterchange mode
UTILCB	The utility service control block.

Terminating the API

You must request the termination function for DataInterchange 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, DataInterchange returns a value other than zero in the return code field (ZCCBRC). If this occurs, your application program should 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 DataInterchange component for which the error occurred attempts to log the error.

After the error occurs, DataInterchange returns control to your program so that your program can request termination again to complete the task. DataInterchange 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:

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:

RC	Explanation
0	Termination was successful.
>0	Termination failed; try again.
<0	An invalid CCB address, or an incorrect name in the ZSNBNAME field.

Translation services

Translation services, enveloping services, and data extraction services are closely related. They share an API implemented by the same logical service (TRANPROC). These services also share the Transaction Store.

Exchanging information with a trading partner is a sequence of interruptible events whose status must be maintained and tracked. The processes used to exchange data fall into two categories:

- A batch-oriented process (MVS)
- 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 may occur, the batches are typically much smaller than in the batch-oriented process. The trading partner is expected to process the data and generate a response immediately.

The Transaction Store database saves all EDI standard data sent or received and tracks the status of the data as it progresses from translated, to send, to received, to acknowledged. The Transaction Store is updated by all translation and enveloping operations, and is maintained and reported on by the Transaction Store Facility or the DataInterchange Utility. Updating the Transaction Store database is optional and may be controlled using the APPDEFS profile. For more information about the APPDEFS profile, refer to the *DataInterchange Administrator's Guide*.

The Transaction Store database consists of seven DB2 tables as described below.

Table	Description						
EDIVTSTH	The transaction handle table. This is the primary table in the Transaction Store and is referenced by all other tables. An entry is created whenever a translation occurs and a transaction is deenveloped. As transactions are added to the Transaction Store, they are assigned a unique key value called a transaction handle (THANDLE). The format of the transaction handle is: <i>YYYYMMDDH-HMMSSxxxxnnnn</i> , where: <table> <tr> <td><i>YYYYMMDD</i></td><td>The date of the first transaction for the translation session.</td></tr> <tr> <td><i>HHMMSS</i></td><td>The time of the first transaction for the translation session.</td></tr> <tr> <td><i>xx</i></td><td>A random number assigned by DataInterchange. Valid values are 00 - 99.</td></tr> </table>	<i>YYYYMMDD</i>	The date of the first transaction for the translation session.	<i>HHMMSS</i>	The time of the first transaction for the translation session.	<i>xx</i>	A random number assigned by DataInterchange. Valid values are 00 - 99 .
<i>YYYYMMDD</i>	The date of the first transaction for the translation session.						
<i>HHMMSS</i>	The time of the first transaction for the translation session.						
<i>xx</i>	A random number assigned by DataInterchange. Valid values are 00 - 99 .						

Table	Description
<i>nnnn</i>	A sequential number assigned by DataInterchange. Valid values are 0 - 9999 . If more than 9999 transactions are translated in the same session, DataInterchange 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 DataInterchange 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 develope takes place.
EDIVTSTO	The transaction override table. Contains override values for fields in the service segments. A transaction override table is created only when translation occurs and overrides are supplied for service segment fields. These values are captured at translation time and used when the transaction is enveloped.
EDIVTSEV	Contains information about an entire interchange, such as the network status, the date and time created, and the date and time sent. A table entry is created during enveloping or deenveloping for each interchange header created or deenveloped.
EDIVTSGP	Contains information about a group of transactions in an interchange, such as the functional acknowledgment status of the group. A table entry is created during the enveloping or deenveloping process. At least one table entry is created, even when the interchange does not contain a functional group.
EDIVTSTU	Contains information about a transaction in an interchange or group, such as the date and time the transaction was enveloped, and the acknowledgment status of the transaction. A table entry is created during the enveloping or deenveloping process. A transaction that has been enveloped more than once (REENVELOPED) will have entries in the EDIVTSTU, EDIVTSGP and EDIVTSEV tables.
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 table below lists the functions provided by the translation service. The logical name for the translation service is TRANPROC.

Function	Code
Test translate-to-EDI-standard	111
Production translate-to-EDI-standard	131
Test deenvelope and translate-to-application	211
Production deenvelope and translate-to-application	212
Translate a specific transaction to application	213
End translation	1000

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 311 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 350 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 384 for more information.
- **Retrieve interchange header**
Retrieves the EDI standard image of the current interchange, and is used by the DataInterchange Utility to create the optional **E** record. See “Retrieve interchange header API” on page 399 for more information.
- **Retrieve group header**
Retrieves the EDI standard image of the current group, and is used by the DataInterchange Utility to create the optional **G** record. See “Retrieve group header API” on page 400 for more information.
- **Retrieve transaction**
Retrieves the EDI standard image of the current transaction, and is used by the DataInterchange Utility to create the optional **T** record. See “Retrieve transaction header API” on page 400 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 400 for more information.



NOTE: The transform function for data transformation maps is not currently supported using the DataInterchange 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	TF	Translate file
EV	Envelope	TS	Translate to standard
TA	Translate to application		

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 DataInterchange 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 table below contains an overview of the functions provided by the enveloping service. The logical name for the enveloping service is TRANPROC. This service is described in “Environmental services” on page 304.

The syntax for the enveloping and sending service is:

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

Function	Code
Retrieve interchange header	1
Retrieve group header	2
Retrieve transaction header	3
Deenvelope transaction	214
Envelope transaction	215
Close and QUEUE the current interchange	990
Issue database COMMIT	991

The following three API functions allow you to translate, envelope, and send a file to a trading partner:

- The translation service (function code **131**) translates the data and updates the Transaction Store database with the transaction information and an image of the EDI standard data produced. For more information, see “Translate-to-standard API” on page 311.
- The enveloping service (function code **215**) builds the proper service segments and retrieves the transaction image from the Transaction Store. It invokes the communication service with function code **110** which writes the interchange to a file associated with the network. For more information, see “Envelope API” on page 368.
- The communication service (function code **211**) builds the proper commands for the network, and then invokes the network to send the file to your trading partner. For more information, see “Send transactions and restart send transactions API” on page 412.

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 diagram” on page 314.

You can control whether enveloping is done at the time of translation with the **ENVLDELAY** field in the TCB. You can only delay enveloping for translated transactions already stored in the Transaction Store database. For more information, refer to the APPDEFS profile description in the *DataInterchange Administrator's Guide*. For more information about the **ENVLDELAY** field, see “Translator Control Block (TRCB)” on page 586.

Consider the following points when performing automatic enveloping:

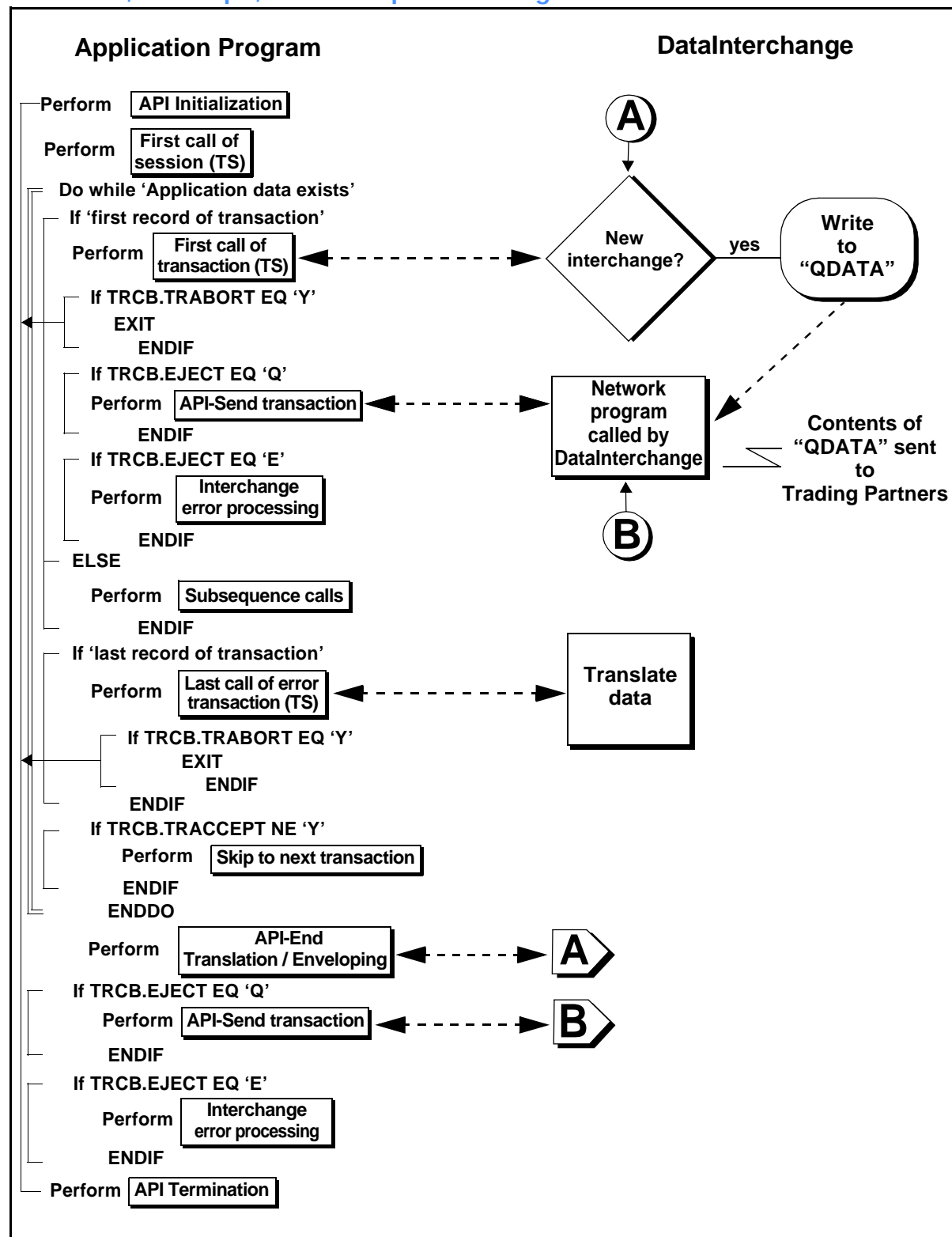
- All data involved in a translation, including the image of the EDI standard data produced, is saved in the Transaction Store. This occurs whether enveloping is part of the translation or is delayed. An EDI standard image is saved even when the translation is not successful, because the image might prove helpful in determining why the translation failed.
- If enveloping is done separately from translation, the enveloping service must be notified which transaction to envelope. This is done through the unique key value assigned to the transaction when it is added to the Transaction Store. The key is stored in the database as a 10-byte packed value and is returned in packed format in the TSKEY field of the translator control block. It is also returned as a 20-byte character value in the TSKEYU field.

When an envelope function is requested, either the TSKEY or TSKEYU value must be accessible to the enveloping service. The program that issues the translate-to-standard function can save the TSKEY values in a file which the program that issues the envelope calls can read to get the TSKEY values. To organize the data the way you want it enveloped, store the TSKEY values in a file with other data that can be sorted.

If only the TSKEY values are necessary, you can use the DataInterchange 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 DataInterchange 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 diagram



Test translate-to-standard

There are two methods for testing your transactions:

1. You can use test function code **111** to translate your transactions to EDI standard format in test mode. Your program follows the same steps as in production mode. Translation occurs and the interchange created is written to the file associated with the network, but no information is posted to the Transaction Store. To post information to the Transaction Store, you must use the method described below. You can use the results of the translation for receiving transactions in test mode.



NOTE: This form of testing is available only through the API.

When you use this method, it is assumed that:

- You require a test usage/rule, if one is available, and the **TEST** field in the TRCB is set to **T**
- You want enveloping to occur so that an EDI standard transaction image will be created and written to a file that you can inspect

If your test translation is followed by a call to send the data, make sure to define a destination file that is in no danger of being sent accidentally. The EDI standard data produced is written to ddname EDITEST. Communications ignores any request to send data when the ddname of the file to be sent is EDITEST. In the CICS environment, EDITEST is a TS queue.

You can verify the results of your program's efforts by examining the resulting data and the information recorded in the event log. To view or print the event log, choose **EVENT LOGGING** from the Administrator's Menu.

2. You can use the test usage indicator to inform trading partners that a transaction you are sending is for test purposes only. When you set the **Test usage** field on the Send Usage panel to **T**, translation and communications occur normally (including updating the Transaction Store) except that the envelope carries an indicator that the data is for testing. You may 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 DataInterchange 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. DataInterchange 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 should 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 may 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 should 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 may 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 331.

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 DataInterchange initialization or after a translation service termination. This call establishes the default values that apply from the first call until a translator termination call is received.
- The first call to the translator for a transaction. Key information concerning the transaction is provided, and the translator determines the relationship that this transaction has with the previous transaction.
- The final call to the translator for a transaction. Translation from the application format to the EDI standard format actually takes place.
- The final call to the translator to terminate the session. Cleanup is done and final processing occurs.

All other calls merely involve the movement of data from the application buffers into the translator buffers. The following topics describe these four critical calls in detail.

Translate-to-standard API

You can use the two following function codes to request a translate-to-standard function:

- 111** Requests translation in test mode
- 131** Requests translation in production mode

You do not need to change your program to switch from test mode to production mode. You only need to change the function code used to invoke the services. Test mode is different from production mode in the following ways:

- DataInterchange 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.
- DataInterchange forces the **ENVLDELAY** field in the TRCB to contain a value of **N**. This writes the EDI standard data produced to a file.
- DataInterchange 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, DataInterchange ignores the request.
- Control numbers are not taken from the trading partner profile member. Instead, they are assigned values of **Tnnnnn**, where **nnnnn** is a sequential value starting with 00001.
- DataInterchange automatically logs the EDI standard image that is produced and the application data that is received.
- DataInterchange does not write any data to the Transaction Store, to prevent cluttering up the database with test data.
- Functional acknowledgments are not generated even if requested in the usage/rule record.
- Statistics maintained by the Management Reporting component of DataInterchange 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:

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

Parameter	Description
TRCB	The translator control block. For more information about this block, see “Translator Control Block (TRCB)” on page 586.
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. DataInterchange uses the output data block as a work buffer. Minimum size is 32000 bytes. Maximum size should 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 should 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 the following tables.

SNB initialization for translate-to-standard



Field name	Initialization
ZSNBLL	32
ZSNBNAME	TRANPROC
ZSNBPC	6 (all calls for translation services have six parameters)





FCB initialization for translate-to-standard

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

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
XPANDED	Y. Indicates that DataInterchange should check the BLKLEN field to determine the software version and release being used.

Field name	Initialization
ENVLDELAY	<p>Indicates whether enveloping should be delayed until translation is complete.</p> <p>Y Enveloping does not occur at the same time as translation.</p> <p>(other) Enveloping occurs at the same time as translation.</p>
SCOPE	<p>Indicates that the point at which DataInterchange issues a database COMMIT.</p> <p>E Interchange (envelope) level recovery. Applies only when enveloping occurs.</p> <p>(other) Transaction level recovery.</p> <p> NOTE: The value set in this field affects the recovery scope used during the session. For more information, see “Send Recovery Scope:” on page 334.</p>
INMEMTRANS	<p>Indicates the maximum number of transactions that should 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.</p> <p>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 606.</p> <p> NOTE: The value set in this field affects the recovery scope used during the session. For more information, see “Send Recovery Scope:” on page 334.</p>
FILEID	<p>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.</p>
ITPBREAK	<p>Indicates whether a change in the internal trading partner ID (vendor number) starts a new interchange.</p> <p>You can use this field to create separate interchanges for the individual vendors associated with a trading partner.</p> <p>Y Starts a new interchange each time the internal trading partner ID changes. The current interchange should be closed and written.</p> <p>N or (other) Starts a new interchange only when the trading partner nickname changes. N is the recommended value.</p>

Field name	Initialization
BATCHID	<p>The batch ID for a group of transactions. The Transaction Store creates an alternate key value using the batch ID. Other than using the transaction handle, searching on batch ID is the quickest way to retrieve a transaction from the Transaction Store during the selection process. If you do not specify a value for this field, the default value <i>DDHHMMSS</i> is used, where <i>DD</i> is the current day of the month, and <i>HHMMSS</i> is the current time. For more information, see “Batches and Bundles:” on page 335.</p> <p> 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.</p>
TRXLIFE	<p>The amount of time a transaction remains in the Transaction Store before it is eligible for purging. Default is 30 days.</p> <p> 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.</p>
IMGLIFE	<p>The amount of time a transaction's image (the standard data produced) remains in the Transaction Store. Default is 30 days.</p> <p> NOTE: This field is not currently supported.</p>
ENVLDATE	<p>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.</p> <p> 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.</p>
ERRFILTER	<p>Indicates which error codes to filter out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more information, see “Error filtering” on page 21.</p>
RAWDATAOUT	<p>Indicates whether raw data output is desired for fixed-to-fixed mappings.</p> <p>Y Uses raw data output for fixed-to-fixed mappings</p> <p>(other) Uses C and D record output for fixed-to-fixed mappings</p>
FFILEID	<p>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.</p>

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:

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: <table> <tr> <td>I</td><td>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.</td></tr> <tr> <td>P</td><td>This is a production transaction. Use only a production usage/rule (default).</td></tr> <tr> <td>T</td><td>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.</td></tr> <tr> <td>U</td><td>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 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.</td></tr> </table>	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 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 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.
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 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 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.								

DataInterchange requires these fields to determine what translation is being requested. The TRCB fields described below must be set by the application 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
RAWDATA	Indicates whether the RAWDATA interface should be used. For more information on the RAWDATA interface, see "Send raw data" on page 331. 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 334. Y All application data has been provided and translation should begin. For more information on this field, see "Translate-to-standard API" on page 311. For more information on the effects of the EJECT field settings, see "Last call for transaction (TS)" on page 327.
ATSID	The name of the structure for which data is being provided in the TRIDB.
BNDLFLAG	Indicates whether a bundle should be started or ended. The first transaction of a cluster is called the controlling transaction and is the transaction displayed when you use the Transaction Store Facility. DataInterchange 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 335. 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.

Field name	Initialization
HOLDFLAG	Indicates whether this transaction is to be placed on hold when added to the Transaction Store. A transaction in hold status is not available for any other activity, such as enveloping. You must release a held transaction before it can be processed. Y Places this transaction in Held status N or (other) Does not place this transaction in Held status
ROUTCODE	A three-character generic routing code provided by the application and used by DataInterchange 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 DataInterchange can use them when the transaction is enveloped.

You can use the TRCB fields described below 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 367.

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 .

Field name	Overrides:
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 .
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 .
GRESAGENCY	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 .

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 should match the number of bytes defined for the structure in the data format.
DATA	The application data. The format of the data must match the format of the structure (in the ATSID field) defined in the data format.

When initialization is complete, the translator is invoked with the following API request:

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

When the translator receives this request, it verifies that everything it needs to translate the transaction is available. The **ATFID**, **INTPID**, and **TEST** fields are used to locate the trading partner usages/rules and map, and retrieve the control string generated from the map along with the EDI standard and transaction descriptions.

Numerous errors can occur at this point. See “Translation return codes” on page 680 for a description of errors generated by the translator. 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 described below indicate the status of the transaction and of the translator.


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 described below are returned on the first call for a transaction and that identify the key attributes of the 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 should 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 380 for a description of the fields that cause a new interchange to be started.

The TRCB fields described below are returned when an interchange is written. The fields provide information about the interchange, the network, and the file receiving the interchange.

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 DataInterchange 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.

Field name	Description
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 380 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 412 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 should 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)” on page 327.

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. For detailed information, see “Translation return codes” on page 680. 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 should 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 below are updated during translation, based on translation type.

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.
TSKEY	The transaction handle for this transaction in format <i>YYYYMMDDH-HMMSSxxxxnnnn</i> . 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 614.

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

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 399.
Y	Started a new interchange
(other)	Did not start a new interchange

Field name	Description
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.
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 & 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 400.
Y	Started a new group
(other)	Did not start a new group

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. 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).
GTT	The character representation of TRNGRP .

TRCB fields 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 400. Y Started a new transaction (other) Did not start a new transaction
THCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros. Applies only to Version 2.1 and earlier.
THXCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros. Applies only to Version 3.1 and later.
SEGTRN	The total number of segments in the current transaction stored as a 4-byte binary value. See TST for the character representation of this value.
TTC	The current transaction or message ID value. See also TRNID on page 345.

Field name	Description
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, DataInterchange 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 DataInterchange Utility, your application may 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 DataInterchange 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 transaction. 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 following TRCB fields are updated:

Field	Description
ATSID	The name of the structure that was received, based on the raw data specifications.
INTPID	<p>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.</p> <p>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.</p> <p>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, then you must call the translator again with an EJECT value of F. This causes the previously accepted structures for this transaction to be discarded.</p> <p>If a internal trading partner ID is not defined in the data format, or if this field contains all blanks, then the value of INTPID from the first call for the transaction is used as the default.</p>
TRNSTAT	<p>Indicates the status of the transaction and, therefore, determines what your program should do next. Valid values are:</p> <p>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.</p> <p>S The data that you supplied was recognized and has started a transaction.</p> <p>C The data that you supplied was identified and continued the transaction.</p> <p>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 then call the translator again with the same data.</p> <p>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.</p>

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** should 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:: DataInterchange allows you to specify how much processing should take place before DataInterchange 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 should 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 586 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. DataInterchange 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. DataInterchange 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 should 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 DataInterchange, 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 should 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 399.



NOTE: Setting the **NOCOMIT** field to **Y** prevents the translator from issuing commits, but it does not prevent DataInterchange termination from issuing a COMMIT. Making a SYNCPOINT service call with the syncpoint interval set to **-1** prior to the DataInterchange termination call will prevent this termination commit. For more information, see “SYNCPOINT services” on page 436.

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 DataInterchange with the Transaction Store files defined as recoverable.

Forcing Interchange Termination:: If enveloping occurs at the same time as translation, the translator attempts to build the largest interchange possible. The current interchange is completed and a new one started only when certain field values change. For more information and a complete list of items that cause a new interchange, see “Fields that cause a new interchange to start” on page 380.

You cannot control the size or content of an interchange using the DataInterchange 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 382.

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 DataInterchange 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 DataInterchange Utility or facility.

Batches and Bundles: You can group transactions by using the **BATCHID** field or the **BNDLFLAG** field. These two fields are not related and the transaction associations formed by each are different.

BATCHID creates an informal association provided as a fast, convenient way to select transactions from the Transaction Store. An alternate index is keyed on the **BATCHID** value making it the best field on which to base a selection if you do not use the **THANDLE** field.

For example, you could use this field to isolate all the transactions from a particular program execution. If a problem occurs with the execution, you can use **BATCHID** to place all the transactions from that execution on hold, preventing them from being enveloped until the problem is resolved. You can also use **BATCHID** to reenvelope transactions from a certain program execution if the file containing the interchanges was corrupted.

However you use it, the **BATCHID** association between transactions is informal and differs little from a set of transactions created using any common value, such as trading partner nickname or EDI standard ID.

On the other hand, **BNDLFLAG** creates a formal relationship between transactions. The transactions associated with **BNDLFLAG** are bundled (clustered) and are generally treated as a single unit. Your application can use bundling if you want to isolate a group of transactions and have them processed as a single unit rather than as individual transactions.

Bundling is required for UN/TDI transactions but is optional for the other EDI standards. With all other EDI standards, transactions are independent of each other and can be handled independently. For example, X12 defines a header segment, detail segments, and a trailer segment within a transaction, while UN/TDI defines a header transaction, detail transactions, and a trailer transaction. You cannot send the header transaction in one interchange, and the detail and trailer transactions in another interchange. You must send them all together in the same interchange. Do not include any additional transactions in that interchange. By bundling, you tell DataInterchange which header belongs with which detail and trailer transactions. DataInterchange 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 DataInterchange 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 DataInterchange first and then translated. Next, one or more detail records (line items) are passed into DataInterchange and then translated. You can pass in and translate as many detail records as necessary. Finally, the trailer is passed into DataInterchange 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 should only be mapped in maps used for incremental translation. Incremental translation should 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 308.

Incremental translation

The header records are first passed into DataInterchange, 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 DataInterchange to translate the header, and then remove the header application data from memory. Make this call without data.

This next step is iterative. One or more detail records may be passed into DataInterchange, 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 DataInterchange to translate the detail records, and then 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 DataInterchange, 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 DataInterchange to translate the trailer, and then 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 DataInterchange can transfer with pageable translation is approximately one gigabyte (specifically, 32,000 multiplied by 2863 bytes).

A sample definition of EDIVAX is:

```
//EDIVAX DD DISP=(NEW,DELETE,DELETE),UNIT=SYSDA,SPACE=(CYL,25)
```

In MVS, 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:

$$\begin{array}{rclclcl} \text{Number of bytes in} & & \text{Number of bytes in} & & & & \text{Number of structures in} \\ \text{largest interchange} & + & \text{largest application} & + & \text{4 MB overhead} & + & \text{largest interchange} \\ & & \text{transaction image} & & & & \text{multiplied by 120 bytes} \end{array}$$

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 may 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 DataInterchange 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 **Application data format** field) as defined by a map (from the **Trading partner transactions** field).

This is the same API that the DataInterchange 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 420.
Deenveloping data	Enveloping/Deenveloping services (function code 214). See "Deenvelope API" on page 384.
Translating data	Transaction services (function code 213). See "Translate specific API" on page 341.

The Communication service invokes the network and receives the data from the network into the specified file.

The Deenveloping service:

- Parses the interchanges in the file
- Extracts each transaction
- Places details of each transaction along with the transaction image into the Transaction Store
- Generates a functional acknowledgment, if one was requested
- Reconciles any received acknowledgments with the original transactions

The Translation service retrieves the transaction image from the Transaction Store and translates the data into the application format.

You can combine the deenvelope and translate functions into one step by calling the Translation service with a function code of **212**. For more information, see “Translate-file-to-application API” on page 350.

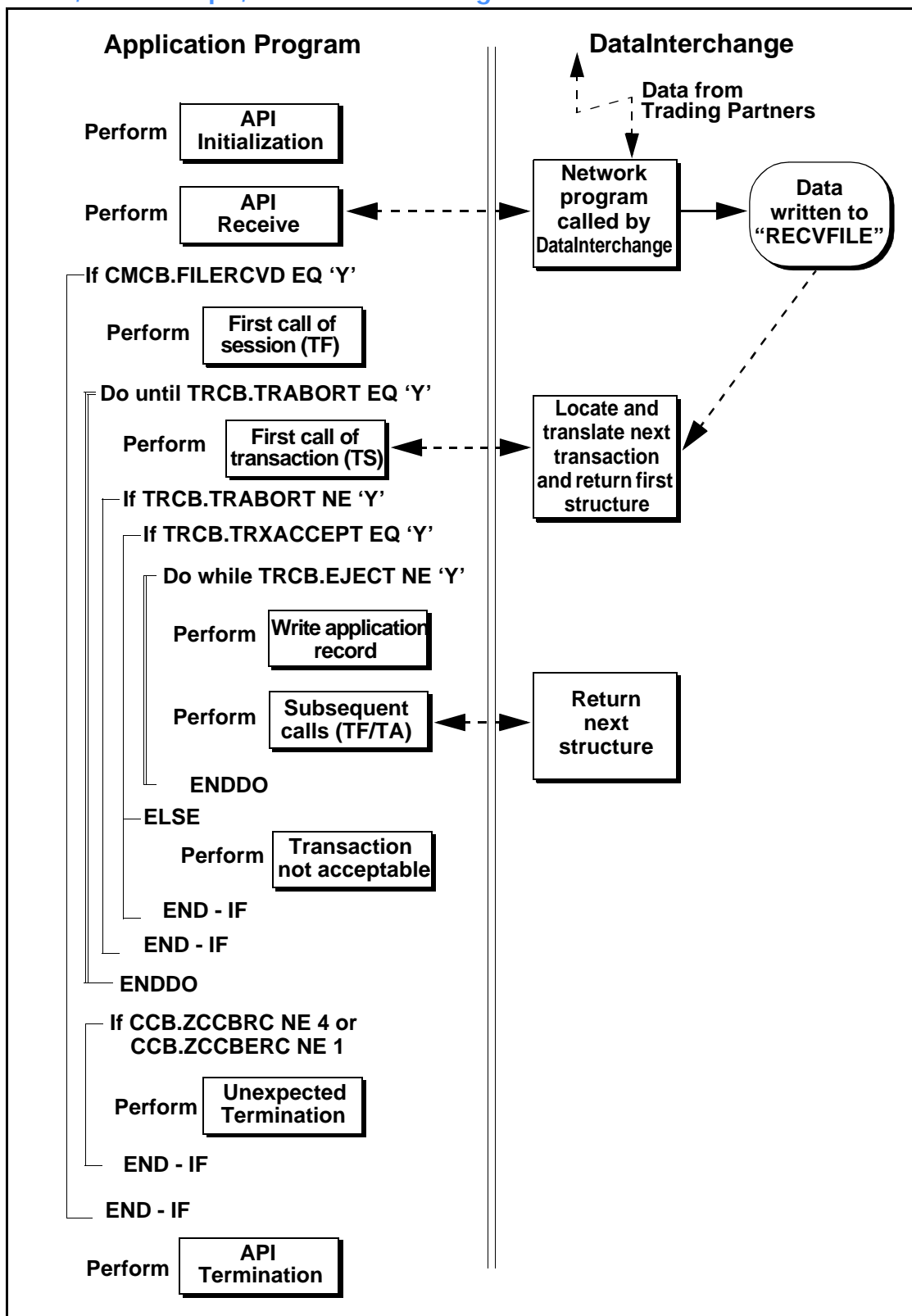
Processing speed can be improved when you use the DEENVELOPE AND TRANSLATE combination command. This sequence is illustrated in “Receive, deenvelope, and translate diagram” on page 340.

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. DataInterchange counts the bytes in the interchanges in the file in order to update the management reporting component of DataInterchange. No information from the interchanges is recorded in the Transaction Store.
- The deenvelope function generates the functional acknowledgment for the data received and reconciles the received functional acknowledgments with the original transactions. This is true whether code **214** (deenvelope only) or code **212** (deenvelope and translate) is used.
- Because deenveloping and translation can be separate functions, you need a way to tell the translator which transaction from the Transaction Store to process. Use the transaction handle (**THANDLE**) that is assigned to the transaction when it is added to the store. The program that is deenveloping the data should track the transaction handle. If the program does not track the transaction handle, you can retrieve a list of these values from the Transaction Store using the **QUERY** command.
- No single API function is the equivalent of the DataInterchange Utility **RECEIVE AND TRANSLATE** command. Using the API, receiving the data, and deenveloping and translating the data are two separate steps.

Receive, develope, and translate diagram



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 trading partner, use production mode.

When you send test data in text mode (function code **211**), the following occurs:

- Data is translated to application format in test mode. Your program follows the same steps as for production mode. Interchanges are read from a file and parsed. Transactions are translated and presented to your program one transaction at a time.

However, no information is written to the Transaction Store, and no functional acknowledgments are generated or reconciled. If you want information written to the Transaction Store, or if you want functional acknowledgments returned to your trading partner, you must use the second test method outlined below.

- When you use the test function code, DataInterchange looks for a test usage/rule (if one is available) and forces test mode. You can verify the results of your program's efforts by examining the application data produced, as well as the information recorded in the event log. To view or print the event log, choose Event logging from the Administrator's Menu.

When you send test data in production mode, the following occurs:

- The data is sent with the **test indicator** set to **T** in the interchange header segment, if the envelope type being used has a test indicator. When you use the test indicator, translation and functional acknowledgment processing occurs as usual (including updating the Transaction Store).
- The interchange that contains the functional acknowledgment is identified as being for test purposes. You may 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 briefly in the table below and in more detail in the following sections.

Translate Specific	Translates a specific transaction into application format. Use this API to translate a transaction that has been deenveloped or deenveloped and translated, and is already in the Transaction Store.
Translate File	Deenvelopes and translates all the transactions in a file. Use this API when all the interchanges in a file have been processed, and all transactions in the interchanges have been deenveloped, translated, and added to the Transaction Store.

Translate specific API

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

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

The unique parameters for the Translate specific API request are:

Parameter	Description
SNB	ZSNBLL 32 ZSNBNAME TRANPROC ZSNBPC 6
FCB	ZFCBFUNC 213
TRCB	The translator control block. See “Translator Control Block (TRCB)” on page 586 for details about this block.
TRIDB	The input data block. DataInterchange uses this block as a work buffer. This buffer must be at least 32000 bytes in length. Its maximum size should be the size of the maximum EDI standard segment that is received, excluding the BIN segment. See “Translator Input Data Block (TRIDB)” on page 616 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 618 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 tables describe the control block fields and initialization considerations for the first session call.


SNB initialization for translate-to-application

SNB	Initialization
ZSNBLL	32
ZSNBNAME	TRANPROC
ZSNBPC	6 (all calls for translation services have six parameters)

FCB initialization for translate-to-application

FCB	Initialization
ZFCBLL	4
ZFCBFUNC	213

TRCB initialization for translate-to-application

Field name	Initialization
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 DataInterchange should check the BLKLEN field to determine the software version and release being used.
BATCHID	The batch ID for a group of transactions. The Transaction Store creates an alternate key value using the batch ID. Other than using the transaction handle, searching on batch ID is the quickest way to retrieve a transaction from the Transaction Store during the selection process. The default is <i>DDHHMMSS</i> , where <i>DD</i> is the current day of the month, and <i>HHMMSS</i> is the current time.
	NOTE: This field is checked with each call to translator. The value in this field when the first call of transaction (TA) is made will be associated with the transaction.
ERRFILTER	Indicates which error codes to filter out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more detailed information, see “Error filtering” on page 21.
MAPCHAIN	Indicates whether mapchaining is in effect. Y Translates the current transaction again using the value in the variable DIMAPCHAIN to select the map (other) Translates the next transaction
FORCETEST	Indicates whether the translate process is forced to select only test usages/rules. If a value of Y is used in this field for the DEENVELOPE command, you must also use it with the TRANSLATE TO APPLICATION command to select only deenveloped transactions.

TRIDB initialization for translate-to-application

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	Binary zeros.

TRODB initialization for translate-to-application

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	Binary zeros.

First call for transaction (TA)

The following TRCB fields are used on the first call for transaction and should be set before the first call for transaction is made.

Field name	Initialization
TSKEY	The transaction handle for the transaction you want to translate. The format of the handle is <i>YYYYMMDDHHMMSSxxxxnnnn</i> 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 <i>YYYYMMDDHHMMSSxxxxnnnn</i> 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 should automatically fill in the values for the internal trading partner ID and the record ID before returning the structure to your application. <div style="display: flex; justify-content: space-between;"> <div style="width: 10%;">Y</div> <div>Fills in the values for internal trading partner ID and record ID</div> </div> <div style="display: flex; justify-content: space-between;"> <div style="width: 10%;">(other)</div> <div>Does not fill in the values for internal trading partner ID and record ID.</div> </div>

The following TRODB fields are used on the first call for transaction and should be set before the first call for transaction is made.

Field name	Initialization
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.

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

On the first call to the translator for a transaction, the transaction identified in the **TSKEY** or **TSKEYU** field is retrieved from the Transaction Store along with the interchange, group, and transaction header images. The interchange sender ID and sender qualifier are extracted from the interchange header. These values are used to identify the trading partner who sent the data. For a description of the information used to identify the trading partner, see “Locating sending trading partner profile members” on page 396.

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 397.

Numerous errors can occur at this point. See “Translation return codes” on page 680 for a description of the possible errors generated by the translator. 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 should 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 below provide some basic attributes of the transaction just processed, and relate to the transaction side of the processing rather than the application side.

Field name	Description
TRNID	The standard transaction or message ID for the transaction or message received
TEST	Indicates the type of transaction I Information P Production T Test
MAPKEY	The map used to translate the EDI standard data
TPNICK	The trading partner nickname associated with the transaction
DUPTRAN	Indicates whether the interchange being received has been received before
TRXACCEPT	Indicates whether the transaction had an acceptable translation and indicates that application data has been returned in the TRODB, unless the EJECT field has a value of Y .
TRABORT	Indicates whether an error occurred that was so severe that the translator did not continue. If this field is set to Y , the translator has terminated of its own accord.
TSKEY	The key value assigned to the transaction in the Transaction Store. This is called the transaction handle and has a format of YYYYMMDDH-HMMSSxxxxnnnn . It is returned as a packed 10-byte value in this field, which is how it is stored in the database.
TSKEYU	The key value assigned to the transaction in the Transaction Store. It has the same value as the TSKEY field, except that TSKEYU is an unpacked 20-byte value.

Field name	Description
EJECT	Indicates whether the end of the transaction has been reached. For more information, see “Partial structures (TA)” on page 363.
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 614.

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

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.
APPFIL	The ddname for the application file to which the application data should be written. This field is taken from the data format definition, but can be overridden in the trading partner receive map rule record.
APPTYPE	The type of file identified in APPFILE. Applies only to CICS and MQ (which is supported in both MVS and CICS). Valid values are: <ul style="list-style-type: none"> MQ MQSeries queue profile member name PG Program identified in APPFILE TD TD queue name identified by the first four characters of APPFILE TM TS queue (main) identified in APPFILE TS TS queue (auxiliary) identified in APPFILE TX CICS transaction code identified by the first four characters of APPFILE For MVS, 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.

Field name	Description
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 363 for special considerations.

The following TRODB fields are concerned with the application data.

Field name	Description
DATALEN	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 following TRCB fields are concerned with the interchange for the current transaction.

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 .

Field name	Description
ISYNTAXVER	The interchange syntax version for envelope types E and T .
ISIDQUAL	The interchange sender ID qualifier for envelope types E , I , and X .
ISID	The interchange sender ID (interchange header with data type IS).
ISENDNAME	The interchange sender name for envelope types U and T .
IREVROUT	The interchange reverse routing for envelope type E .
IRIDQUAL	The interchange receiver ID qualifier for envelope types E , I , and X .
IRID	The interchange receiver ID (interchange header with data type IR).
RECVNAME	The interchange receiver name for envelope types U and T .
IROUTEADDR	The interchange routing address for envelope type E .
IDATE	The interchange date (interchange header with data type DT).
ITIME	The interchange time (interchange header with data type TM).
IVERREL	The interchange version and release (interchange header with data types VR or LV).
ISPW	The interchange password (interchange header with data type PW).
IAPREF	The application reference (interchange header with data type AP).
ISTDID	The interchange standard ID for envelope types I and X .
IPRIOR	The interchange priority code for envelope types E and T .
ICOMMAGREE	The interchange communication agreement for envelope types E and T .

The following TRCB fields are concerned with the group for the current transaction.

Field name	Description
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).

Field name	Description
GVER	The group version (group header with data type VR).
GREL	The group release (group header with data type LV).
GRESAGENCY	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.

Field name	Description
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 page 345.
TVER	The transaction version (transaction header with data type VR).
TREL	The transaction release (transaction header with data type LV).

Subsequent calls (TF/TA)

The items listed below summarize the activity that takes place on the first call for a transaction (TA):

1. The next transaction in the file is located or the specific transaction you requested is retrieved from the Transaction Store.
2. The map associated with the transaction is found.
3. The translation from EDI standard format into application structures takes place.
4. The application structures are sorted into the order defined by the data format definition.
5. The first application structure is returned to the calling program in the TRODB.

If the first call for a transaction was successful, the **TRXACCEPT** field has a value of **Y**, and the application continues to call the translator using the following call with the same parameters as the first call of a transaction):

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

The application continues to call the translator until the **EJECT** field has a value of **Y**. For each call made, the next application structure is returned in the TRODB and the name of the structure is placed in the **ATSID** field.

Last call for transaction (TA)

The translator signals that the last call for a transaction has just been made by setting the **EJECT** field to a value of **Y**. No data accompanies this call. At this point, your application might perform application-related processing relative to the end of the transaction, including updating the application databases. The next call is interpreted as the first call for the next transaction. The translator issues a COMMIT request with the next call.

Last call of session (TA)

When all the specific transactions have been processed, call the translator one last time with an end-translation function code. For more information, see “End translation/enveloping API” on page 384.

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:

- DataInterchange 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.
- DataInterchange automatically logs the EDI standard image received and the application data produced.
- DataInterchange does not write any data to the Transaction Store to prevent cluttering up the store with test data.
- Functional acknowledgments are not generated even if requested in the usage record.
- Statistics maintained by the Management Reporting component of DataInterchange are not updated.

The basic format of the API request to translate-to-application format is:

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

The unique parameters for this API request are:

Parameter	Description	
SNB	ZSNBLL	32
	ZSNBNAME	TRANPROC
	ZSNBPC	6
FCB	ZFCBFUNC	212 (production)
		211 (test)

Parameter	Description
TRCB	The translator control block. For more information, see “Translator Control Block (TRCB)” on page 586.
TRIDB	The input data block. DataInterchange uses the input data block as a work buffer. The minimum size is 32000 bytes. Maximum size should be the size of the largest EDI standard segment that is received excluding the BIN segment. See “Translator Input Data Block (TRIDB)” on page 616 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 618 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 should 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



Field name	Initialization
ZSNBLL	32
ZSNBNAME	TRANPROC
ZSNBPC	6 (all calls for translation services have six parameters)




FCB initialization for translate-file-to-application

Field name	Initialization
ZFCBLL	4
ZFCBFUNC	212 (production) 211 (test)

TRCB initialization for translate-file-to-application

Field name	Description
BLKLEN	1536.
BLKNME	EDITRCB.

Field name	Description
BLKTYPE	<p>The format for the TRIDB and TRODB buffers.</p> <p>H Unlimited size</p> <p>(other) Limited to 32768 bytes</p>
XPANDED	Y. Indicates that DataInterchange should check the BLKLEN field to determine the software version and release being used.
ENVLDELAY	<p>Indicates whether functional acknowledgments should not be enveloped.</p> <p>Y Does not envelope functional acknowledgments</p> <p>(other) Envelopes functional acknowledgments</p>
DUPTRAN	<p>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:</p> <p>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).</p> <p>Y or (other) Processes duplicate interchanges and returns transactions flagged as duplicate transactions. Y is the recommended value.</p>
SCOPE	<p>Indicates the level of recovery required.</p> <p>E Issues a database COMMIT on the completion of every interchange. Applies only when enveloping is performed.</p> <p>(other) Issues a database COMMIT at the start of every transaction</p> <p> NOTE: This field contains values that affect the recovery scope during the session. For more information, see "Send Recovery Scope:" on page 334.</p>
INMEMTRANS	<p>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.</p> <p>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 586.</p> <p> NOTE: The value set in this field affects the recovery scope during the current session. For more information, see "Send Recovery Scope:" on page 334.</p>
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.

Field name	Description
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 should 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	<p>The batch ID of a group of transactions. The Transaction Store creates an alternate key using the batch ID. Other than using the transaction handle, searching on batch ID is the quickest way to retrieve a transaction from the Transaction Store during the selection process. The default is <i>DDHHMMSS</i>, where <i>DD</i> is the current day of the month, and <i>HHMMSSL</i> is the current time.</p> <div>  <p>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.</p> </div>
TRXLIFE	<p>The amount of time that a transaction remains in the Transaction Store before it is eligible for purging. The default is 30 days.</p> <div>  <p>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.</p> </div>
IMGLIFE	<p>The amount of time that a transactions image (that is, the EDI standard data produced) remains in the Transaction Store. The default is 30 days.</p> <div>  <p>NOTES:</p> <ol style="list-style-type: none"> 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. </div>
ERRFILTER	Indicates which error codes to filter out during this session. The values specified here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more information, see “Error filtering” on page 21.

Field name	Description
MAPCHAIN	Indicates whether mapchaining is in effect. Valid values are: Y Translates the current transaction again (using the map identified in the DIMAPCHAIN variable) (other) Translates the next transaction
FORCETEST	Indicates whether the translate process is forced to select only a test usage/rule. If you specified a value of Y for this keyword on the DEENVELOPE command, then you must also specify a Y on the TRANSLATE TO APPLICATION command to select only deenveloped transactions.

TRIDB initialization for translate-file-to-application

Field name	Initialization
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

Field name	Initialization
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)

The following tables describe the fields you should set before the first call for a transaction is issued.

TRIDB initialization - first call for transaction

Field name	Description
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

Field name	Description
HOLDFLAG	Indicates whether the transaction is placed in HELD status when added to the Transaction Store. A transaction in HELD status is not available for any other activity until you release it.
Y	Places the transaction in hold status
N or (other)	Does not place the transaction in hold status

TRODB initialization - first call for transaction

Field name	Initialization
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 396 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 may 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 397 for a description of the fields used and the search to locate a transaction.

Numerous errors can occur at this point. For a description of error codes generated by the translator, see "Translation return codes" on page 680. 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 should check the **TRXACCEPT** and **TRABORT** fields for error information.

Field name	Description
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 may 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 below provide information on basic attributes of the transaction. This information is about the transaction side of the processing rather than the application side.

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.

Field name	Description
TRABORT	Indicates whether an error occurred that was so severe the translator stopped processing. Y The error was so severe that translation was halted. (blank) No errors occurred.
TSKEY	The key value assigned to the transaction in the Transaction Store. This is called the transaction handle and has a format of <i>YYYYMMDDH-HMMSSxxxxnnn</i> . It is returned as a 10-byte packed value in this field, which is how it is stored in the database.
TSKEYU	The key value assigned to the transaction in the Transaction Store. This is the same value as the TSKEY field, except that this field contains an unpacked 20-byte value.
EJECT	Indicates whether the end of the transaction has been reached. See “Partial structures (TA)” on page 363 for related considerations. Y This transaction is complete. No data is returned and the next call is treated as a first call for transaction. (other) More data remains for the current transaction. The next call will return the next structure or an EJECT value of Y .
ERRNUM	The total number of errors flagged during data translation.
ERRCODES	An array of the first 10 different errors flagged during data translation. For more information, see “Translator Error Codes” on page 614.

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

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.
APPPFILE	The ddname name for the application file to which the application data should be written. This value is taken from the data format definition but can be overridden in the map receive usages/rules record.
APPTYPE	The type of file represented by APPFILE. Applies only to CICS and MQ (which is supported in both MVS and CICS). Valid values are: MQ MQSeries queue profile member name PG Program identified by APPFILE TD Transient data queue name identified by the first 4 characters of APPFILE TM Temporary storage queue (main) identified by APPFILE TS Temporary storage queue (auxiliary) identified by APPFILE

Field name	Description
TX	CICS transaction code identified by the first 4 characters of APPFILE For MVS, 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 363 for special considerations.

The TRODB fields described below are related to application data.

Field name	Description
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.

Field name	Description
FABUILT	The envelope type of the functional acknowledgment. Valid values are: E UNB/UNZ. I ICS/ICE. T STX/END.

Field name	Description
U	BG/EG.
X	ISA/IEA.
G	The acknowledgment does not have an interchange header.
S	The acknowledgment was written to the Transaction Store, but was not enveloped.
FARC	The return code from the translator for the functional acknowledgment translation performed during deenveloping. For more information about return codes, refer to <i>DataInterchange Messages and Codes</i> .
FAERC	The extended return code from the translator for the functional acknowledgment translation performed during deenveloping. For more information about return codes, refer to <i>DataInterchange Messages and Codes</i> .

The TRCB fields described below 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.

Field name	Description
DSNAME	The physical data set name from which transactions are processed.
QTPNICK	The trading partner nickname for which the last transaction was processed.
QSIZE	The total number of bytes that were in the interchange, stored as a 4-byte binary value.
QBT	The character representation of the QSIZE field.
ENVTYPE	The type of envelope type received.
IHCTL	The interchange control number assigned to the current interchange. This value is returned in this field as a right-justified value with leading zeros. Applies to version 2.1 or earlier.
IHXCTL	The interchange control number assigned to the current interchange. This value is returned in this field as a right-justified value with leading zeros. Applies to version 3.1 or later.
ISYNTAXID	The interchange syntax ID for envelope types E and T .
ISYNTAXVER	The interchange syntax version for envelope types E and T .
ISIDQUAL	The interchange sender ID qualifier for envelope types E , I , and X .
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 .

Field name	Description
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	<p>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 399.</p> <p>Y Starts a new interchange</p> <p>(other) Does not start a new interchange</p>
GRPNUM	The total number of groups processed so far in the current interchange, stored as a 4-byte binary value. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed. See IGT for the character representation of this field.
TRNNUM	The total number of transactions processed so far in the current interchange, stored as a 4-byte binary value. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed. See ITT for the character representation of this field.
SEGNUM	The total number of segments processed so far in the current interchange, stored as a 4-byte binary value. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed. See IST for the character representation of this field.
ESIZE	The total number of bytes processed so far in the current interchange, stored as a 4-byte binary value. The value in this field changes as the transactions in the interchange are processed, indicating how much of the interchange has been processed. See IBT for the character representation of this field.
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.

Field name	Description
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 below are associated with the group to which the current transaction belongs.

Field name	Description
GHCTL	The group control number assigned to the current group. The value returned in this field is right-justified with leading zeros. This field value is established when the first transaction for the group is processed. Applies to version 2.1 or earlier.
GHXCTL	The group control number assigned to the current group. The value returned in this field is right-justified with leading zeros. This field value is established when the first transaction for the group is processed. Applies to version 3.1 or later.
GSIDQUAL	The group sender ID qualifier for type E envelope groups.
GSID	The group sender ID (group header for data type AS). This value in this field is established when the first transaction for the group is processed.
GRID	The group receiver ID (group header for data type AR data type). This value in this field is established when the first transaction for the group is processed.
GRIDQUAL	The group receiver ID qualifier for type E envelope groups.
GDATE	The group date (group header for data type DT). This value in this field is established when the first transaction for the group is processed.
GTIME	The group time (group header for data type TM). This value in this field is established when the first transaction for the group is processed.
GAPW	The group password (group header for data type PW). This value in this field is established when the first transaction for the group is processed.
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.
GRESAGENCY	The group responsible agency code for types E , U , and X envelopes.

Field name	Description
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 400. 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 below are associated with the transaction header and trailer for the current transaction.

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 400. Y A transaction header is available. (other) A transaction header is not available.
LASTINENV	Indicates whether this transaction is the last transaction in the current interchange.
THCTL	The transaction control number assigned to the current transaction. The value is returned in this field as a right-justified value with leading zeros. Applies to version 2.1 or earlier.
THXCTL	The transaction control number assigned to the current transaction. The value is returned in this field as a right-justified value with leading zeros. Applies to version 3.1 or later.
SEGTRN	The total number of segments in the current transaction, stored as a 4-byte binary value. See TST for the character representation of this field.
TTC	The current transaction or message ID value. See also TRNID on page 345.
TVER	The transaction version (transaction header for data type VR).
TREL	The transaction release (transaction header for data type LV).
TST	The character representation of SEGTRN .

Subsequent calls (TF/TA)

The following activities take place on the first call for a transaction (TF):

- The next transaction in the file is located or the requested transaction is retrieved from the Transaction Store.
- The map associated with the transaction is found.
- The translation from EDI standard format into application structures takes place.
- The application structures are sorted into the order defined by the data format definition.
- The first application structure is returned to the calling program in the TRODB.

If the first call for a transaction was successful, the **TRXACCEPT** field has a value of **Y** and the application continues to call the translator until the **EJECT** field has a value of **Y**. The application uses the following call:

```
FXZXccc(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 should 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.

DataInterchange automatically clusters all the transactions for a UN/TDI interchange, but does not cluster transactions from any other interchange type.

If the DataInterchange 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 DataInterchange 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 may perform differently.

DataInterchange 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 should 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 should be present at the next level. For example, the control count in the interchange trailer indicates how many groups should be present; the control count in the group trailer indicates how many transactions should be present; and the control count in the transaction trailer indicates how many segments should 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. DataInterchange supports the following EDI enveloping standards:

- EDIFACT envelope standard - envelope and profile ID is **E**

Segment ID	Purpose
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**

Segment ID	Purpose
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**

Segment ID	Purpose
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 DataInterchange during receive processing but are never generated by DataInterchange.

- UCS envelope standard - envelope and profile ID is **U**

Segment ID	Purpose
BG	Interchange header
GS	Group header
ST	Transaction header
SE	Transaction trailer
GE	Group trailer
EG	Interchange trailer

- X12 envelope standard - envelope and profile ID is **X**

Segment ID	Purpose
ISA	Interchange header
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 should receive the data. The network uses the sender ID in the interchange header for routing error messages and network acknowledgments for the interchange. DataInterchange 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 DataInterchange 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 then routed to the proper destination based on the values in the group header.

DataInterchange 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 397. 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 may 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 DataInterchange to locate a map for translating the transaction. The transaction header contains a control number that must be unique in the group (if groups are being used), or unique in the interchange (if groups are not being used).

The transaction trailer also contains a control number that must match the control number in the header and a count field that identifies the number of segments included in the transaction. This number must match the actual number of segments received or processing is stopped and an error message is generated.

Enveloping service

The enveloping service provides the following two functions:

- An enveloping function that extracts transactions from the Transaction Store and builds transaction headers and trailers, organizing the transactions into groups, and the groups into interchanges. The enveloping function adds the EDIVTSEV, EDIVTSGP and EDIVTSTU records to the Transaction Store database.
- A deenveloping function that locates interchanges in a file, parses the interchange into groups and transactions, and adds the transactions to the Transaction Store. The deenveloping function also adds the EDIVTSEV, EDIVTSGP and EDIVTSTU records to the Transaction Store database.

These functions are described in detail in the following sections.

Envelope API

This section describes the details of the envelope API request. The envelope function extracts transactions from the Transaction Store and then 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 311 and “Translate-to-standard API” on page 317.

The API that is described below is the same API that the DataInterchange 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 381 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 380 for an explanation of the fields that will start a new interchange.

When DataInterchange 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, then 10 interchanges are created. Sorting the transactions first would yield 2 interchanges (A + B) rather than 10.

An application program using the envelope API must obtain a list of the transactions to envelope. Some ways to obtain a transaction list from the Transaction Store are:

- Set the application program that is adding the transactions to save the transaction handle values (and any other information needed for sorting) to a file that is then sorted and read by your enveloping application.
- Use the QUERY command to select transactions. The transaction handles for the selected transactions are written to the EDIQUERY file in transaction handle sequence. This does not result in optimal enveloping.
- Use the TRANSACTION DATA EXTRACT command to select transactions. As with the QUERY command, the transactions are written to the EDIQUERY file in transaction handle sequence, but this command provides complete transaction information that can be used to sort the transactions into a optimal sequence for enveloping.

Envelope API

The basic format of the API request to envelope a transaction is:

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

The unique parameters for this API request are.

Parameter	Description
SNB	ZSNBNAME TRANPROC ZSNBPC 6
FCB	ZFCBFUNC 215
TRCB	The translator control block. For more information, see “TRCB field descriptions” on page 590.
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 617.
TRODB	The output data block. The output data block is used by DataInterchange 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 619.

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


Field name	Initialization
ZSNBLL	32
ZSNBNAME	TRANPROC
ZSNBPC	6 (all calls for enveloping services have six parameters)

FCB initialization for enveloping function

Field name	Initialization
ZFCBLL	4
ZFCBFUNC	215

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
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.

Field name	Description						
INMEMTRANS	<p>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.</p> <p>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 590.</p> <div>  <p>NOTE: This value affects the recovery scope during the session.</p> </div>						
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 5, “Exit routines”.						
IUSERAREA	A pointer to a user-defined area. The pointer is passed to the user exit defined in IUSEREXIT field.						
ITPBREAK	<p>Indicates whether a change in the internal trading partner ID starts a new interchange. Valid values are:</p> <table> <tr> <td>Y</td><td>Each interchange contains data for a single vendor number.</td></tr> <tr> <td>N or (other)</td><td>All data for a trading partner is included in a single interchange, regardless of vendor number.</td></tr> </table>	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.		
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	<p>Indicates whether the translator checks the transaction’s enveloping status before processing the transaction. See “Envelope versus reenvelope” on page 379 for more detailed information. Valid values are:</p> <table> <tr> <td>1</td><td>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).</td></tr> <tr> <td>2</td><td>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).</td></tr> <tr> <td>(other)</td><td>Does not verify the transaction’s enveloping status.</td></tr> </table>	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.
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.						

Field name	Description
ERRFILTER	Indicate which error codes to filter out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more detailed information, see "Error filtering" on page 21.
RAWDATAOUT	Indicates whether RAWDATA output is used for fixed-to-fixed mappings. Y Uses RAWDATA output (other) Uses C & D record output
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

Field name	Initialization
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

Field name	Description
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.

Field name	Description
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 <i>YYYYMMDDH-HMMSSxxxxnnnn</i> . 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 <i>YYYYMMDDH-HMMSSxxxxnnnn</i> . This field is only used if TSKEY does not have a value.

When initialization is complete, the enveloper is invoked with the following API request:

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

The transaction identified by the **TSKEY** or **TSKEYU** field is retrieved from the Transaction Store along with the other database records needed, such as the trading partner profile member, and the send usage/rule and map. Everything that was needed to translate the transaction (except the control string) is also needed when the transaction is enveloped.

Numerous errors can occur at this point. These errors are described in “Translation return codes” on page 680. 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 should 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 below provide some basic attributes of the transaction just processed. These fields are related to the transaction side of the processing.

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.

Field name	Description
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 <i>YYYYMMDDH-HMMSSxxxxnnn</i> .
TSKEYU	The transaction handle for the transaction you want to envelope, formatted as a 20-byte character field. The format of the handle is <i>YYYYMMDDH-HMMSSxxxxnnn</i> .
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 586 and “Translation return codes” on page 680.

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

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.
APPLTPID	The application trading partner.
APPCTLNUM	See XACFIELD .
XACFIELD	The application control value defined by the application field with data type of AC or by the fields that are assigned when the map is created. The application control value uniquely identifies the transaction to the application.
MAPCHAIN	Indicates whether mapchaining is in effect. Y Translates the current transaction again. (other) Translates the next transaction

Enveloping - TRCB Fields: 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 below are 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 399. 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.
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).

Field name	Description
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 below are related to the 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 400. Y Started a new group (other) Did not start a new group
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).

Field name	Description
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 .
GRESAGENCY	The group controlling agency for envelope type E . The group responsible agency for envelope types I , U and X .

The TRCB fields described below are related to the 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 400. Y Started a new transaction (other) Did not start a new transaction
THCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros. Applies to version 2.1 or earlier.
THXCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros. Applies to version 3.1 or later.
SEGTRN	The total number of segments in the current transaction stored as a 4-byte binary value. See TST for the character representation of this value.
TTC	The current transaction or message ID value. See also TRNID on page 345.
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 below 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.

Field name	Description
EJECT	Indicates whether an interchange was written to the file associated with the network. Valid values are: <div> <div>Q</div> <div>The interchange was written successfully.</div> </div> <div> <div>E</div> <div>The interchange was not written successfully.</div> </div> <div>  <div> 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 DataInterchange Communication services. The most likely error is that the ddname could not be opened. </div> </div>
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 380 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 384.

Envelope processing considerations (EV)

The following sections list special considerations for envelope processing.

Envelope versus reenvelope

Both the DataInterchange Utility and Transaction Store Facility provide a means of enveloping and reenveloping transactions. When the **ENVELOPE** function is used, only transactions in the Transaction Store that have never been enveloped are considered for enveloping. When the **REENVELOPE** function is used, only transactions in the store that have already been enveloped are considered for reenveloping.

The API services do not provide a similar distinction. The application program making the API request must distinguish whether a transaction is enveloped or reenveloped. However, you can set the **ENVCHK** field in the TRCB to indicate the intentions of the application (**ENVELOPE** or **REENVELOPE**), and the envelope service will verify that the status of the transaction matches the intentions of the application. The **ENVCHK** field has the following values and meanings:

Value Meaning

- | | |
|----------------|--|
| 1 | Envelopes a transaction. If the transaction has ever been previously enveloped, attempting to envelope it again should 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 should be considered an error. The translator issues message TR0121 and returns a transaction level error (extended return code value of 3) to the application. |
| (other) | Does not check the status of the transaction. |

Clustered transactions (EV)

You can cluster (batch) transactions during translation. The Transaction Store Facility and DataInterchange 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 “**BNDLFLAG**” on page 322.

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 412) 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 DataInterchange 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 may 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 should 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 should 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 should 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 (GRESAGENCY)

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 DataInterchange 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:

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 590 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 619 for a general description of this block.

Queueing - TRCB Fields: The TRCB fields described below 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.

Field name	Description
EJECT	Indicates whether an interchange was written to the file associated with the network. Valid values are: <div> <div>Q</div> <div>The interchange was written successfully.</div> <div>E</div> <div>The interchange was not written successfully.</div> <div>  <div> 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 DataInterchange communications services. The most likely error is that the ddname could not be opened. </div> </div> </div>
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: <div> <div>Y</div> <div>The network is a point-to-point network.</div> <div>(other)</div> <div>The network is not a point-to-point network.</div> </div>
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.) <div> <div>Y</div> <div>The network has a send and receive program.</div> <div>(other)</div> <div>The network does not have a send and receive program.</div> </div>
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 380 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:

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 should be the size of the largest standard segment (excluding the BIN segment) that will be produced. See “TRODB field descriptions” on page 619 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 382 for an explanation of these fields.

Deenvelope API

This section describes the details of the deenvelope API request. Deenveloping extracts interchanges from a file and parses them to extract transactions to add to the Transaction Store. The syntax of the interchange or transaction is checked and functional acknowledgments are generated. Received acknowledgments are reconciled with the original transactions.



NOTE: Deenveloping does not need to be a separate API request. You can deenvelope and translate transactions at the same time. For more information, see “Receiving and deenveloping” on page 338 and “Translate-file-to-application API” on page 350.

The deenvelope API described in the following section is the same API used internally by the DataInterchange Utility when you issue any of the following PERFORM commands:

- DEENVELOPE
- DEENVELOPE AND TRANSLATE
- RECEIVE AND DEENVELOPE
- RECEIVE AND TRANSLATE

Each time the deenveloping API is invoked, the next transaction from the file being processed is located and added to the Transaction Store. Sometimes locating the next transaction involves locating the next interchange in the file. Once an interchange is located, the complete interchange is read into virtual storage. A syntax check and validation are done on the service segments to verify that the interchange received has a valid format and is consistent with the specified format.

The basic format of the API request to deenvelope transaction from a file is:

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

The unique parameters for this API request are:

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 586.
TRIDB	The input data block. This buffer must be at least 32000 bytes in length, but its maximum size should be the size of the largest standard segment (excluding the BIN segment) that is received. See “Translator Input Data Block (TRIDB)” on page 616 for a general description of this block.
TRODB	The output data block. DataInterchange 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 618 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 should also follow the instructions in the next section, “Deenvelope transaction (DE). The control blocks, the fields within the control blocks, and the initialization considerations are described in the following tables.


SNB initialization for deenveloping


Field name	Description
ZSNBLL	32
ZSNBNAME	TRANPROC
ZSNBPC	6 (number of calls for enveloping services)




FCB initialization for deenveloping

Field name	Description
ZFCBLL	4
ZFCBFUNC	214 (deenvelope)

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 DataInterchange should check the BLKLEN field to determine the software version and release being used.
ENVLDELAY	Indicates whether functional acknowledgments should 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 334.

Field name	Description
INMEMTRANS	<p>Applies only if the value of SCOPE is E. Indicates 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.</p> <p>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 586.</p> <div>  <p>NOTE: This field contains values that affect the recovery scope during the session. For more information, see “Send Recovery Scope:” on page 334.</p> </div>
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 DataInterchange instead of reading the input file. This exit is used when the envelope is to be stored into DataInterchange through the Put Envelope service before deenveloping. For more information on the Get/Put Envelope Exit processing, see Chapter 5, “Exit routines”.
IUSERAREA	A pointer to a user-defined area. The pointer is passed to the user exit defined in IUSEREXIT field.
IUSERACCESS	<p>Indicates whether the interchange is presented to the IUSEREXIT program in virtual storage or in a file.</p> <p>M Gives the interchange to the exit in virtual storage. Applies when the IUSERTYPE is UE (user exit).</p> <p>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.</p>
IUSERTYPE	<p>The type of user exit program specified in IUSEREXIT.</p> <p>PG IUSEREXIT is a program that should be linked to (EXEC CICS LINK in CICS).</p> <p>UE IUSEREXIT is a DataInterchange user exit program defined in the ADAMCTL profile</p>
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 should be updated. This update is done only if the interchanges were not received using the Communication service receive function.

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.
TRXLIFE	<p>The amount of time this transaction should remain in the Transaction Store before it is eligible for purging. The default is 30 days.</p> <div>  <p>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.</p> </div>
IMGLIFE	<p>The length of time this transaction's image (the standard data produced) should remain in the Transaction Store. The default is 30 days.</p> <div>  <p>NOTES:</p> <ol style="list-style-type: none"> 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. </div>
HOLDFLAG	<p>Indicates whether this transaction should be placed on hold when added to the Transaction Store. A transaction in hold status is not available for any other activity until it is released.</p> <p>Y Holds the transaction.</p> <p>N or other Does not hold the transaction. N is the recommended value.</p> <div>  <p>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.</p> </div>
ERRFILTER	Specifies which error codes to filter out during this session. The values set here will be the initial values for the DIERRFILTER named variable at the start and end of each transaction. For more information, see "Error filtering" on page 21.

Field name	Description
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 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

Field name	Description
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

Field name	Description
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 396.

On the first call to the translator for a transaction, the next transaction in the file is located (this may involve finding the next interchange) and the transaction/message ID value is extracted from the transaction header. The translator attempts to locate a usage/rule and map. This search is done

even though a translation does not take place. The usage/rule indicates the type of functional acknowledgment requested, and the map identifies the segments and data elements in the requested transaction.

In functional acknowledgments, only mapped segments and data elements are validated. Your EDI administrator must establish the transactions and usages/rules before API requests are made. For an explanation of the fields used and the search done to locate a transaction, see “Locating sending and receiving trading partner profile members” on page 397. The deenveloping process still occurs without a transaction or usage/rule, but functional acknowledgments are not generated.

Numerous errors can occur at this point. For more information about the errors generated by the translator, see “Translation return codes” on page 680. 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 should 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 below are related to the transaction side of the processing rather than the application side.

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.

Field name	Description
	(other) The translator continued processing.
TSKEY	The key value assigned to the transaction in the Transaction Store. This is called the transaction handle and has a format of YYYYMMDDH-HMMSSxxxxnn. It is returned as a 10-byte packed value in this field, and stored in the database.
TSKEYU	The key value assigned to the transaction in the Transaction Store. The same value as the TSKEY field, but TSKEYU is an unpacked (character) 20-byte value.
ERRNUM	The total number of errors flagged during data processing.
ERRCDES	An array of the first 10 different errors flagged during data processing. See “Translator Error Codes” on page 614 for a list of errors.

The TRCB fields described below are related to the application side of the processing rather than the transaction side.

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 below are related to the functional acknowledgments.

Field name	Description
FABUILT	The envelope type for the functional acknowledgment. Valid values are: E UNB/UNZ. I ICS/ICE. T STX/END. U BG/EG. X ISA/IEA. G The acknowledgment does not have an interchange header. S The acknowledgment was written to the Transaction Store, but was not enveloped.
FARC	The return code from the translator for the functional acknowledgment translation done during deenvelope. For more information, refer to <i>DataInterchange Messages and Codes</i> .
FAERC	The extended return code from the translator for the functional acknowledgment translation done during deenvelope. For more information, refer to <i>DataInterchange Messages and Codes</i> .

The TRCB service segment fields (TF) described below 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.

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.
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).

Field name	Description
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 below change as transactions are processed to indicate how much of the interchange has been processed.

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 399. 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 below 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.

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).
GREL	The group release (group header for data type LV).
GRESAGENCY	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 below change as transactions are processed to indicate how much of the group has been processed.

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 400. 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 below are associated with the transaction header and trailer for the current transaction.

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 400. Y A transaction header is available. (other) A transaction header is not available.
LASTINENV	Indicates whether this transaction is the last transaction in the current interchange. Y Ends the interchange (other) Does not end the interchange
THCTL	See THXCTL .
THXCTL	The transaction control number assigned to the current transaction. The value returned in this field is right-justified with leading zeros.
SEGTRN	The total number of segments in the current transaction stored as a 4-byte binary value. See TST for a character representation of this value.
TTC	The current transaction or message ID value. See also the TRNID field on page 345.
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 .

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 should 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.

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.

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, then 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.

For application data:	For C & D records:	Then:
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

DataInterchange 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 DataInterchange 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, then 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, DataInterchange 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 should 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 DataInterchange 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 436.

The basic format of the COMMIT request is:

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

The unique parameters for this API request are:

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:

Parameter	Description
SNB	The same SNB used in previous requests. No further initialization is required.
FCB	ZFCBFUNC 1

Parameter	Description
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 **NEWGRP** 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. DataInterchange 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:

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

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:

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

The unique parameters for this API request are:

Parameter	Description
SNB	ZSNBNAME TRANPROC ZSNBPC 6
FCB	The function control block with one of the following values in ZFCBFUNC : 216 Detailed transaction data 217 Transaction image 218 Transaction acknowledgment image 219 Functional acknowledgment image
TRCB	The translator control block. See “Translator Control Block (TRCB)” on page 586 for details on this control block.
TRIDB	The translator input data block. See “Translator Input Data Block (TRIDB)” on page 616 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 618.

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 for data extraction

Field name	Description
ZSNBLL	32
ZSNBNAME	TRANPROC
ZSNBPC	6


FCB initialization for data extraction

Field name	Description
ZFCBLL	4
ZFCBFUNC	216, 217, 218, or 219

TRCB initialization for data extraction

Field name	Description
BLKLEN	1536
BLKNME	EDITRCB
BLKTYPE	The format for the TRIDB and TRODB buffers
	H Unlimited size
	(other) Limited to 32768 bytes

TRIDB initialization for data extraction

Field name	Description
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 for data extraction

Field name	Description
BLKLEN	The size of the data block, including this BLKLEN field. The minimum value for this field is 32000 bytes.
RESERVED	Binary zeros.

Retrieve detailed data API

This API function allows the application to retrieve detailed information concerning a transaction from the Transaction Store and returns it as formatted records in the TRODB. For more information, see “Transaction Store data extract information categories” on page 280.

The TRCB fields described below are required to initialize transactions.

Field name	Description
TSKEY	The transaction handle for the target transaction. The handle is <i>YYYYM-MDDHHMMSSxxnnnn</i> , 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 <i>YYYYM-MDDHHMMSSxxnnnn</i> , 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: <div> Y Concatenate detailed information records in the TRODB. N Return detailed information records individually. </div>
EJECT	Blank.

Once the initialization is complete, the data extraction service is invoked with the following API request:

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

Complete information about the transaction identified by the **TSKEY** or **TSKEYU** field is retrieved from the Transaction Store and formatted into the data extract record formats. For more information, see “Transaction Store data extract record formats” on page 282. The data extraction records are returned in the TRODB data block in the following sequence:

1. Interchange record
2. Group record, which can be followed by:
 - Functional acknowledgment interchange record
 - Functional acknowledgment group record
 - Functional acknowledgment transaction record
3. Transaction record, which can be followed by:
 - Transaction acknowledgment interchange record
 - Transaction acknowledgment group record
 - Transaction acknowledgment transaction record
4. Application record

When an outbound transaction is enveloped more than once, the preceding sequence is repeated for each interchange to which the transaction belongs.

When an inbound transaction is translated more than once, more than one application record is returned for each translation.

If there is a functional acknowledgment for the group, the interchange, group, and transaction records for the functional acknowledgment are returned with the group record even if concatenation has not been requested.

If there is a transaction acknowledgment for the transaction, the interchange, group, and transaction records for the transaction acknowledgment are returned with the transaction record even if concatenation has not been requested.

Numerous errors can occur at this point. These errors are described in “Translation return codes” on page 680. 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 should check the **TRXACCEPT** and **TRABORT** fields for error information.

When the translation was acceptable (**TRXACCEPT** value of **Y**), and:

If CONCATENATE is: Then:

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 following TRCB fields are returned in response to the first request for detailed information about a transaction.

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: <table> <tr> <td>X</td><td>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.</td></tr> <tr> <td>N</td><td>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.</td></tr> <tr> <td>Y</td><td>The data returned on this request was the last data for the transaction.</td></tr> </table>	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.
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.

TRCB	Description
TRODB	The translator output data block.
DATALEN	The amount of data returned in the DATA field.
DATA	The formatted data extraction records. For more information, see “Transaction Store data extract information categories” on page 280.

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 acknowledgment image API

This API function allows your application to retrieve the transaction acknowledgment image for the current transaction returned from the Retrieve detailed data API. You must request the current transaction (function code **216**) before requesting an image (function code **217**). Once the transaction detail record is returned, make the following API request:

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

A transaction acknowledgment image cannot be generated until the transaction detail record has been generated. The transaction image detail record is returned in the TRODB. If the image is too large to fit in the TRODB, the **EJECT** field of the TRCB has a value of **X**. In this case, make repeated requests (leaving the **EJECT** value as **X**) until the **EJECT** value changes to **Y**, indicating that no data remains.

Retrieve functional acknowledgment image API

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

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

A functional acknowledgment cannot be generated until the transaction detail record has been generated. The transaction image detail record is returned in the TRODB. If the image is too large to fit in the TRODB, the **EJECT** field of the TRCB has a value of **X**. In this case, make repeated requests (leaving the **EJECT** value as **X**) until the **EJECT** value changes to **Y**, indicating that no data remains.

Communication services

The functions provided in the Communication API enable an application program to send or receive transaction data, files, and messages to and from trading partners. This section provides some general comments about data flow and the components of the Communication services, followed by detailed data about each function.

The transaction or file data that is to be 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 368 and “Translate-to-standard API” on page 311.
- Using the ENVELOPE, REENVELOPE, or combination commands provided by the DataInterchange Utility.
- Using the enveloping and reenveloping functions provided by the Transaction Store facility. If transactions are being received from a trading partner, the communication functions ask the appropriate network programs to collect the data in the specified file. Management reporting is called at this time to record the number of interchanges and the total number of bytes received. However, responses are generated from the transactions in the file in several ways, including:
- Writing an API program to request that the transactions in the file be deenveloped, or deenveloped and translated. For more information, see “Deenvelope transaction” on page 389 and “Translate-file-to-application API” on page 350.
- Using the DEENVELOPE or combination commands provided by the DataInterchange Utility.
- Using the deenvelope functions provided by the Transaction Store facility.

In some cases an application program indirectly requests a communications API function from DataInterchange by going through a network program which processes the request with DataInterchange. Not all the programs involved in processing the request are written by or provided with DataInterchange. 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. DataInterchange invokes the communication component of DataInterchange (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 DataInterchange and your network, because it knows the interfaces on both ends and uses data provided by DataInterchange to build the commands required by your network program. DataInterchange provides communication routines for the networks directly supported by DataInterchange.
 - The network program. Network programs are not provided by DataInterchange, 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. DataInterchange provides message handlers for the networks directly supported by DataInterchange. However, anyone can write a message handler for an internal network or for one not directly supported by DataInterchange.
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 DataInterchange utilities and facilities use the communications API functions described in this section.

Communications service functions

The table below contains an overview of the functions provided by the communications service. The logical name for the communications service is COMM. This service is described in “Communication services” on page 407.

Function	Code	Sample Call Statement for Function
Queue standard data	110	FXXZccc (SNB , CCB , FCB , CMCB , TPPDB , DATABLK)
Send transactions	211	FXXZccc (SNB , CCB , FCB , CMCB , TPPDB)
Send files	221	FXXZccc (SNB , CCB , FCB , CMCB , TPPDB)
Receive	232	FXXZccc (SNB , CCB , FCB , CMCB , TPPDB)
Cancel previously sent data	233	FXXZccc (SNB , CCB , FCB , CMCB , TPPDB)
Return file name	300	FXXZccc (SNB , CCB , FCB , CMCB , TPPDB)
Process network acknowledgments	252	FXXZccc (SNB , CCB , FCB , CMCB , TPPDB)

Trading partner profile data block (TPPDB)

The trading partner profile data block (TPPDB) is a required parameters for all functions requested from the Communication API. This does not mean that an application making an API request must populate each field in the block with data. In fact, the only field in the TPPDB that must be initialized is the **BLKLEN** field.

This block is intended as an output block, but certain fields in the block will accept data. These fields are used by the communication routine to build commands for the network program and contain the parameters that control or direct the network program. If your program does not provide values for these fields, corresponding values retrieved by the communications routine from the mailbox (requestor) profile entry or the trading partner profile entry are used. The Communication API determines if you are providing values in this block by looking at the **TPNICKNM** field. If this field contains blanks, the TPPDB is interpreted as the output block. If this field is not blank, the TPPDB is interpreted as the input block and the values specified in the TPPDB override the values from the mailbox (requestor) profile member. The TPPDB fields you can specify are described in the table below.

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.

Field name	Description
NETID	The ID of a network profile (NETPROF) entry that defines the network to which the request should 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.
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.
NETCMD5	The name of a member of the PDS that is allocated to EDINTCMD which contains network commands that DataInterchange should 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 DataInterchange (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 (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.

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.
FAACK	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. For more information about return codes, see "Communication return codes" on page 691. In general, the **ZCCBRC** field has the following values:

Return code	Explanation		
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: <table> <tr> <td>1</td><td>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.</td></tr> </table>	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.
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.		

Return code	Explanation
	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, DataInterchange 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 DataInterchange 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 MVS, if an error causes network processing to restart during a send operation, you can use the Restart send transactions API to restart and complete the send. The Restart send transactions request is not supported in CICS. For more information on checkpoint recovery, refer to the *Expedite Base/MVS Programming Guide*.



1. To use the Restart send transactions API, you must initialize the control blocks with the same values as on 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 429). The DataInterchange 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 DataInterchange 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:

Parameter	Description
SNB	ZSNBNAME COMM ZSNBPC 5
FCB	ZFCBFUNC 211 for send transactions or 260 for restart send transactions
CMCB	The communications control block. For more information, see “Communication Control Block (CMCB)” on page 621.
TPPDB	The trading partner profile data block. For more information, see “Trading partner profile data block (TPPDB)” on page 409.

CMCB initialization for sending transaction data

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

Field name	Initialization
BLKLEN	254 (the length of the CMCB).
BLKNME	Any 8-character name. The recommended value is EDICMCB .
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 415.
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 DataInterchange 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

Field name	Initialization
ACCTYP	Account type. Applies only if the SENDFILE network command is being used. This field should contain a value of D to indicate that a trading partner account number and user ID are being supplied.
DATATYP	<p>The type of file name supplied in the FILENAME field. Valid values are:</p> <p>A Data set name</p> <p>D ddname</p> <p>If FILENAME is not provided on input, both FILENAME and DATATYP are provided on output.</p> <p> NOTES:</p> <ol style="list-style-type: none"> 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 DataInterchange can build the proper IEBASE INMSG FILEID parameter. If FILENAME is left blank, a D is still required in this field.
FILENAME	<p>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, DataInterchange assigns a value using the following rules:</p> <ul style="list-style-type: none"> • 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. <p>If FILENAME is not provided on input, both FILENAME and DATATYP are provided on output.</p> <p>NOTES:</p> <ol style="list-style-type: none"> 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 DataInterchange can build the proper IEBASE INMSG FILEID parameter. If this field is left blank, a D is still required in the DATATYP field.
DCIND	<p>The delivery class for the data being sent. This value is not interpreted by DataInterchange but is handled and understood by the network program. Valid values are:</p> <p>blank Normal delivery</p> <p>P High priority</p>

Field name	Initialization
ACKIND	The type of acknowledgment wanted. See the definition of this field in “Communication Control Block (CMCB)” on page 621 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 416 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 417 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 should be built for the network program to process. DataInterchange and the DataInterchange 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. These error conditions are explained in “Translation return codes” on page 680.

The CMCB fields described below are returned on a request to send transactions.

Field name	Description
SEQNUM	The network sequential number assigned to this transmission. The number is maintained in the Network sequence field of the network profile entry and is incremented on each request to send data. This number might be important later if you want to recall (cancel) this transmission.
ENAME	If a value was not supplied as input, the value from the Message user class field in the mailbox (requestor) profile entry is returned.
ACKIND	If a value was not supplied as input, the value from the Net acknowledgment field of the mailbox (requestor) profile or the Net acknowledgment field of the trading partner profile is returned.

Field name	Description
DATATYP	<p>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.</p> <p> NOTES:</p> <ol style="list-style-type: none"> 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	<p>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.</p> <p> NOTES:</p> <ol style="list-style-type: none"> 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.
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:

**If no envelope type
is supplied for:**

The message user class is:

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 should 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:

**If no envelope type
is supplied for:**

The message user class is:

ISA or ICS

The last 8 bytes of the interchange control number

GS

The group control number, left-justified and padded with blanks

BG, STX, or UNB

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 DataInterchange 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 CMCB settings for this request are described below. The TPPDB is described on page 409. The unique SNB and FCB parameters for the send files API request are:

Parameter	Description
SNB	ZSNBNAME COMM
	ZSNBPC 5
FCB	ZFCBFUNC 221

CMCB initialization

Some CMCB fields must be initialized before the API request is made. The following table describes the fields and the initialization requirements.

Field name	Initialization
BLKLEN	254 (the length of the CMCB).
BLKNME	EDICMCB .
TPNICKNM	The trading partner nickname to which the file should 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 DataInterchange 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.

Field name	Initialization
DCIND	The delivery class for the data being sent. The value entered in this field is not interpreted by DataInterchange, but is handled and understood by the network program. Valid values are: blank Normal delivery P High priority I Express delivery
ACKIND	The type of acknowledgment wanted. See “Communication Control Block (CMCB)” on page 621 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. These error conditions are explained in “Translation return codes” on page 680.

The CMCB fields described below are returned on a send file API request.

Field name	Initialization
SEQNUM	The network sequential number assigned to this transmission. The number is maintained in the Network sequence field of the network profile and is incremented on each request to send data. This number might be important later if you want to RECALL (CANCEL) this transmission.
ENAME	If a value was not supplied as input, the value in the Message user class field of the mailbox (requestor) profile is returned.
ACKIND	If a value is not supplied as input, the value from the Net acknowledgment field of the mailbox (requestor) profile or the trading partner profile is returned.
NPSSCDE	The start session response code returned by the message handler associated with the network. The value is extracted from the network program response records. This might not apply to all networks.

Field name	Initialization
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.

Receive and restart receive API

The DataInterchange 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 MVS, if an error causes network processing to restart during a send operation, you can use the Restart send transactions API to restart and complete the send. The Restart send transactions request is not supported in CICS. For more information on checkpoint-level recovery, refer to the *Expedite Base/MVS Programming Guide*.

The Restart receive transactions request applies only when your network supports restart and you specified checkpoint level recovery when initially receiving the data. For MVS, if an error causes network processing to restart during a receive operation, you can use the Restart receive transactions API to restart and complete the receive. The Restart receive transactions request is not supported in CICS. For more information on checkpoint-level recovery, refer to the *Expedite Base/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 DataInterchange Utility uses the Restart receive API when the RESTART RECEIVE command is requested.

The format of the receive API request is:

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

The CMCB settings for this request are described below. The TPPDB is described on page 409. The unique SNB and FCB parameters for the receive and Restart receive API request are:

Parameter	Description
SNB	ZSNBNAME COMM ZSNBPC 5
FCB	ZFCBFUNC 232 for receive 261 for restart receive

CMCB initialization for receive requests

Some CMCB fields must be initialized before the receive API request is made. The following table describes the fields and initialization requirements.

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 422.
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 should contain a blank.
DATATYP	For MVS, 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: <div style="display: flex; justify-content: space-between; padding: 0 20px;"> A Data set name </div> <div style="display: flex; justify-content: space-between; padding: 0 20px;"> D ddname </div> <p>If FILENAME is not provided on input, both FILENAME and DATATYP are provided on output.</p>

CMCB	Initialization
FILENAME	The name of a file for receiving the data. For MVS, 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, DataInterchange 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 RCVFILE 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.
RCVTTY	Indicates whether only the first file that meets the selection criteria should be returned. G Returns only the first file that meets the selection criteria blank Returns all files meeting the criteria
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 should be built for the network program to process. DataInterchange and the DataInterchange utilities and facilities set the network operator based on the type of data contained in the file.



Two network commands are available: **RCVEDI** and **RCVSTREAM**. Use **RCVEDI** when you want the network program to parse the interchange headers in the file. Use **RCVSTREAM** 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. These error conditions are explained in "Translation return codes" on page 680.

The CMCB fields described below are returned on receive data API request.

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.
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, then, 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.
FILERCV	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.

Field name	Description
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 DataInterchange Utility supports the Cancel API.

The basic format of the Cancel API request is:

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

The CMCB settings for this request are described below. The TPPDB is described on page 409. The unique SNB and FCB parameters for the Cancel API request are:

Parameter	Description
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. The following table describes the fields and initialization requirements.

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 .

Field name	Initialization
REQID	The mailbox (requestor) profile ID value used for this request. The mailbox (requestor) profile identifies the mailbox that you want to use to identify yourself. Numerous values are taken from this profile entry while building the commands for the network.
ACCTYP	D . A trading partner account number and user ID is being supplied.
ACKIND	The type of acknowledgment wanted regarding the cancellation. See “Communication Control Block (CMCB)” on page 621 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.
CANS D	The cancellation start date in <i>YYMMDD</i> format. Initialize with blanks if not used.
CANST	The cancellation start time in <i>HHMMSS</i> format. Initialize with blanks if not used.
CANED	The cancellation end date in <i>YYMMDD</i> format. Initialize with blanks if not used.
CANET	The cancellation end time in <i>HHMMSS</i> format. Initialize with blanks if not used.
TMZONE	The time zone used for CANS D, 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. These error conditions are explained in “Translation return codes” on page 680.

The CMCB fields described below are returned from a Cancel request. The values are returned by the message handler associated with the network. The values are extracted from the network program response records. This might not apply to all networks.

CMCB	Initialization
NPSSCDE	The start session response code.
NPESCDE	The end session response code.

CMCB	Initialization
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 CMCB settings for this request are described below. The TPPDB is described on page 409. The unique SNB and FCB parameters for the Return filename API request are:

Parameter	Description
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. The following table describes the fields and initialization requirements.

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. These error conditions are explained in “Translation return codes” on page 680.

The CMCB fields described below are returned on a Return filename request.

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 DataInterchange 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 380. For information about how to provide the name of the file where the transaction data should be written, see “Translate-file-to-application API” on page 350 and “Envelope API” on page 368.

Process network acknowledgments API

The Process network acknowledgments request is used to build all the commands needed to instruct the network program to return network acknowledgment data. The DataInterchange Utility uses this API when the UPDATE STATUS command is issued.

The basic format of the Process network acknowledgments API request is:


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

The CMCB settings for this request are described below. The TPPDB is described on page 409. The unique SNB and FCB parameters for the Process network acknowledgments API request are:

Parameter	Description
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 below.

Field name	Initialization
BLKLEN	254 (the length of the CMCB).
BLKNME	EDICMCB .
NETID	The network that is invoked. This field is ignored if REQID has a value.
	NOTE: This function must be invoked for each network individually. The UPDATE STATUS command issues a request for every network defined in the network profile unless you use the NETID keyword to request a specific network.
NETOP	If a file of network acknowledgments has already been received and you want to process it, use a NETOP value of NO-NETOP . Otherwise, leave the field blank and the communication routine will invoke the network program to return the acknowledgment data.
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. For more information, see “Communication return codes” on page 691.

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.

Update status service overview

The table below describes the functions provided by the update status service. The logical name for the update status service is TRANSERV.

The basic format of the Process network acknowledgments API request is:

```
FXXZC(SNB,CCB,FCB,USKB###,status[,USDB[,REQID]])
```

where ### is the function code.

Update status using:	Function Code
The envelope full key with trading partner identified by the account number and user ID.	210
The transaction handle value for one of the transactions in the interchange.	211
An alternate key with the trading partner identified by the account number and user ID.	212
The full envelope key with the trading partner identified by the interchange qualifier and interchange ID.	213
An alternate key with the trading partner identified by the interchange qualifier and interchange ID.	214
The full envelope key with the trading partner identified by the trading partner nickname.	215

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:

```
FXZZccc(SNB,CCB,FCB,USKB,status[,USDB[, 'reqid']])
```

The unique parameters for the Update status API request are:

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 432.

Parameter	Description																								
STATUS	The new status for the interchange. Valid values are: <table> <tr><td>41</td><td>Send error</td></tr> <tr><td>42</td><td>Send request error</td></tr> <tr><td>43</td><td>Not sent, network error</td></tr> <tr><td>46</td><td>Send started</td></tr> <tr><td>48</td><td>Send requested</td></tr> <tr><td>49</td><td>Sent to network</td></tr> <tr><td>50</td><td>Accepted by the network</td></tr> <tr><td>51</td><td>Delivered by the network</td></tr> <tr><td>52</td><td>Purged by the network</td></tr> <tr><td>53</td><td>Recall requested from network</td></tr> <tr><td>54</td><td>Recall request error</td></tr> <tr><td>55</td><td>Recalled from network</td></tr> </table>	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
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 435).																								
REQID	If you specify this parameter, the management reporting component of DataInterchange 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:

Return Code	Explanation						
0	Status update was successful.						
8	Status update failed. Extended return codes are: <table> <tr><td>210</td><td>No interchange was found.</td></tr> <tr><td>211</td><td>Internal program error.</td></tr> <tr><td>212</td><td>An error occurred when attempting to update the database.</td></tr> </table>	210	No interchange was found.	211	Internal program error.	212	An error occurred when attempting to update the database.
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

The table below describes the format for a full key using a trading partner nickname. Use this update status key block when **ZFCBFUNC** contains **215**.

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. Should 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

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

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

The table below describes the format for a full key using an account number and user ID. Use this update status key block when **ZFCBFUNC** contains **210**.

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**.

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 YYMMDD-DHHMMSSxxxxnnnn 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 435) 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 - account number and user ID

The alternate key format described below uses the account number and user ID. Use this update status key block when **ZFCBFUNC** contains **212**.

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.
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 - interchange qualifier and ID

The alternate key format described below uses the interchange qualifier and ID. Use this update status key block when **ZFCBFUNC** contains **214**.

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.

Name	Offset	Length	Format	Description
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.

The table below describes the fields used to build an alternate key that can be used later to retrieve and update the status of this interchange.

Name	Offset	Size	Format	Description
MSGCLASS	0	8	Char	The message user class assigned to the interchange.
MSGNAME	8	8	Char	The message name assigned to the interchange.
SEQNUM	16	5	Char	The message sequence number assigned to the interchange.
UNIQUEID	21	8	Char	The unique message ID assigned to the interchange by the network. If this field is not blank, it must contain a value that is unique for all interchanges currently in the Transaction Store.
SENDDATE	29	8	CHAR	The date the interchange was sent (YYYYMMDD).
SENDTIME	37	6	CHAR	The time the interchange was sent (HHMMSS).
NETACK	43	1	CHAR	The type of network acknowledgment expected for this interchange. See “Trading Partner Profile Block (TPPDB)” on page 632 for valid values.

SYNCPPOINT services

The descriptions of translation and enveloping services frequently refer to the recovery scope for the session (see “Send Recovery Scope:” on page 334). SYNCPPOINT 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 should 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 MVS/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 SYNCPPOINT 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 SYNCPPOINT ROLLBACK request. Again, CICS passes this request to DB2 and controls the removal of CICS resources with DB2 resources.

SYNCPPOINT 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 DataInterchange Utility, recovery scope is established by using the **RECOVERY** keyword on the PERFORM command.

You can use SYNCPPOINT services to lengthen the recovery scope by setting the **SYNCPPOINT** interval to a positive integer value greater than **1**. You can effectively turn off SYNCPPOINT service by setting the **SYNCPPOINT** interval to **-1**. In this case, the application program controls the **SYNCPPOINT** 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 SYNCPPOINT) request in the middle of the interchange, the chances of deadlock increase, and the consistency between the application databases and the DataInterchange Transaction Store is compromised.

The table below lists the functions provided by the SYNCPPOINT service. The logical name for the SYNCPPOINT service is SYNCSEV.

Function	Code	Sample Call Statement for Function
Initialize SYNCPPOINT 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 DataInterchange termination process from issuing a commit. To prevent this termination commit, issue a SYNCPPOINT service call with the **SYNCPPOINT** interval set to **-1** prior to the DataInterchange termination call. For more information, see “Send Recovery Scope:” on page 334.

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:

Return code	Description
-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 may 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, DataInterchange has the options to issue either:

- A ROLLBACK command
- A FETCH command for the trading partner control numbers again (TA)

The following table describes the DB2 recovery conditions and actions.

Row	Conditions True or False				Actions Yes or No	
	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
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:

Parameter	Description
SNB	ZSNBNAME SYNCSERV
	ZSNBPC 4
FCB	ZFCBFUNC 1
	INTERVAL Indicates the length of time that elapses between synchroni- zation points. A 4-byte binary value. Valid values are:
	0 General SYNCPOINT processing based on the recovery scope in effect.
	<i>n</i> A positive number indicating how often a SYNCPOINT should be taken. For example, if <i>n</i> 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 trans- action, but only every 5th request is honored by the SYNCPOINT service.
	-1 The application is controlling the SYNCPOINT interval, and all DataInterchange requests for COMMITs are ignored. Application control of the SYNCPOINT interval does not mean that COMMIT requests can be issued indiscriminately. For example, if the application requests an interchange recovery scope and then issues a COMMIT after each trans- action, the results are not as expected and the chance of deadlock increases dramatically. You can lengthen the DataInterchange recovery scope value (as in the preceding example where a COMMIT is issued after every 5th transaction), but shortening the recovery scope leads to problems with database integrity and deadlocks.

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:

Return Code	Explanation
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:

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:

Return Code	Explanation
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: <ul style="list-style-type: none"> • An interval of -1 was established on the SYNCPOINT initialization function. • The interval has not been reached yet.
12	COMMIT 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:

Parameter	Description
SNB	ZSNBNAME SYNCSERV ZSNBPC 3
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:

Return Code	Explanation
0	ROLLBACK work was successful.
12	ROLLBACK work failed. The extended return code is: N The SQLCODE from DB2 indicating the reason for 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 DataInterchange 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. DataInterchange 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 869 and “Get/Put envelope exit and service” on page 457.

The syntax of the Get envelope API request is:

```
FXXZASM (GPCB,CCB,FCB,BUFFER,LEN)
```

The parameters for this function request are:

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 DataInterchange. 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, DataInterchange 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. DataInterchange 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 871 and “Get/Put envelope exit and service” on page 457.

The syntax of the Put envelope API request is:

```
FXXZASM (GPCB,CCB,FCB,BUFFER,LEN)
```

The parameters for the Put envelope function request are:

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 DataInterchange. 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.

Exit routines

An exit routine is a program that you provide to perform some service for your application or data. DataInterchange 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 DataInterchange. For example, DataInterchange may 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 DataInterchange and can interact directly in the current DataInterchange 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 five user exits that your application programs can use to extend or enhance the capabilities of DataInterchange:

- Field exit routines provide additional processing for application data (translate-to-standard) and EDI standard data (translate-to-application) during translation.
- Pre-translation and post-translation exit routines 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 DataInterchange.

The first three instances are described in the following sections. Point-to-point network programs and message handlers are discussed in Chapter 7, “Interfacing to other networks and applications.”

The architecture for user extensions to DataInterchange is very similar to the architecture used when calling services using the DataInterchange 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 DataInterchange in the user program information profile (ADAMCTL). 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 *DataInterchange Administrator's Guide*.

Return codes

The tables in this chapter refer to the DataInterchange Utility Severity (UTILSEV) and Condition (UTILCCODE) codes. For MVS, the UTILCCODE code is returned as the Job Step Condition Code. For CICS, these codes are returned in the UTILCB. See Chapter 6, "Using DataInterchange in the CICS environment." UTILSEV refers to the UTILCB field, CCBRC, and UTILCCODE refers to the UTILCB field, CCBERC. The UTILCB fields CCBRC and CCBERC should not be confused with the CCB fields ZCCBRC and ZCCBERC.

Exit languages

A user exit may be written in Assembler, C, or COBOL. You must specify the implementation language for a user exit in the user program information profile (ADAMCTL) in the **Program language** field of the profile entry. Valid values are:

- | | |
|----------|---|
| A | Assembler programs. |
| C | C language programs. DataInterchange supports the System Application Architecture (SAA) C/370 compiler. |
| J | COBOL programs written using a COBOL compiler other than IBM COBOL II. |
| K | COBOL II programs. DataInterchange 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 DataInterchange determines that a user exit should be called, the logical name of the user exit is used to read the ADAMCTL profile entry to determine the physical load module name (**Load module name**) and the implementation language (**Program language**). DataInterchange issues an operating system load for the load module and then passes control to the exit routine. Information about the exit is retained by DataInterchange 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 may 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 may 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 may reside either above or below the 16 MB line.

The **Load module name** field in the ADAMCTL profile must match the name given to the load module in the linkage editor control statements. This is the name that DataInterchange 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 DataInterchange 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.

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 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 DataInterchange to ignore the data or to ignore the rules through return code settings. For more information, see “Send parameters” on page 447, and “Receive parameters” on page 448.
- Change the value of the data and tell DataInterchange to use the new value.
- Save the value of the data for use by other field exit routines.

A sample field exit routine is provided in “Field exit programs” on page 814.

Field exit routines shipped with DataInterchange

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 DataInterchange:

- 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 450 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 579 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 581 for a detailed description of this block. The **ZCCBRC** field is used to tell DataInterchange 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. DataInterchange 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 DataInterchange.

Field offset (4-byte binary value)

You can use this value to determine which field you are processing. However, DataInterchange 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. DataInterchange 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. DataInterchange 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 DataInterchange, 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 450 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 579 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 581 for a detailed description of this block. The **ZCCBRC** field is used to tell DataInterchange 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. DataInterchange creates a log record (message TR0006) indicating a user exit error occurred, which is treated as a level 1 (data element) error, and does not process the data element any further (same as a return code of 1). To set the functional acknowledgment code, see the temporary work area.

Data element value

The actual data element value from the segment. You can change the value directly in this buffer if you do not increase the length. If the value is changed directly in the buffer, however, and the data element is mapped more than once, subsequent mappings and exits may 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 DataInterchange.

Field offset (4-byte binary value)

You can use this value to determine which data element you are processing. However, DataInterchange 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. DataInterchange 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. DataInterchange initializes this work area with blanks before calling the exit routine. If the user exit found an error in the data and rejects the data by setting the CCB return code to a value greater

than 20, you can use the first byte of this work area to set the functional acknowledgment error code for the AK4 segment. The return value must be numeric, or it is ignored. Other than that, the return value is neither edited or validated. If 999 or CONTRL functional acknowledgments are being created, DataInterchange 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 DataInterchange, 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. The CCB and SNB are omitted here, but are provided in “Assembler SNB” on page 745 and “Assembler SNB” on page 745.

```
*
* 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    DSAAddress of the SNB
CCBDATA    DSAAddress of the CCB
FLDDATA    DSAAddress of data
FLDOFF     DSAAddress of 4 bytes containing offset
FLDLEN     DSAAddress of 4 bytes containing length
PERMAREA   DSAAddress of 4096 work area
TEMPAREA   DSAAddress of 1024 temporary area
TEMPLEN    DSAAddress of 4 bytes to return length value
*
USREXIT    CSECT
          USING PARMS,R1
```

C definition

The C definitions for parameters are shown below. The CCB and SNB are omitted here, but are provided in “C SNB” on page 734 and “C SNB” on page 734.

```
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*ccbddata;/* 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. The CCB and SNB are omitted here, but are provided in “COBOL CCB” on page 712 and “COBOL SNB” on page 712.

LINKAGE SECTION.

```

*****
*           ADF - APPLICATION DATA FORMAT  FIELD DATA      *
*****

01  FLD-DATA                      PIC X(100).

*****
*           OFFSET OF ADF 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 ADAMCTL profile.

Pre-translation exit

During translate-to-application operations, the translator calls a pre-translation routine before any translation takes place. The exit has access to the entire transaction (all data between but not including the transaction set header and transaction set trailer). The pre-translation exit routine can perform any operation on this data. Any modifications made by the exit routine become part of the transaction image. When modifying the data, observe the following restrictions:

- Do not change the length of the data.
- Do not change any character to make it look like a segment delimiter.

Post-translation exit

During translate-to-standard operations, the translator calls a post-translation routine after the translation is complete and before the transaction image is written to the Transaction Store. The exit program has access to the entire transaction (all the data between but not including the transaction set header and transaction set trailer). The post-translation exit routine can perform various modifications on this transaction. In modifying the data, observe the following restrictions:

- Do not change the length of the data.
- Do not change any segment delimiters.
- Do not change any non-segment delimiter character to a segment delimiter character. This restriction ensures that the receiving translator can verify that it is receiving the correct number of segments, specified by the transaction set trailer.
- Do not change the data to appear as though it is a transaction set trailer, group trailer, or interchange trailer.

Pre- and Post-translation exit parameters

The parameter list for pre-translation and post-translation exit routines consists of the following pointers.

Service name block (SNB)

See “Service Name Block (SNB)” on page 579 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 581 for a detailed description of this block. The **ZCCBRC** field is used to tell DataInterchange 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. DataInterchange 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. DataInterchange 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. DataInterchange 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. The CCB and SNB are omitted here, but are provided in “COBOL CCB” on page 712 and “COBOL SNB” on page 712.

```
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. The CCB and SNB are omitted here, but are provided in “C SNB” on page 734 and “C SNB” on page 734.

```
typedef struct WORKAREA worka;
struct WORKAREA {
    char    working-4096-;
};

main(snbdata,
     ccbdata,
     trxdata,
     fldcomp,
     trxlength,
     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   *fldcopatl;  /* Compatibility field */
```

Assembler definition

The Assembler definitions for parameters are shown below. The CCB and SNB are omitted here but are provided in “Assembler SNB” on page 745 and “Assembler SNB” on page 745.

```
*
* 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

DataInterchange 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 441 and “Put envelope service” on page 441.

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 483 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 DataInterchange from a user-written API program. DataInterchange 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, DataInterchange calls the user exit. A pointer field (**IUSERAREA**) is provided in the TRCB to allow the user's API program to pass a user-defined area to the user exit. This user-defined area may 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 DataInterchange 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), DataInterchange 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 DataInterchange from a user-written API program. DataInterchange 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, DataInterchange calls the user exit. A pointer field (**IUSERAREA**) is provided in the TRCB to allow the user's API program to pass a user-defined area to the user exit. This user-defined area may 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 DataInterchange 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), DataInterchange deenvelopes the data that was stored by the user exit. On any unsuccessful return (CCB return code not zero), a message (TR1255 or TR1256) is logged with the return code, processing terminates, and the return code is returned to the API program.

FXXZccc stub program

When DataInterchange 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 DataInterchange language-dependent stub (FXXZccc). The format of the stub is:

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

The parameters for the FXXZccc stub are:

Parameter	Description
SNB	The service name block that DataInterchange passes as input to the exit routine.
CCB	The common control block that DataInterchange passes as input to the exit routine.
FCB	The function control block that DataInterchange passes as input to the exit routine. This block is initialized by DataInterchange 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 DataInterchange passes as input to the exit. This block was initialized by DataInterchange 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. DataInterchange 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 DataInterchange 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 FXXZccc stub are:

Parameter	Description
GPCB	The Get/Put control block that DataInterchange passed to the exit. This block was initialized by DataInterchange with values necessary to call the Get or Put service.
CCB	The common control block that DataInterchange passed to the exit routine.

- FCB** The function control block that DataInterchange passed to the exit routine. This block was initialized by DataInterchange 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 DataInterchange should place the envelope on a Get function or the address from which DataInterchange 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).

These return codes: Indicate:

- | | |
|-----------------|---|
| 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).

These return codes: Indicate:

- | | |
|------------------|---|
| RC = 8, ERC = 1 | The function code in the FCB is invalid. |
| RC = 12, ERC = 2 | A virtual storage failure occurred during the Put function. |
| RC = 0, ERC = 0 | The envelope data was stored. |

Security routines

Security exit routines protect transaction data through encryption and authentication. Encryption protects data against unauthorized viewing. Authentication protects data against unauthorized changes.

Filtering and compression are also defined as part of the encryption and authentication architecture. Filtering ensures that data does not contain characters that could conflict with the control characters used in transmitting an interchange. Filtering is only necessary if the network used to send the data has restrictions on the characters that can be transmitted. Compression can be used to decrease the amount of data that is being transmitted, which results in more efficient transmission and storage of the interchange. If a network is sensitive to the data that is transmitted, a filtering routine must be used to ensure there are no conflicting characters.

You need a way to communicate with your trading partner that data you are sending requires security processing, and a way for your trading partner to tell you that data being sent to you requires security processing. ANSI has defined how this communication takes place by defining security segments that are placed within the interchange to signal that security processing is required and the exact nature of that processing. The security segments consist of a security header to flag the start of security processing and a security trailer segment to flag the end of security processing. ANSI has also defined that security processing can only occur at specific points within an interchange.

An entire functional group can require security processing and is identified by the S1S security header segment and the S1E security trailer segment.

A transaction can require security processing and is identified by the S2S security header segment and the S2E security trailer segment.



NOTE: It is possible for a transaction to be secured using the S2S and S2E segments and to exist within a functional group that has been secured with the S1S and S1E segments. Group and transaction security are independent.

The order of security processing has also been standardized. When data is being prepared for sending (ENVELOPE function), the order is:

1. Authentication
2. Compression
3. Encryption
4. Filtration

When data is being received that requires security processing (DEENVELOPE function), the order is:

1. Filtration
2. Decryption
3. Decompression
4. Authentication

A transaction image is always stored in the Transaction Store in clear text. Security processing takes place during the enveloping and deenveloping processes. During the envelope process:

- A transaction image is retrieved from the Transaction Store.
- Security processing takes place against the image.
- The secured image within an interchange is written to the file associated to the network and the interchange is delivered to the trading partner by the network.

During the deenvelope process:

- The secured interchange is read from the file.
- Security processing takes place against the image.
- The transaction image (now in clear text) is written to the Transaction Store and future translations for the transaction do not require security processing.

The following section describes how security processing is enabled for both the send and receive side, and is followed by detailed descriptions of the interface to the user exit routines that provide the encryption, authentication, compression, and filtration functions.



NOTE: DataInterchange provides encryption, authentication, and filtration routines that may be used if they suit your needs. Refer to the network security profile (SECUPROF) description in the *DataInterchange Administrator's 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, then during the enveloping process, the group in which this transaction is placed has security processing. S1S and S1E segments are built by DataInterchange 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, then during the enveloping process, this transaction will have security processing. S2S and S2E segments are built by DataInterchange 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, DataInterchange 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. DataInterchange 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, DataInterchange 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 481.

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 DataInterchange. 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, DataInterchange 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, DataInterchange 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 DataInterchange. For more information, see “Put data routine” on page 480. 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. The COBOL, C, and Assembler definitions for this block are in “COBOL SPDB” on page 720, “C SPDB” on page 742 and “Assembler SPDB” on page 753.

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 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 DataInterchange using a Put data routine. For more information, see “Put data routine” on page 480.

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 479.

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 849.

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 450, “C definition” on page 450, and “COBOL definition” on page 451.

Service name block (SNB)

See “Service Name Block (SNB)” on page 579 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 581 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, DataInterchange 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 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 584 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 479) and to put results (“Put data routine” on page 480). 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 479). 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. The SNB, CCB, FCB, and SPDB are omitted here but are provided in “COBOL CCB” on page 712, “COBOL SNB” on page 712, “COBOL FCB” on page 713, and “COBOL SPDB” on page 720.

```

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. The SNB, CCB, FCB, and SPDB are omitted here but are provided in “C SNB” on page 734, “C SNB” on page 734, “C FCB” on page 735, and “C SPDB” on page 742.

```

main(snbdata,
    ccldata,
    fcldata,
    handle,
    keyname,
    secdata,
    bufsize,
    inbuf,
    outbuf,
    datalen,
    residual,
    vector)

snb    *snbdata; /* Service name block pointer set up by DI */
ccb    *ccldata; /* Common block pointer used by DI          */
fcb    *fcldata; /* 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. The SNB, CCB, FCB, and SPDB are omitted here but are provided in “Assembler SNB” on page 745, “Assembler SNB” on page 745, “Assembler FCB” on page 746, and “Assembler SPDB” on page 753.

```
*
* 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 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 479. A sample authentication routine is provided in “Authentication examples” on page 840.

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 450, “C definition” on page 450, and “COBOL definition” on page 451. The parameter list consists of the following pointers:

Service name block (SNB)

See “Service Name Block (SNB)” on page 579 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 581 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, DataInterchange 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 ADAMCTL profile.
5 - 10	Reserved.
11	The authentication key name is not known.
12 - 20	Reserved.
21 and higher	The errors were defined by the exit routine.

Function control block (FCB)

See “Function control block (FCB)” on page 584 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 479). 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 479). 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. The SNB, CCB, FCB, and SPDB are omitted here but are provided in “COBOL CCB” on page 712, “COBOL SNB” on page 712, “COBOL FCB” on page 713, and “COBOL SPDB” on page 720.

```

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. The SNB, CCB, FCB, and SPDB are omitted here but are provided in “C SNB” on page 734, “C SNB” on page 734, “C FCB” on page 735, and “C SPDB” on page 742.

```

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. The SNB, CCB, FCB, and SPDB are omitted here but are provided in “Assembler SNB” on page 745, “Assembler SNB” on page 745, “Assembler FCB” on page 746, and “Assembler SPDB” on page 753.

```

*
* 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 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 DataInterchange using a Put data routine. For more information, see “Put data routine” on page 480.

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 479.

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 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 DataInterchange using a Put data routine. For more information, see “Put data routine” on page 480.

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 479.

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 468, “**C definition**” on page 467, and “**COBOL definition**” on page 466

Service name block (SNB)

See “Service Name Block (SNB)” on page 579 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 581 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, DataInterchange 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 ADAMCTL profile.
5 - 20	Reserved.
21 and higher	The errors were defined by the exit routine.

Function control block (FCB)

See “Function control block (FCB)” on page 584 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 479) and to put results (“Put data routine” on page 480). 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 may be used to hold the processed data. The Put data routine must be used to communicate the results back to DataInterchange.

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 479). A 4-byte binary value.

Filtering exit language definitions**COBOL definition**

The COBOL definitions for the filtering parameters are shown below. The SNB, CCB, FCB, and SPDB are omitted here but are provided in “COBOL CCB” on page 712, “COBOL SNB” on page 712, “COBOL FCB” on page 713, and “COBOL SPDB” on page 720.

```
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. The SNB, CCB, FCB, and SPDB are omitted here but are provided in “C SNB” on page 734, “C SNB” on page 734, “C FCB” on page 735, and “C SPDB” on page 742.

```

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. The SNB, CCB, FCB, and SPDB are omitted here but are provided in “Assembler SNB” on page 745, “Assembler SNB” on page 745, “Assembler FCB” on page 746, and “Assembler SPDB” on page 753.

```

*
* 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 DataInterchange administrator does not specify a buffer size, DataInterchange obtains a buffer that is large enough to hold the data; however, the buffer cannot exceed 32 K.

When DataInterchange calls a security exit routine, the values that DataInterchange 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 DataInterchange language-dependent stub (FXXZccc). The format of the stub is:

```
FXXZccc(handle,ccb,fcb,buffer,length)
```

The parameters of the FXXZccc stub are:

Parameter	Description
handle	The authentication, encryption, compression, or filtering handle that DataInterchange passes as input to the calling routine.
ccb	The common control block that DataInterchange 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 DataInterchange should place the residual data. This address can be the same address that DataInterchange passes to the calling routine.
length	The address of a 4-byte field that contains the maximum number of bytes of residual data that DataInterchange 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):

These return codes	Indicate:
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. DataInterchange 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 DataInterchange 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 DataInterchange language-dependent stub (FXXZccc). The format of the stub is:

```
FXXZccc(handle,ccb,fcb,buffer,length)
```

The parameters of the FXXZccc stub are:

Parameter	Description
handle	The encryption, compression, or filtering handle that DataInterchange passes as input to the calling routine.
ccb	The common control block that DataInterchange passes as input to the calling routine.
fcb	A function control block with a function value of 2 (put data request).
buffer	The address where DataInterchange should put the data. This address can be the same address as the output buffer that DataInterchange passes as input to the calling routine.
length	The address of a 4-byte field that specifies the maximum number of bytes that DataInterchange 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):

These return codes	Indicate:
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 may 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 DataInterchange language-dependent stub (FXXZccc). The format of the stub is:

```
FXXZccc(handle,ccb,fcbl,routname,0)
```

The parameters of the FXXZccc stub are:

handle	The encryption, compression, or filtering handle that DataInterchange passes as input to the calling routine.
ccb	The common control block that DataInterchange passes as input to the calling routine.
fcbl	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 DataInterchange administrator specifies in the 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):

These return codes: Indicate:

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 DataInterchange can invoke. There are response programs which only apply in the CICS environment (see Chapter 6, “Using DataInterchange in the CICS environment.”). 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 DataInterchange session. The program is not accessed using the service and exit architecture of DataInterchange but is linked using an MVS LINK macro or the EXEC CICS LINK facility. This being the case, there are no language restrictions imposed by DataInterchange for the implementation language of the program. However, in an MVS environment, the parameters passed to the routine can reside above the 16 MBb line and, therefore, the language must be capable of supporting 31-bit addressing mode (AMODE 31).

Data extract exit

When performing any of the data extract commands described in “Producing management reports from the Transaction Store” on page 9, you can use a program that gets control before the extraction record is written. This program may 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 DataInterchange will link before writing an output record.

In MVS, 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 581 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 following format:

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 DataInterchange.
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

DataInterchange 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 457.

This program is invoked through an MVS LINK macro in the MVS 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 MVS, 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 DataInterchange CCB.

In MVS, 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
```


Using DataInterchange in the CICS environment

Most functions of DataInterchange operate similarly regardless of the environment in which they are executed. However, before designing applications to use DataInterchange in the CICS environment, programmers should be aware of the differences that do occur. This chapter describes how to interface application programs with DataInterchange 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 DataInterchange with other networks and applications, see Chapter 7, “Interfacing to other networks and applications”.

Running the DataInterchange Utility in the CICS environment

The DataInterchange Utility is the central point of access to DataInterchange functions and the most commonly used interface. In CICS, when you develop an application program to run the DataInterchange Utility, you can write the application in any programming language supported by CICS.

The DataInterchange 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 DataInterchange Utility. For a description of the specific DataInterchange functions that can be accessed by the DataInterchange Utility, see Chapter 1, “Using The Datainterchange Utility.”

Invocation options

The following methods can be used to invoke the DataInterchange 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 DataInterchange Utility executes synchronously with your application. When the DataInterchange 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 497.
- **CICS START command**
Your application uses this command to start the CICS transaction EDIB. The DataInterchange 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 521.
- **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 DataInterchange 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 521 for more details.
- **EDIW CICS transaction**
You can use this CICS transaction to enter ad hoc PERFORM commands which, in turn, invoke the DataInterchange 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 DataInterchange Utility to process your request. See “DataInterchange Utility control information” on page 507 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 DataInterchange 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 DataInterchange 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 DataInterchange 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 “DataInterchange Utility control information” on page 507.

The results are passed as follows:

1. If you used the CICS LINK command and the control information was supplied to the DataInterchange 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 DataInterchange 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 DataInterchange 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 DataInterchange 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 521 for more details about response applications.

DataInterchange/MVS- CICS abend return codes

When an abend is detected, DataInterchange 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 DataInterchange Utility supports the use of MQSeries queues and two different types of CICS storage mechanisms for most of the sequential files it uses:

- Temporary Storage (TS) queues

TS queues are useful for files that are processed multiple times, or when recovery is not an issue. When TS queues are used as an output destination, such as a print file, exception file, and query file, they are cleared before they are used. The following three DataInterchange 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 DataInterchange Utility appends the transmitted data to the TS queue.

CICS restricts the size of a TS queue to 32-K records. Generally, DataInterchange processes only one envelope TS queue (either inbound or outbound) at a time. However, the DataInterchange 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 489.

- 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 DataInterchange 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 DataInterchange Utility are:
 - Extrapartition TD queues
Extrapartition TD queues are MVS sequential data sets. You can use them for output when further processing is required in MVS or use them to input data generated in MVS. Extrapartition TD queues are always appended by the DataInterchange Utility. These files are never cleared.
 - Intrapartition TD queues
If recovery is a high priority, recoverable intrapartition TD queues are the best choice. If recovery is not a high priority, non-recoverable intrapartition TD queues are a good choice.

The destructive read property of intrapartition TD queues is ideal in eliminating the potential for reprocessed data. If the queues are recoverable, the input queues being read are kept in synchronization with all modified resources. This helps clearly define what resources are part of the unit of work. Restart processing is much easier with recoverable TD queues because the queues are always at a complete unit of work boundary. These files are never cleared.
- MQSeries queues
MQSeries queues are specified using DataInterchange MQSeries Queue profile member names and a **File type** of **MQ**. MQSeries queues are very useful if your application receives data from, or passes data to, other applications which are not in your CICS region. These other applications can be MVS batch applications or applications running on other operating systems such as AIX. MQSeries allows cross-platform communications between applications. In order to use DataInterchange MQSeries support, you must have MQSeries installed and running on your CICS region.



ATTENTION: Be careful when selecting the storage mechanism for your application's interface with DataInterchange. Data can be inadvertently reprocessed or lost if the wrong storage mechanism is used.

CICS envelope queue alternatives

Normally, envelope files created by DataInterchange/MVS-CICS are TS queues. Whether DataInterchange 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 DataInterchange 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 869 and “Put Envelope service example” on page 871.

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 DataInterchange. The DFHCOMMAREA contains pointers to the following control blocks: UCB, TRCB, TPPDB, and CCB. When using the DataInterchange 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 deenveloping program example” on page 873 and “Outbound enveloping program example” on page 874.

Processing multiple incoming TS queues

The DataInterchange 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.

Name	Offset	Length	Type	Description
RESERVED	0	16	Char	Reserved for DataInterchange
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

Name	Offset	Length	Type	Description
MTSQTSQ5	48	8	Char	5th TS queue name
MTSQTSQ6	56	8	Char	6th TS queue name
RESERVED	64	16	Char	Reserved for DataInterchange

When DataInterchange 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 “DataInterchange Utility control information” on page 507.

After invoking all necessary response programs, the DataInterchange Utility releases the storage it acquired to hold the multiple-TSQ control blocks. If a continuous receive is not set up to deenvelope or translate (in other words, invokes only a continuous receive response program), the response program must release any storage it acquires to hold multiple-TSQ control blocks. Incoming S@#xxxxx TS queues generated by Expedite/CICS are not deleted by DataInterchange or by Expedite/CICS. Your response program must delete these queues. If there are multiple TS queues, there will be multiple S@#xxxxx queues.

When the DataInterchange 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 DataInterchange Utility files

The DataInterchange Utility must ensure serial access to all sequential files it uses, such as the print file, envelope file, and tracking file. If two DataInterchange Utility transactions are executed concurrently, and both are generating interchanges to the same TS queue, the interchange records in the output file may be intermixed. To avoid this potential problem, the DataInterchange 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 DataInterchange 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, DataInterchange 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 DataInterchange 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 DataInterchange Utility). In general, a unit of work is defined as the portion of work between two synchronization points. For CICS, the following are synchronization points:

- **CICS Transaction Initiation**
Marks the beginning of the first unit of work in a single CICS transaction. This unit of work might be the only unit of work in the transaction if the CICS SYNCPOINT command is not issued.
- **CICS SYNCPOINT Command**
Marks the completion of the current unit of work and the start of the next unit of work.
- **CICS Transaction Termination**
Marks the end of the last unit of work in a CICS transaction. At this point, an implicit CICS SYNCPOINT occurs. If no CICS SYNCPOINT commands were issued during transaction execution, transaction termination marks the completion of the only unit of work in the transaction.

CICS also supplies the CICS SYNCPOINT ROLLBACK command backs out the current unit of work as though it never existed. CICS also backs out in-process transactions that were interrupted because of an extraneous event such as a power outage or cancellation of the CICS job.

For CICS, a unit of work consists of updates to recoverable resources. If the unit of work is synchronized, all modifications to recoverable resources are applied. If the unit of work is not synchronized, all modifications are backed out. When a backout occurs, none of the updates made to recoverable resources are applied. Whether the backout occurs after one or multiple updates, all changes that occurred after the last synchronization point are removed.

All updates made to recoverable resources within a unit of work are considered a single group. Either the entire group of changes is applied or none are applied, depending on whether the unit of work is synchronized or backed out.

Transaction Store DB2 tables are inherently recoverable. The following user resources can be defined as recoverable and may be accessed by the DataInterchange 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 MQSeries Queues. If you define these files as recoverable, all changes made to them are synchronized when the unit of work is completed successfully or backed out when the unit of work is not completed.

DataInterchange 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 DataInterchange Utility, passing the same control and even the application information as it was given when the failure occurred.



NOTE: DataInterchange recovery does not use any type of journaling or checkpointing internally. DataInterchange does not automatically pick up where it left off but must be re-driven by your application.

DataInterchange Utility unit of work

The unit of work varies based on the function of the DataInterchange Utility being executed. If you want the DataInterchange Utility to control the unit of work, set the **Syncpoint interval** field to **0** or **1**. For more information, see the **SYNCVAL** field in “DataInterchange Utility control information” on page 507.

The following table lists all DataInterchange Utility commands and the point at which a CICS SYNCPOINT command can be issued.

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.
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 DataInterchange 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.

For command:	A syncpoint is issued:
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.
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 DataInterchange Utility commands that include some type of interchange processing, the unit of work is controlled by the recovery level. The following commands include interchange processing:

- DEENVELOPE
- DEENVELOPE AND TRANSLATE
- ENVELOPE
- ENVELOPE AND SEND
- RECEIVE AND DEENVELOPE
- RECEIVE AND TRANSLATE
- REENVELOPE
- REENVELOPE AND SEND
- TRANSLATE AND ENVELOPE
- TRANSLATE AND SEND

You can specify the recovery level by setting the **RECOVERY** keyword as described below:

- **Interchange level recovery (E)**
All recoverable resources associated with an interchange are included in the unit of work scope. This includes input data read from recoverable intrapartition TD queues, Transaction Store DB2 files, records written to the tracking file, and so on.

As discussed previously, when a CICS SYNCPOINT command is issued, the updates are applied to all recoverable resources within the unit of work. If a backout occurs, all updates are removed from the system and all resources return to the state they were in before the unit of work began (default in CICS).

- **Transaction level recovery (T)**
All recoverable resources associated with a transaction are included in the unit of work scope. This poses a problem for clean recovery because interchange and group records are added to the Transaction Store with the first transaction of every interchange. A CICS SYNCPOINT is issued after each transaction.

If a backout occurs in the middle of an interchange, any transactions previously translated would already be added to the Transaction Store. To avoid this condition in the CICS environment, it is best to use interchange-level recovery.

Identifying the DataInterchange unit of work

When you perform a TRANSLATE TO STANDARD operation, there are some situations where the DataInterchange 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 DataInterchange in determining the end of the unit of work.

- When using C and D record formatted data, the DataInterchange 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 DataInterchange Utility recognizes the Z record as the signal to complete the transaction and end the unit of work, starting a new interchange. DataInterchange 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 DataInterchange 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 DataInterchange Utility does not perform an extra read to determine the end of interchange condition. This preserves interchange level recovery with the application file.

Including DataInterchange changes in your application's unit of work

This is an important issue if you want the changes made by the DataInterchange 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 DataInterchange 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 DataInterchange 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 “DataInterchange Utility control information” on page 507.

The following guidelines are suggested if the DataInterchange 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 DataInterchange Utility is sensitive to the recovery level. Using a recovery level of **E** causes DataInterchange to hold Transaction Store updates until all CPU-intensive work has completed. This greatly increases the level of concurrency which can be achieved with other CICS transactions performing DataInterchange Utility functions.
- Make all application changes to recoverable resources after the DataInterchange Utility has returned control to your application. The translation process is CPU-intensive and DataInterchange 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 DataInterchange 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 521.
- Use the CICS SYNCPOINT command in continuous receive response programs and utility termination response programs. For more information, see “Response applications” on page 521.

Terminal-attached applications

The throughput time of the DataInterchange 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 DataInterchange 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 DataInterchange 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 DataInterchange 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 DataInterchange Utility control information.

Running the DataInterchange Utility in a separate CICS region

You can execute the DataInterchange Utility in a separate region from the driving application. All three methods of invocation can be used to initiate the DataInterchange 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 DataInterchange Utility is to be executed through the CICS START or ATI commands, then any version of CICS can be used.

When splitting the DataInterchange 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 DataInterchange Utility in the same region as the application, but the DataInterchange Utility provides the flexibility to allow the split, if necessary.

DB2 setup considerations

DataInterchange 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 DataInterchange have access to the DataInterchange DB2 plan. If you are developing an application that issues the CICS LINK command to program EDIFFUT to gain access to DataInterchange 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.

Assumptions:

1. No more than 5 online users, EDIA entry, increase if more are required.
2. No more than 10 translations or other Expedite transactions allowed concurrently, EDIB entry, increase if more are required and system resources are available.
3. A maximum of 17 DB2 threads allowed in the pool, the sum of THRDM values for all ENTRY statements in the pool.
4. A DB2 TCB limit of 20, 3 + 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
*-----*
*                               DataInterchange/MVS-CICS
* Customer written transactions which EXEC CICS LINK to program
* EDIFFUT should also be added to the EDIB macro entry. Note:
* LG01, IMR1, and IST1 are Expedite/CICS transactions and may be
* removed if Expedite/CICS is not used.
*-----*
DSNCRCT TYPE=ENTRY,AUTH=(SIGNID,*,*),THRDM=10,THRDA=10,        X
      TXID=(EDIB,EDID,EDIG,EDIM,EDIQ,EDIR,                      X
      EDIS,EDIT,EDIV,EDIW,EDIZ,LG01,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 DataInterchange is running. The number of DB2 threads represents the number of EDIB pool transactions that will be executed concurrently. The maximum number of threads needed is the sum of all TRANCLASS values for the transactions in the EDIB pool. The minimum is three, but five or more is typical. The pool transactions include EDIB, EDID, EDIG, EDIQ, EDIR, EDIS, EDIV, EDIW, EDIZ, LG01, IMR1, and IST1. You may also include user transactions that initiate the DataInterchange 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 DataInterchange Transaction Store lock (EDITSLT) to complete its processing. EDIE is a started task, triggered by CICS when an entry is written to the EDI3 intrapartition TD queue, and requires a thread for processing. More than one EDIE transaction can be started by CICS, but the transactions will serialize so that two threads are sufficient. No more than 3 threads are recommended for this pool.

The EDIA pool is for the long running task, EDIA. EDIA is the administrative tool for DataInterchange 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, then 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

DataInterchange 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 DataInterchange transactions EDIA and EDIT, or EDIB and EDIT.

Running DataInterchange/MVS-CICS 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, DataInterchange 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 DataInterchange's initialization routine. Each execution of the DataInterchange 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 DataInterchange once, invoke DataInterchange's translation services multiple times, and terminate DataInterchange 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 DataInterchange 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 DataInterchange API and the Utility, which requires user-written applications or tasks. This can be achieved several ways. One scenario is outlined below.

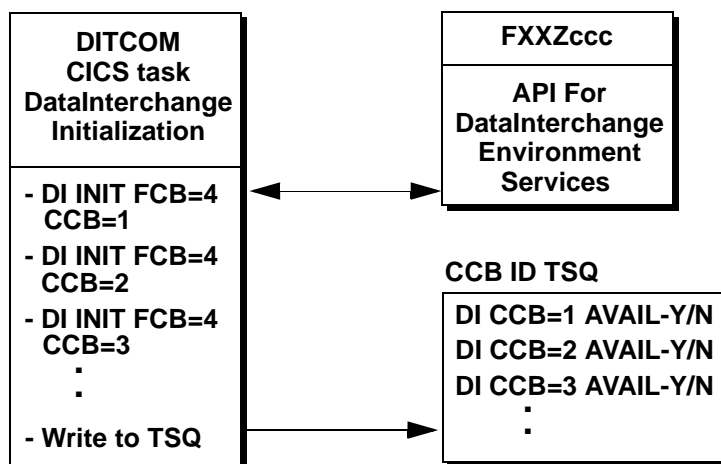
HOT-DI implementation requires DataInterchange/MVS-CICS and can only be used with Expedite/CICS when using the PERFORM commands to send data. It is not currently supported by environments using DataInterchange's continuous receive mailbox function.

Initializing HOT-DI

Initialization syntax

```
DIINIT  
FXXZccc(SNB,CCB1,FCB,applid,sysid)  
FXXZccc(SNB,CCB2,FCB,applid,sysid)  
FXXZccc(SNB,CCB3,FCB,applid,sysid)
```

HOT-DI initialization diagram



Initializing DataInterchange

A user-written CICS task, such as DIINIT, is required to initialize DataInterchange but after initialization, this task should end. Field information and pointers to storage areas required by DataInterchange 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 304.

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**. DataInterchange will use the shared storage option when acquiring main storage.
- The CCB must be different for each initialized DataInterchange task and must be used when requesting other DataInterchange 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 DataInterchange service is in process).

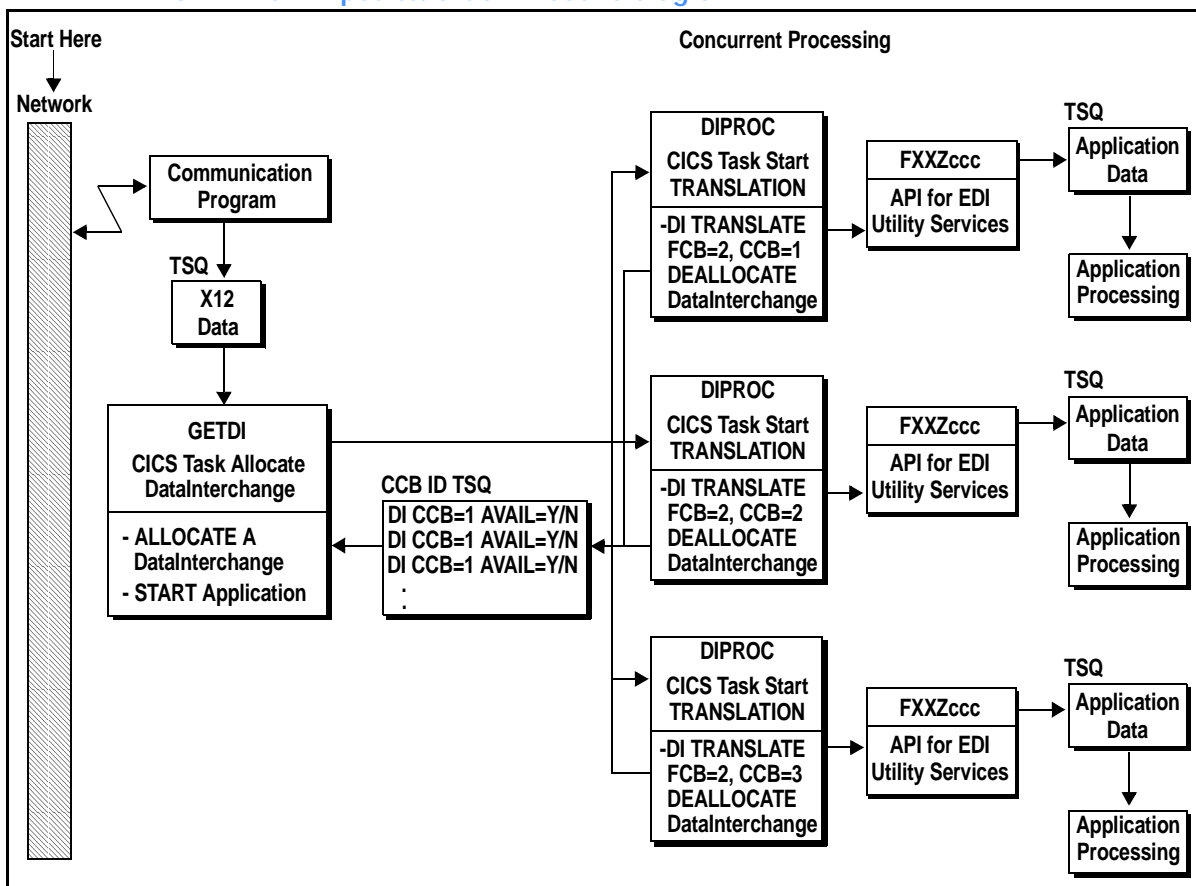
HOT-DI processing considerations

A user-written CICS task (such as GETDI) controls when a DataInterchange 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 “DataInterchange/MVS-CICS considerations” on page 715.

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 may 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**.

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 306.

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**. DataInterchange 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 “DataInterchange Utility control information” on page 507.

DataInterchange return code considerations

Your user-written program may interrogate DataInterchange's API return codes (**ZCCBRC** field) and extended return codes (**ZCCBERC** field) in the CCB. For descriptions of these codes, see “DataInterchange condition codes and API return codes” on page 655.

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 DataInterchange variable DIMAPSWITCH, then you must ensure that global variables are handled appropriately in both translation usages/rules.

Terminating DataInterchange

A user-written CICS task (such as DITERM) is needed to terminate each active HOT-DI task. Termination involves making a DataInterchange 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 306.

If any CCB is busy for a specified number of attempts on each DataInterchange 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)

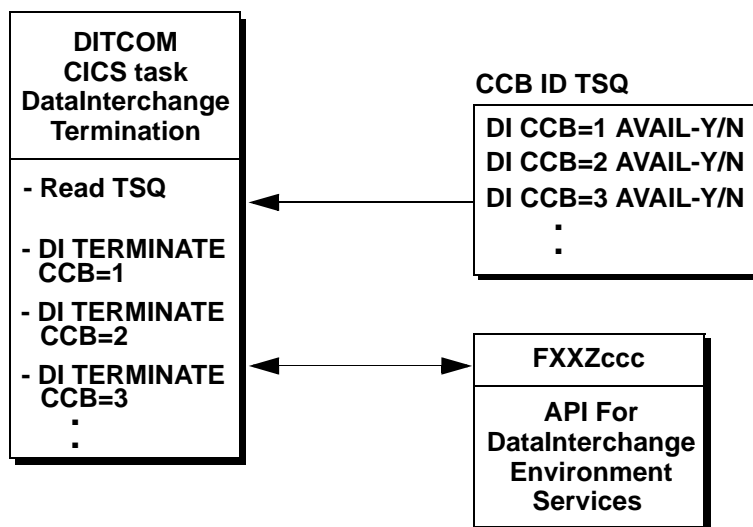
- Set the **ZSNBNAME** field in the SNB to **ENVSERV**.
- Set the **ZFCBFUNC** field in the FCB to **2**. DataInterchange will use shared GETMAINs.

Terminating HOT-DI

Termination syntax

```
DITERM
FXXZccc (SNB,CCB1,FCB)
FXXZccc (SNB,CCB2,FCB)
FXXZccc (SNB,CCB3,FCB)
```

HOT-DI termination diagram

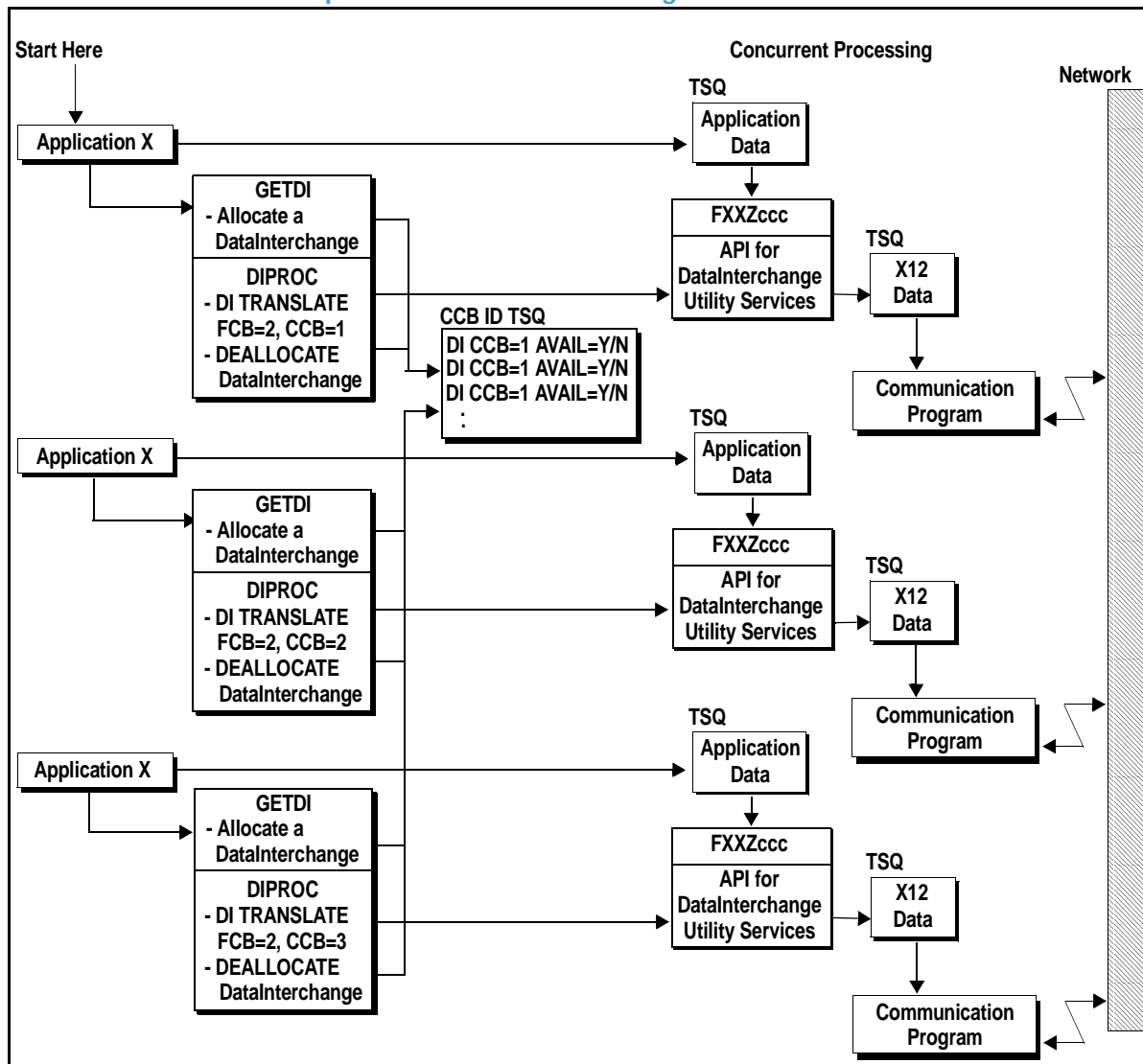


Outbound communications

Outbound communications can be requested using DataInterchange's API with Utility services or Communication services.

For information about the Utility service or Communication service function request SYNTAX and control block parameters, see Chapter 1, "Using The Datainterchange Utility," or "Communication services" on page 407.

HOT DI non-Expedite/CICS - outbound diagram



DataInterchange Utility control information

When your application invokes the DataInterchange Utility, control information must be passed. The table below describes the format of the area given to the DataInterchange 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 DataInterchange.

The Result field indicates whether the field is used internally by DataInterchange or is set by DataInterchange 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 DataInterchange 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 DataInterchange before control is returned to the initiating application or passed on to a response application
- X** The field is used internally by DataInterchange. Initialize it with blanks or binary zeros, and do not modify it after initialization.
- blank** The field is used by DataInterchange but does not require special consideration.

Format of DataInterchange 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
APPLID	Char	73	8		Override application ID
LANGID	Char	81	6		Override language ID

Name	Type	Offset	Length	Result	Description
RESPID	Char	87	8		Response program or transaction ID
RESPTYP	Char	95	2		Response type
RTERMID	Char	97	4		Response terminal ID
USRFLD	Char	101	16		User field to pass data
SYSID	Char	117	8	X	CICS system ID override
RESPFLAG	Char	125	1	Y	Response indicator
FILETYP	Char	126	2	Y	Network acknowledgment file type
ECBP	Bin	128	4		ECB address
CCBRC	Bin	132	4	Y	Severity code (UTILSEV)
CCBERC	Bin	136	4	Y	Condition code (UTILCCODE)
FILEID	Char	140	8	Y	Envelope TSQ/Net acknowledgment file
APPFILE	Char	148	8	Y	Translated data TS Queue
ABNDCODE	Char	156	4	Y	CICS abend code if one occurred
THANDLE	Char	160	20	Y	Transaction handle
FFIEHDR	Bin	180	4	Y	Information Exchange long header address
FARC	Bin	184	4	Y	Functional acknowledgment return code
FAERC	Bin	188	4	Y	Functional acknowledgment extended return code
FABUILT	Char	192	1	Y	Functional acknowledgment built flag
FFMTFLG	Char	193	1		EDI standard multi-TSQ flag
USERSYNC	Char	194	1		User syncpoint flag
APMTFLG	Char	195	1		Application multi-TSQ flag
USERCOND	Bin	196	4		User condition code pointer
APMTCBP	Bin	200	4		Application multi-TSQ pointer
RES1	Char	204	10		Reserved for future use
FFNOCNV	Char	214	1		No return code conversion flag
FANAME	Char	215	8		Functional acknowledgment file name
RESPACTV	Char	223	1	X	Response program active flag
WORKNAME	Char	224	8	X	Name of workfile

Name	Type	Offset	Length	Result	Description
BATFLG	Char	232	1		Batch or UTILSRV API (HOTDI) flag
NOEXCP	Char	233	1	X	No exception file
INVPARM	Char	234	1	X	Network program invalid parameter
LOGACTV	Char	235	1	X	Logging CE0050 active
FFUSADDR	Bin	236	4	X	Pointer to this block
CCBP	Bin	240	4	X	Pointer to the CCB
FFMTCBP	Bin	244	4		EDI standard multi-TSQ control block pointer

DataInterchange Utility control information field descriptions

SYNCVAL

The number of units of work completed before DataInterchange issues a CICS SYNCPOINT command. For a complete description of the unit of work for each command, see “DataInterchange Utility unit of work” on page 493. 4-byte binary value. The IMPORT command ignores this value. Valid values are:

- 0 or 1** Data Interchange issues the syncpoints (default).
- 1** Your application issues the syncpoints. DataInterchange does not issue any syncpoints.
- other** DataInterchange 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 DataInterchange 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 DataInterchange Utility.

CMDLEN

The length of the command language string in main storage. If the main storage option is not used, you must set this field to **0**, to indicate that values in the command file name (**CMDNAME**) and command type (**CMDTYPE**) fields should be used instead. 4-byte binary value.

CMDNAME

The file name of a TS queue or TD queue that contains the command language input to be processed. Lowercase characters are changed to uppercase characters, so the command language syntax is free-form. The default is **SYSIN**.

CMDTYPE

The file type of the command file. Valid values are **MQ**, **TD**, **TM**, and **TS**. The default is **TS**.

DELIMITER

This command value delimiter can be used in instead of the left and right parentheses to enclose values in the DataInterchange Utility command language. You must supply this value if a command includes the application control field keyword (**ACFIELD**) with a value of a left or right parenthesis.

PRTNAME

The name of the print file. For a description of this file, see “Print file (PRTFILE)” on page 179. The default is **PRTFILE**.

PRTTYPE

The type of print file. Valid values are **MQ**, **TD**, **TM**, and **TS** (default).

RPTNAME

The name of the report file. For a description of this file, see “Report file (RPTFILE)” on page 180. The default is **RPTFILE**.

RPTTYPE

The type of report file. Valid values are **MQ**, **TD**, **TM**, and **TS** (default).

EXCPNAME

The name of the exception file. For a description of this file, see “Exception file (FFSEXCP)” on page 177. The default is **FFSEXCP**.

EXCTYPE

The type of exception file. Valid values are **MQ**, **TD**, **TM**, and **TS** (default).

TRAKNAME

The name of the tracking file. For a description of this file, see “Tracking file (FFSTRAK)” on page 178. The default is **FFSTRAK**.

TRAKTYPE

The type of tracking file. Valid values are **MQ**, **TD**, **TM**, and **TS** (default).

QRYNAME

The name of the query file. For a description of this file, see “Query file (EDIQUERY)” on page 180. 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 DataInterchange. The activity log profile must contain a matching entry to define the log file used for recording errors and events pertaining to the application. The default is **EDIFFS**.

LANGID

The override language ID identifies which language version is being used. Valid value is:

ENU English (default)

RESPID

The ID of a response program or transaction that DataInterchange should invoke after it has finished processing. See “Response applications” on page 521 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 521 for additional information.

SYSID

The CICS system ID override. For DataInterchange use only.

RESPFLAG

The response indicator identifies which action caused the response program to be invoked. Valid values are:

- C** The response program was invoked because a continuous receive completed. The **ENVFILE** field contains the name of the TS queue holding the interchange processed during this continuous receive. If the continuous receive was on behalf of a network acknowledgment, the **ENVFILE** field contains the name of the TS queue holding the network acknowledgment.
- T** The response program was invoked because of a translation-to-application function. The **APPFILE** field contains the name of a TS queue holding data in the application format. The **HANDLE** field contains the transaction handle associated with the transaction.
- U** The response program was invoked because the DataInterchange Utility completed.

FILETYP

The type of network acknowledgment file specified in the **FILEID** field if the **PROCESS NETWORK ACKS** command was executed. Possible values are **MQ**, **TD**, **TM**, and **TS**.

ECBP

The address of an ECB that DataInterchange 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 DataInterchange processing.

CCBERC

The DataInterchange Utility condition code. This field is equivalent to the **UTILCCODE** field. For more information, refer to *DataInterchange Messages and Codes*.

FILEID

The envelope file name of the TS queue holding the interchange that was received with the continuous receive facility. Applies only when the response application is invoked at the completion of a continuous receive. The response application must delete this TS queue or process it further. If the continuous receive is set up for network acknowledgments, this field contains the name of the TS queue that contains the network acknowledgment processed. If the TS queue contains a network acknowledgment, DataInterchange deletes the TS queue upon return from your continuous receive response application. If the **PROCESS NETWORK ACKS** command is used, this field contains the name of the TS queue or TD queue containing the network acknowledgment. The **FILETYP** field contains the associated file type.

APPFILE

The application data TS queue holding the application data in C and D format or in raw data format. Applies only when the response application is invoked because a translation or retranslation request is being processed. This field tells the response application that the TS queue contains one transaction in application format. The response application must delete or process the TS queue further.

ABNDCODE

If an abend occurs during DataInterchange 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 **ABNDCODE** field. If an abend occurs, DataInterchange also issues the **CICS DUMP** command with an associated dump code of **ED11**.

THANDLE

The character representation of the Transaction Store handle of the translated transaction. Applies only when the response program is invoked because of a translation-to-application type function. A handle is always given, even if no data is generated from the translation. Data is not generated if the error level exceeds the acceptable value for this transaction. This value is not supplied if the response application is invoked because of continuous receive or DataInterchange Utility completion.

FFIEHDR

The address of the receive message long header given by Information Exchange to Expedite/CICS during a continuous receive. This field is only passed when the utility is invoked due to a continuous receive. Refer to the *Information Exchange Interface Programming Guide* for the format of the Information Exchange receive message header. This address is set only under the following conditions:

- If either the **Translate** or **Deenvelope** value is set to **Y** in the associated continuous receive profile member, the EDI1 TD queue must be defined. If EDI1 is not defined, the address is not set.
- If both the **Translate** and **Deenvelope** values are set to **N**, the **Response Type** value must be set to **PG** in the associated continuous receive profile member. In other words, if only a response application is to be invoked due to a continuous receive and no other DataInterchange processing is requested, the application must be a program and not a CICS transaction to receive this field.

FARC

The highest return code encountered during functional acknowledgment processing.

FAERC

The highest extended return code encountered during functional acknowledgment processing.

FABUILT

Indicates whether functional acknowledgments were generated. This field works in tandem with the **FANAME** field.

FFMTFLG

During continuous receives, indicates whether incoming EDI standard data spans more than one TS queue. This field may 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 489.

USERSYNC

Indicate to DataInterchange 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 may 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 489.

USERCOND

The user condition code pointer. For DataInterchange 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 489.

RES1

Reserved for DataInterchange.

FFNOCONV

The return code conversion flag. For DataInterchange 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 DataInterchange use only.

WORKNAME

The name of the work file. For DataInterchange 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 DataInterchange use only.

INVPARM

Indicates whether an invalid parameter was passed to the network program. For DataInterchange use only.

LOGACTV

Indicates that logging CE0050 messages is active. For DataInterchange use only.

FFUSADDR

The pointer to this block. For DataInterchange use only.

CCBP

The pointer to the CCB. For DataInterchange 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 489.

Continuous receive considerations

The continuous receive Facility is a DataInterchange service that works in conjunction with Expedite/CICS and Information Exchange. It also can be used with MQSeries trigger queues. By defining the members of a continuous receive profile, you can completely automate the receive process. For details on each field contained within the continuous receive profile, see "Continuous receive profile (CONTRECV-P2 for CICS only)" on page 255.

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, DataInterchange 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 DataInterchange generates the application data. When using continuous receive for network acknowledgments, the status of EDI transactions in your Transaction Store is kept up to date without initiating the UPDATE STATUS command at different points during the session.

Continuous receive using MQSeries

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

1. Your CICS and MQSeries administrators must enable the CICS region for MQSeries support. As part of this process, they should define an MQSeries trigger event queue, monitored by the MQSeries transaction CKTI in the region. In this example, the name of this queue is **CICS1.TRIGGER**.
2. Using the DEFINE PROCESS command, your MQSeries administrator must define one MQSeries process to provide MQSeries with information about the DataInterchange transaction EDIQ. An example of the DEFINE PROCESS command follows:

```
DEFINE PROCESS(EDIPROCESS) APPLICID(EDIQ) APPLTYPE(CICS)
```

3. Using the DEFINE QLOCAL command, your MQSeries administrator must define at least one MQSeries queue to receive envelopes. An example of the DEFINE QLOCAL command follows:

```
DEFINE QLOCAL(EDIRECEIVE) INITQ(CICS1.TRIGGER)
PROCESS(EDIPROCESS) TRIGGER TRIGTYPE(FIRST)
TRIGDATA( 'CRPROF=MQ1 MQPROF=RECV1 ' )
```

The **TRIGDATA** field contains two mandatory keyword-value combinations. The first, **CRPROF**, indicates the name of the DataInterchange continuous receive profile member to use when data is received. The second, **MQPROF**, relates a DataInterchange MQSeries profile member to this specific MQSeries queue. The order in which you specify the two keyword-value combinations is not important.

In this example, only one MQSeries queue is used for event-driven EDI but you may define multiple queues. As long as the DataInterchange and MQSeries information is set up correctly, DataInterchange will process any number of queues.

4. In DataInterchange, you must define an MQSeries queue profile member. In this example, the name of the member is **RECV1**, and the **Full Queue Name field** is set to **EDIRECEIVE**. Other fields are set accordingly.
5. In DataInterchange, 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 “DataInterchange processing after data is received” on page 518 to tell DataInterchange what processing is required once data is received from MQSeries.

Once the above steps have been completed successfully, DataInterchange automatically processes all data written to the MQSeries queue as soon as MQSeries dispatches the trigger event messages.

Continuous receive selection criteria

You can send and receive data from a single mailbox and to process network acknowledgments through continuous receive. By defining selection criteria, you can set up continuous receive members to receive and process EDI data, and also set up another member to receive and process network acknowledgments, all with the same requestor ID.



NOTE: When using MQSeries trigger queues, there is no real selection criteria available through the DataInterchange continuous receive facility. Rather, the continuous receive profile (CONTRECV) is used to tell DataInterchange how to handle data once an MQSeries trigger event has occurred. So, if you are using MQSeries for continuous receive processing, skip to the next section, “DataInterchange processing after data is received” on page 518.

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 CONTRECV makes the profile member active. The settings in the following profile fields combine to make the selection criteria unique:

Profile field	Description
REQUESTOR ID	The mailbox (requestor) profile member from which you wish to receive data and which contains the Information Exchange account, user ID, and password information. You can have many continuous receive profile members with the same requestor ID, but each member must be unique. You can also have different requestor IDs in each continuous receive profile member. This makes each member unique without having to further qualify the continuous receive.
TP NICKNAME	A trading partner from which you can receive data. If you complete this field, you must also supply the account and user ID fields for the associated trading partner profile member. If you use this field, only incoming data for this specific trading partner is received and processed.
MESSAGE USER CLASS	A code that you and your trading partners agree to use. If you use this field, only incoming data with a matching message user class is received and processed.
NETWORK ACKS ONLY	If you use this field, only network acknowledgments are received and processed. DataInterchange issues a continuous receive with the sender account of *SYSTEM* , a user ID of *ERRMSG* , and a data type of A . This tells Expedite/CICS that only network acknowledgments (and no data) are to be received. The default is N .

DataInterchange processing after data is received

The previous section describes how to dictate the selection criteria for each continuous receive. Once the data is received, DataInterchange must be told what processing to perform. This is controlled by the following profile fields, listed below in order of precedence.

Profile field	Description
NETWORK ACKS ONLY	If the value of this field is Y , the data received is always a network acknowledgment and is processed as such. The actual processing is done with a CICS START command, and is issued for transaction EDIB (DataInterchange Utility). The PROCESS NETWORK ACKS command is given to the DataInterchange Utility along with the acknowledgment. If a response program is supplied in the continuous receive profile, it is given control and the name of the TS queue containing the acknowledgment is passed. Once control is returned to the DataInterchange Utility, the TS queue is deleted. A value of Y in this field overrides a value of Y in the TRANSLATE field.

Profile field	Description
TRANSLATE	If the value of this field is Y , the interchange received by Expedite/CICS and given to DataInterchange is deenveloped and translated to the application format. This process may also generate functional acknowledgments because they are created during the deenvelope process. Processing is performed in a separate CICS transaction (EDIB). The DEENVELOPE AND TRANSLATE command and the TS queue holding the interchange are passed. A value of Y in this field overrides a value of Y in the DEENVELOPE ONLY field.
DEENVELOPE ONLY	If the value of this field is Y , the interchange received by Expedite/CICS and given to DataInterchange 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 DataInterchange Utility). The DEENVELOPE command and the TS queue holding the interchange are passed.

If you leave these fields blank, or set them to **N**, DataInterchange 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, DataInterchange uses this queue to pass information from transaction IMR1 to transaction EDIB. When the queue is defined, the IMR1 transaction waits for the EDIB transaction to complete. Having IMR1 wait for EDIB to complete closes a recovery exposure and ensures all interchanges received by Expedite/CICS are processed by DataInterchange. 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 Expedite/CICS and DataInterchange can recover appropriately. You must define EDI1 as non-recoverable.

The EDI1 TD queue may also be used with DataInterchange's non-Expedite/CICS continuous receive interface.

Sent to Network status

When Expedite/CICS Version 4.1 (or higher) is used and the IINCICS network profile communication routine name is VANEXPV4, transaction status is updated from **Send requested** to **Sent to network** once the transactions have successfully been sent by Expedite/CICS. If, for some reason, a transaction is not sent successfully, its status is changed from **Send requested** to **Not sent - network error**. DataInterchange and Expedite/CICS typically send interchanges asynchronously. When DataInterchange requests a send, Expedite/CICS will acknowledge receipt of the request immediately but may not perform the send at the same time. Once the message has actually been sent to the network, Expedite/CICS (transaction IST1) will LINK back to DataInterchange to update the status of the transaction appropriately.

DataInterchange and Expedite/CICS can send interchanges synchronously. The Expedite/CICS facility (transaction LG01) is used to indicate whether processing is done synchronously or asynchronously. The VANEXPV4 routine updates transaction status in either case. During the VANEXPV4 status update process:

- DataInterchange Transaction Store statuses are updated.
- The unique ID assigned by the network is inserted into the DataInterchange database and used as an index for processing subsequent network acknowledgments. This will improve performance for the network acknowledgment processing.



NOTE: DataInterchange supports compress and delivery priority for Expedite/CICS.

Using continuous receive outside Expedite/CICS

Thus far, the continuous receive facility has been described as it works with Expedite/CICS and Information Exchange only. You can also take advantage of the continuous receive profile with your own communication routines by using a DataInterchange API to develop applications that interact with DataInterchange in a continuous receive mode. See “Continuous receive interface (CICS only)” on page 565 for details.

You can process multiple incoming TS queues when continuous receive is used without Expedite/CICS. For more information, see “Processing multiple incoming TS queues” on page 489.

Response applications

Response applications are user-written CICS programs or transactions that DataInterchange invokes at certain points of processing. If you are designing your application to run asynchronously with the DataInterchange 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 DataInterchange 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. DataInterchange must handle all abends in order to release ENQs and other resources. If DataInterchange is not allowed to handle abends, subsequent user tasks may hang.

- Specify a response type of **TX** to cause DataInterchange 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 “DataInterchange Utility control information” on page 507 for the format of the results control information given to your application.

Types of response applications

DataInterchange can invoke three types of response applications:

- DataInterchange 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 511.

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**).

DataInterchange Utility response application (U)

The DataInterchange Utility response application is only applicable when your main program invokes the DataInterchange Utility. For example, you can CICS START transaction EDIB and pass control information as specified in “DataInterchange Utility control information” on page 507. 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 DataInterchange Utility completes the processing you requested when you started EDIB. This DataInterchange 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 DataInterchange was originally invoked. In the previous example, your application started transaction EDIB, and DataInterchange processing was asynchronous with your application. It is important that you know the outcome of the function you requested. When you specify a DataInterchange Utility response application, the results that DataInterchange 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 “DataInterchange Utility control information” on page 507 for the format of the control information passed to the response application.

DataInterchange 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 DataInterchange. The user field contains the same value that was supplied when the DataInterchange Utility was initiated, unless it has been modified by translation response applications. For more information, see “Transaction response application (T)” on page 523.

The DataInterchange 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 DataInterchange during the entire process. Based on the outcome, your DataInterchange Utility response application might execute another internal program, or might execute DataInterchange to perform subsequent functions.

A DataInterchange 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 DataInterchange Utility performs a CICS START EDIB and tells the DataInterchange 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 DataInterchange Utility and response application updates in the same unit of work, as long as your response application is a program. This scenario removes the DataInterchange Utility from your initiating application's unit of work, but the response application you develop is still in charge of the DataInterchange 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 DataInterchange has completely processed the entire received envelope TS queue, and DataInterchange 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 DataInterchange Utility control information. For a description of the resulting information, see “DataInterchange Utility control information” on page 507.

The response indicator (**RESPFLAG**) and user area (**USRFLD**) fields are useful in identifying why your application is being invoked. DataInterchange 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 523.

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 DataInterchange has processed the network acknowledgment and updated the status in the Transaction Store. The name of the TS queue containing the acknowledgment is in the **FILEID** field. A continuous receive response application is optional when receiving and processing network acknowledgments. The only reason to use a continuous receive response application is to save the network acknowledgment itself. The TS queue holding the network acknowledgment is deleted before termination. If you have no reason to save the network acknowledgment, you do not need to develop a continuous receive response application.

If the continuous receive response application is a program (**type=PG**), this is a good place to issue a CICS SYNCPOINT. Consider this option, especially if you set the **Allow syncpoints** field in the continuous receive profile to **N**. By doing this, your continuous receive response program gains control and the unit of work is still active. At this point, you can change your recoverable resources and issue the CICS SYNCPOINT command. The unit of work would then include the updates made to the DataInterchange Utility and the changes made by your application to recoverable resources.

If you want the syncpoints issued more frequently, especially where you need control after each transaction is processed, see “Transaction response application (T)” on page 523 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 DataInterchange Utility PEFORM commands is invoked:
 - RECEIVE AND TRANSLATE
 - DEENVELOPE AND TRANSLATE

- TRANSLATE TO APPLICATION
 - RETRANSLATE TO APPLICATION
- After the TRANSLATE RECEIVED TRANSACTIONS or RETRANSLATE RECEIVED TRANSACTIONS options are invoked from the online CICS Transaction Store facility.

The point at which transaction level response programs are invoked is based on the recovery level (DataInterchange Utility parameter **RECOVERY**). If the recovery level is **E** (envelope), then 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 DataInterchange 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 DataInterchange Utility control information. The name of a unique TS queue that holds the translated application data and assigned by DataInterchange is passed within this control information. This is the name of the application data TS queue (APPPFILE). The name of the TS queue is eight characters long and begins with **EDI**. For more information, see “DataInterchange Utility control information” on page 507.

The response indicator (**RESPFLAG**) and user area (**USRFLD**) fields are useful in identifying why your application is being invoked. DataInterchange 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 DataInterchange Utility.
- Blanks if the DataInterchange Utility is being invoked because a TRANSLATE RECEIVED TRANSACTIONS or RETRANSLATE RECEIVED TRANSACTIONS option was requested from the online CICS Transaction Store facility.
- The user area modified by a previous invocation of a translation response application. If a translation response application modifies the user area, the next invocation of a response application will receive the updated user area value, including utility response applications and continuous receive response applications.

The modified value of the user area is reset whenever the DataInterchange 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 DataInterchange Utility condition code (**CCBERC**) fields. You can inspect these fields to determine the success or failure of the translation. For more information, refer to *DataInterchange Messages and Codes*. 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. DataInterchange 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 DataInterchange if no errors occur, and can be used as you choose.



NOTE: Transaction level response applications should not issue the CICS **SYNCPPOINT** command if the DataInterchange 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. DataInterchange 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** and a usage/rule is found for this transaction, the **THANDLE** field is filled in.
- **3** and 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 DataInterchange Utility termination response program, whichever is applicable.

Persistent environment

The DataInterchange persistent environment is an optional feature that can be used with CICS/ESA systems to improve DataInterchange performance. It accomplishes this by reducing the amount of read operations needed to perform translation and other functions. As data is read from the DataInterchange database, some of it is saved in an MVS data space. The next time the data is needed, it is obtained from the data space instead of the database. The data space remains functional throughout a DataInterchange session. A DataInterchange session starts when the first DataInterchange function is requested in a CICS region and ends when DataInterchange transaction EDIT is terminated.

This process allows many DataInterchange transactions to run concurrently and over a long period of time, obtaining the same piece of data repeatedly with only one access to the DataInterchange database.

The data saved into the data space includes control strings, receive usages/rules, send usages/rules, and maps.

Running multiple MVS subtasks

Although there is only one DataInterchange data space per CICS region, DataInterchange will start up to sixteen MVS subtasks to manage the DataInterchange data space. The more subtasks running, the greater the throughput the data space can deliver. The value used to determine how many subtasks will be started is obtained from the SYSPROF profile for the CICS region. You may adjust the number of subtasks up or down to fit your requirements. The number of subtasks defined in the SYSPROF profile should not exceed the number of DataInterchange threads that may execute concurrently on the CICS region.

Sizing the MVS data space

The size of the MVS data space is determined by a couple of factors. First is the maximum size specified in the SYSPROF profile for the CICS region. The data space will not be made larger than the value specified in the SYSPROF profile. The value in the profile cannot exceed the installation maximum. Unless altered by the installation, this value will be the network default of 239 4000-byte blocks (less than 1 MB total). Your installation can use the installation exit IEFUSI to change the IBM default.

Secondly, DataInterchange learns based on past usage and can define a data space smaller than the maximum size specified. The format of the data space is adjusted automatically, based on past usage to provide the most efficient data space structure. An effort is made to structure the data space at initialization so that no restructuring will be needed during regular DataInterchange processing.

However, depending on installation usage and maximum size of the data space, all or part of the data space may become full during processing. When this occurs, DataInterchange expands the data space (if defined smaller than the maximum), and/or reorganizes the structure of the data space. If necessary, older data in the data space is removed. Ideally, you do not want any of these situations to occur since each adjustment momentarily delays DataInterchange processing. DataInterchange will recognize these conditions and try to adjust for them in the next session. If these conditions persist after a few sessions, it may be desirable to increase the maximum data space size. You can monitor the DataInterchange log file for messages indicating that these scenarios have occurred. All message identifiers relating to the persistent environment begin with the qualifier **GB**.

Make sure that DataInterchange sessions are ended normally when using the persistent environment. A DataInterchange session is considered ended when transaction EDIT is removed from the system. For information on removing transaction EDIT from the system, see transaction “EDIT” on page 540.

Enabling and disabling the persistent environment

To enable the persistent environment, you must be running DataInterchange in a CICS/ESA region and the SYSPROF profile member for the region must indicate that you want the persistent environment established.

To disable the persistent environment, use the SYSPROF profile to indicate that you want the persistent environment disabled.

For more information about the SYSPROF profile member, see “CICS performance profile (SYSPROF-P2 for CICS only)” on page 256.

Using multiple regions

Certain installations may allocate a DataInterchange database for use by a single DataInterchange region. In these cases, when online updates are made to database records that are kept in the DataInterchange Persistent Environment, the old copy of the record is removed from the data space.

Transaction EDIG addresses installations that have multiple DataInterchange regions/environments sharing a single DataInterchange database. Transaction EDIG starts automatically at set intervals to check if an update has been made recently that may affect the persistent environment. If an update has been made, then all records that match the updated record are removed from the persistent environment; that is, if a control string is deleted, then all matching control strings are removed from the persistent environment.



NOTE: In installations where multiple regions/environments share a single DataInterchange database, all online updates to the database are made from the same region/environment.

Reserved TS and TD queues

By convention, DataInterchange 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 DataInterchange uses this TS and TD queue naming convention.

TS queues that might require additional processing

The first time DataInterchange initializes in a CICS region, it creates a TS queue named EDITV00. This TS queue holds frequently used translation and validation tables. For more information, refer to the *DataInterchange Administrator's Guide*.

If you modify any of the following tables using the DataInterchange 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 DataInterchange, there is a corresponding TS queue. For more information on EDI envelope standards, refer to the *DataInterchange Administrator's Guide*. The following TS queues are used for envelope standards:

Envelope type: TS queue:

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. DataInterchange 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, DataInterchange/MVS-CICS 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 DataInterchange/MVS-CICS in TD queues. These TD queues can be defined as intra- or extra-partitioned. The DataInterchange/MVS-CICS 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 DataInterchange 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 *DataInterchange Administrator's Guide*.

TS queues used by import only

These TS queue names are currently reserved by DataInterchange 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

DataInterchange 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 519.

EDI2

If TD queue EDI2 is defined as an intrapartition TD queue, DataInterchange will use this queue as the Expedite/CICS administrative response file. Expedite/CICS writes network acknowledgments to this file, and the file is used as input by DataInterchange for updating network status.

If EDI2 is not defined, a unique TS queue is used as the administrative response file for each Information Exchange mailbox. These TS queues are always appended to and never cleared.

If EDI2 is defined, both Expedite/CICS and DataInterchange 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 DataInterchange. If network acknowledgments do not have to be archived, then EDI2 should be

used so that network acknowledgments are not reprocessed. Network acknowledgments are reprocessed when EDI2 is not defined and multiple update status requests are executed in an active CICS region.



NOTE: If network acknowledgments are being received and processed continuously through continuous receive, EDI2 is not used even if it is defined. In this case, Expedite/CICS passes DataInterchange one network acknowledgment at a time in unique TS queues. On completion, DataInterchange 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 DataInterchange/MVS-CICS, Expedite/CICS and Information Exchange

DataInterchange provides communication routines that interface to Expedite/CICS. EDI interchanges can be sent to and received from the Information Exchange network using this interface. Additional features include an event-driven continuous receive process and network acknowledgment reconciliation. Trading partner and mailbox (requestor) profiles should reference network profile IINCICS for this environment.

The communication flow involves these components:

- DataInterchange
- Expedite/CICS
- Information Exchange

All three components maintain control information to track activities, so it is essential to keep them synchronized. Expedite/CICS and Information Exchange are standalone products that can be operated independently from DataInterchange. Therefore, you can execute operational commands without the participation of DataInterchange, preventing DataInterchange from updating control information. Use these commands sparingly to prevent out-of-sync conditions. The two fundamental programming layers (or sessions) between the three components that must be kept in sync are: the low level Information Exchange session through which all commands and data pass, and the higher level continuous receive session which controls the environment and connectivity.

Information Exchange sessions

The Information Exchange session is the fundamental session started between an application program and a particular Information Exchange user ID (mailbox). All commands and data sent out from, or received into, a mailbox must pass through this session. You can have only one active Information Exchange session per mailbox. For example, if you are using Expedite Base/MVS, this means that you cannot have concurrent Information Exchange sessions for the same mailbox on two different Expedite/CICS regions, or one on CICS and another in batch. You can have any number of Information Exchange sessions from various environments if each session pertains to a different mailbox.

Most of the processing that occurs during the Information Exchange session is performed by Expedite/CICS and Information Exchange. DataInterchange is involved only at the application level, but it is still important that the session be initiated and maintained by DataInterchange to ensure proper execution of all DataInterchange network facilities. The first time DataInterchange receives a request from you to perform any network function for a given mailbox, it automatically issues the appropriate session start command to Expedite/CICS. Expedite/CICS then starts the session with Information Exchange and an access key is assigned. All three components now have a record of this particular Information Exchange session. After the Information Exchange session is started, DataInterchange performs the network function you requested and returns control to you.

For efficiency reasons, DataInterchange 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, DataInterchange 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 538.

In theory, one Information Exchange session could be used indefinitely. However, remember that you can only have one active Information Exchange session for a given mailbox, and until this Information Exchange session is ended, no other environment, not even a batch job using Expedite Base/MVS, should access this mailbox. If you have multi-environment communication needs with a particular Information Exchange mailbox, you must end the Information Exchange session from CICS before executing communications in batch. DataInterchange will only request an Information Exchange session end when your application issues a CLOSE MAILBOX request. To keep DataInterchange, Expedite and Information Exchange synchronized, use the DataInterchange CLOSE MAILBOX command to end your Information Exchange sessions. If you are using the continuous receive function of DataInterchange, 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 534 for details.



ATTENTION: Expedite/CICS provides a facility to start and end Information Exchange sessions directly, but you should not use Expedite/CICS facilities to manage Information Exchange sessions used by the DataInterchange application.

The following is for your information and to prevent inadvertent execution of certain Expedite/CICS functions. The Expedite/CICS terminal transaction LG01 can be used to start and end Information Exchange sessions. After you enter your mailbox account and user ID, LG01 checks internal control information. If an Information Exchange session is already active for the mailbox you identified, and the **Force User to log on** option is set to **No**, the main menu displays and Expedite/CICS does not issue a Session Start to Information Exchange. If there is no active Information Exchange session, LG01 prompts you for an Information Exchange password, and if entered, Expedite issues a Session Start to Information Exchange. Once the session starts, the main menu is displayed.

Accessing the mailbox from LG01 is one way to determine if Expedite acknowledges that there is an active Information Exchange session. You can also determine if Expedite is acknowledging an active Information Exchange session by attempting to end an Information Exchange session from LG01 (type **X** to logoff). A confirmation panel is displayed that shows any active network functions that must be completed before the Session End is issued. If one of the active functions is Continuous Receive, it ends the session, causing an out-of-sync condition between Expedite/CICS and DataInterchange. If you confirm the request to end the session, Expedite issues a Session End to Information Exchange. (Expedite is unaware that DataInterchange originally started the session.) For more information, see “Continuous receive sessions” on page 534.

When Expedite/CICS ends the Information Exchange session, DataInterchange is prevented from updating its internal control information, which brings about an out-of-sync condition among the components. However, DataInterchange has built-in logic to re-issue a Session Start if Expedite returns the message HI421: SESSION PROFILE DOES NOT EXIST. Therefore, ending an Information Exchange session only has serious implications when a continuous receive session is ended.

If you find that you inadvertently logged off and no Continuous Receives were active, do not log back on to LG01 to restart the session, because DataInterchange must initiate Information Exchange sessions. DataInterchange will automatically restart a session when you perform your next network function.

As a general rule, leave the session settings on LG01 alone. If you are prompted to enter your password upon entry, log off when you leave. If you were not prompted for a password, do not log off when you leave (use PF3 or type End). For more information on using LG01, refer to the *Expedite/CICS Display Application User's Guide*.

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 DataInterchange application. It may 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 DataInterchange through Expedite/CICS. It causes an out-of-sync condition between the components. For more information regarding Information Exchange Administrative Services, refer to *Using Information Exchange Administration Services*.

Information Exchange session cleanup

An Information Exchange session problem may prohibit starting or ending an Information Exchange session. The most common symptom of an Information Exchange session error is the Expedite/CICS error SDIERR RESPONSE CODE 00008. If the error is encountered by DataInterchange, it is embedded in a logged DataInterchange 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 may mean that whatever reset your session did not end its own session. Enter Y to confirm the reset, then go to Step 2.



NOTE: If you find you must perform this procedure frequently, you may want to inspect the session trace available on IE/SERV to identify the LU name that is resetting your session.

2. Use the Expedite/CICS IDLT CICS terminal transaction to delete the Information Exchange session control information from the Expedite/CICS control file. The format of the IDLT transaction follows:

```
IDLTacct user ID
```

The account number (*acct*) must be eight characters long, left-justified and concatenated to the IDLT transaction ID. If your account is shorter than eight characters, pad the remaining positions with spaces.

No validation is performed on your user ID, so it is important to enter your user ID correctly.

3. If your Information Exchange session had any active continuous receive sessions, try the failed network function again or go to the next section, Continuous receive session. As discussed earlier, DataInterchange will automatically reissue an Information Exchange Session Start.

Continuous receive sessions

The continuous receive (CR) session is a high-level application programming layer that is added on top of an Information Exchange session. A continuous receive session cannot run without an Information Exchange session. An overview of the continuous receive process follows.

When a CR session is active, EDI messages entering the mailbox that match the continuous receive criteria are automatically received. Information Exchange automatically starts Expedite/CICS and passes the EDI envelope. Expedite/CICS then writes the envelope to a unique TS queue, linking to DataInterchange to pass the name of this TS queue. DataInterchange processes the envelope using the processing options specified in the applicable continuous receive profile, after which DataInterchange returns control to Expedite/CICS. Expedite/CICS is then free to process the next envelope for this continuous receive session. Exactly when DataInterchange returns control to Expedite/CICS depends on whether the TD queue EDI1 is defined as intrapartitioned. If EDI1 is defined, DataInterchange and Expedite run synchronously. In other words, Expedite/CICS will not process the next envelope until DataInterchange returns control. If the EDI1 TD queue is not defined, Expedite/CICS and DataInterchange run asynchronously where Expedite is free to start processing the next envelope before DataInterchange has finished with the first. For more information, see “Effects of defining the EDI1 TD queue” on page 519. 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 DataInterchange 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 DataInterchange Utility commands START CONTINUOUS RECEIVE and STOP CONTINUOUS RECEIVE. Also, the DataInterchange Utility provides the REPORT CONTINUOUS RECEIVE STATUS command for reporting the statuses of your continuous receives. For more information on these commands, see “Continuous receive” on page 12.



NOTE: This information applies only to continuous receive processing associated with Expedite/CICS and Information Exchange. Continuous receive processing driven by user-written applications is not started or stopped in this manner. For detailed information, see “Continuous receive interface (CICS only)” on page 565.

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, DataInterchange invokes Expedite/CICS, passing it the continuous receive selection criteria. For criteria options, see “Continuous receive selection criteria” on page 517. Expedite/CICS assigns a key to identify the continuous receive session, passes the criteria and key to Information Exchange, and then returns control to DataInterchange. DataInterchange 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 (DataInterchange, 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, DataInterchange invokes Expedite/CICS, passing the key for the particular continuous receive session. Expedite/CICS issues the end continuous receive request to Information Exchange. The continuous receive may not be stopped at this point. If a receive is taking place, Expedite/CICS returns control to DataInterchange where DataInterchange and Expedite/CICS can complete asynchronously. However, DataInterchange checks Expedite/CICS control information until an indicator shows that the continuous receive session has ended. At that point, DataInterchange issues a message confirming a successful end.

Although Expedite/CICS provides a facility to start and stop continuous receive sessions directly, using these Expedite/CICS facilities regularly to manage your DataInterchange continuous receive sessions is not recommended.

The following is for your information and to prevent inadvertent execution of certain functions in Expedite/CICS. The Expedite/CICS terminal transaction, LG01, can be used to start and stop any active continuous receive sessions by using Option 1, Work with Receive Data. DataInterchange will not be aware of any activities performed in LG01, so use this feature carefully.

Continuous receive session cleanup

Before attempting any cleanup, generate a continuous receive status report. For more information, see “Reporting continuous receive status” on page 13. Based on the reported status, you can initiate a reasonable cleanup.

DataInterchange 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 DataInterchange. It does not affect the state of Expedite/CICS or Information Exchange. Therefore, it is essential that Expedite/CICS (using LG01) be used to terminate all continuous receive sessions before executing EDIZ. If Expedite/CICS has not terminated the continuous receive and EDIZ is executed inadvertently, EDI envelopes will be lost because DataInterchange is unaware of the receive requests for which Expedite/CICS is now passing data.

The format of the EDIZ command is:

```
EDIZ [membername]
```

if *membername* is not specified, all continuous receive members are requested.

Identifying unrecoverable continuous receive sessions

The following table describes some methods to help you to identify an unrecoverable continuous receive session:

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 DataInterchange is not processing the data.	It appears DataInterchange 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, DataInterchange 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 may be caused by a delay in Expedite/CICS to end a continuous receive session.
EDIR or EDIS fails, a negative response is returned, and DataInterchange logs an error that an SDIERR occurred.	EDIR or EDIS will fail immediately if Expedite/CICS cannot communicate with Information Exchange because of an Information Exchange session problem. DataInterchange 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, then no Information Exchange session is active for your ID. Enter your password to access the LG01 main menu. Expedite/CICS will start a new Information Exchange session, which may reactivate continuous receives that were not receiving data previously. This is expected. Continue with Step 2.
2. Choose Option 1, Work with Receive Data, and then choose Option 3, Stop Continuous Receive. If no entries are displayed, go to Step 6. If entries are displayed, go to Step 3.
3. Issue a stop request by all continuous receive sessions. Use the Enter key to refresh the screen.
 - a. If all sessions change status from STARTED to STOPPED, go to Step 6.
 - b. If all sessions do not change status from STARTED to STOPPED, go to Step 4.
4. Sign on to IE/SERV. Choose Option 1, Work with Profiles. Fill in the appropriate account and user ID for the mailbox in question and choose Option 8, Reset a User's Session. Enter **Y** to confirm the reset and go to Step 5.
5. Use the Expedite/CICS IDLT CICS terminal transaction to delete the Information Exchange session control information from the Expedite/CICS control file. The format of the IDLT transaction is:

```
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 DataInterchange 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 may have been lost. You can use LG01 to determine which, if any, envelopes were not processed by DataInterchange.

Choose the Receive option from the main menu, and then Option 5, View List of Completed Receives. Each EDI envelope received by Expedite/CICS will have an entry. A status of **RECEIVED** is normal. A status of **E-HI999** means that DataInterchange did not accept the envelope. DataInterchange 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 DataInterchange and Expedite/CICS

cannot be re-synchronized. However, you can use the View option to obtain the interchange control number, the interchange sender, and so on. If your mailbox has archiving turned on, you can use IE/SERV to retrieve each lost envelope. An envelope retrieved from archive is receivable just as if it had been sent.



NOTE: When an HI999 error is returned, DataInterchange issues an error message to the EXPLOG1 data set associated with your CICS region: CR0120
DATAINTERCHANGE CONTROL RECORD MISSING FOR RECEIVED DATA,
WRITING DATA TO EXPDERR FILE.

Program list table considerations

There are two post-initialization PLT programs that affect continuous receive processing. All customers should use the Expedite/CICS PLT program called EXPOSTRT that re-establishes both Information Exchange and continuous receive sessions automatically during CICS start-up, even after an immediate shutdown. The second program is an optional DataInterchange 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.

DataInterchange also provides an optional pre-termination PLT program named EDICRSP that issues an EDIS to stop all continuous receives, and also issues a CLOSE MAILBOX request against all involved mailboxes. During a normal shutdown, EDICRSP ends all continuous receive sessions and all Information Exchange sessions. This releases all mailboxes using continuous receive so that other communication applications requiring the same mailboxes can execute while the CICS region is down. This will prevent the Information Exchange session problems discussed earlier. However, if you have no other Information Exchange communication applications, or if the other applications you run do not use the same mailbox, you do not need to release the Information Exchange session, and PLT program EDICRSP is not needed.

As you can see, the pre-termination program EDICRSP and post-initialization program EDICRTS go hand in hand. If you use EDICRSP for pre-termination, then you should use EDICRTS for post-initialization to start those continuous receive sessions ended by EDICRSP. EXPOSTRT does not start these sessions, because they will not be active after shutdown is complete.

The following are sample post-initialization and pre-termination PLTs.

```
*-----*
* Sample post-initialization PLT including program EDICRTS,      *
* the continuous receive start program, and the necessary        *
* Expedite/CICS PLT program 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 DataInterchange transaction, EDIT, *
* is shown along with DB2 attachment program, DSNCCOM1, to     *
* provide the proper order for the entires.                    *
*-----*
PLTZZ      DFHPLT TYPE=INITIAL,SUFFIX=ZZ

           DFHPLT TYPE=ENTRY,PROGRAM=EDICRSP
           DFHPLT TYPE=ENTRY,PROGRAM=EDIXSOX
           DFHPLT TYPE=ENTRY,PROGRAM=DSNCCOM1
           DFHPLT TYPE=FINAL
           END

```

Note: If you will be using the EDICRSP PLT program, it is necessary to add an XLT entry for Expedite/CICS transaction ISC2. A sample entry follows:

```

*-----*
*                               DataInterchange for CICS        *
*                               *                               *
* The following XLT (CICS Transaction List Table) entries are to be *
* applied to any CICS region where DataInterchange 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.

DataInterchange-supplied transactions

DataInterchange provides a set of CICS transactions to perform many different functions. The following summary describes each transaction and its associated function.

Transaction	Description
EDIA	The online administrative transaction. Use this transaction to customize in the CICS environment.
<div>  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. </div>	

Transaction	Description
EDIB	<p>The DataInterchange Utility transaction. If you run the DataInterchange Utility asynchronously, use EXEC CICS START to start this transaction. You can also start EDIB with DataInterchange as part of continuous receive processing.</p> <p> 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.</p>
EDID	Used internally by DataInterchange. A background transaction used to update network status. EDID does not have a user interface.
EDIE	<p>Used internally by DataInterchange. 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.</p> <p> 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.</p>
EDIG	A transaction automatically invoked periodically when using the persistent environment.
EDIM	A transaction that executes the DataInterchange Message Broker to transform data.
EDIQ	The transaction that gains control from MQSeries transaction CKTI when data is received into an MQSeries queue that has trigger processing (continuous receive) associated with it.
EDIR	The online or background transaction used to initiate continuous receive requests.
EDIS	The online or background transaction used to terminate continuous receive requests.
EDIT	<p>A transaction that can be used to terminate the DataInterchange environment. The DataInterchange environment is usually terminated by placing EDIXSOX in the CICS pre-termination PLT, but you can also terminate the DataInterchange environment by running EDIT. Certain pieces of information are maintained from the start of the first DataInterchange activity within a CICS region, continuing throughout the session, until the DataInterchange environment is terminated. You can use EDIT to reset this information. EDIT shuts down transactions EDIG and EDIX, shuts down the persistent environment, releases CSD storage, and deletes the EDICSDA and EDITV00 TS queues. Using EDIT to perform these functions is not part of standard procedures.</p>
EDIV	The installation verification transaction. For more information on this transaction, refer to the <i>DataInterchange Installation Guide</i> .

Transaction	Description
EDIW	The DataInterchange Utility invocation transaction. For more information, see “Using EDIW to invoke the DataInterchange Utility” on page 544.
EDIX	<p>A background transaction used to perform some Transaction Store updates for other DataInterchange transactions. This transaction does the following: acquires a transaction handle and deletes an envelope for a DEENVELOPE command with the DUPENV(Y) option.</p> <p>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 DataInterchange 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.</p> <p>Eventually, as all DataInterchange 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.</p> <div>  <p>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.</p> </div>
EDIZ	The online or background transaction used to clean up continuous receive problems.

Performance monitor user exit

DataInterchange/MVS-CICS 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 APPDEFS profile. This function applies only for DataInterchange for CICS.

For performance monitoring to be meaningful, it should be done in tandem with Expedite/CICS. Typically, DataInterchange and Expedite/CICS work asynchronously with one another (except for continuous receives). To understand resource utilization involved in data translation/transmission, the DataInterchange and the Expedite/CICS components must be considered together. This can be achieved by using the performance monitor user exit capabilities of both products. The COMMAREA format passed to each is the same. “Format of performance monitor commarea” on page 542 describes which fields are filled in by DataInterchange and which fields are filled in by Expedite/CICS.

On the send side, DataInterchange could invoke the user exit during enveloping and, subsequently, Expedite/CICS could invoke it after sending. These two disparate invocations could be associated using the interchange key fields:

- Sender ID
- Receiver ID
- Control number
- Direction

On the receive side, Expedite/CICS could invoke the user exit after receiving the interchange and subsequently DataInterchange could invoke it during deenveloping. If more than one envelope is generated or deenveloped during a single execution of DataInterchange, 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

Name	Offset	Length	Type	Owner	Description
RESPCODE	0	5	Char	Exp	Response code
RESPTYPE	5	8	Char	DI/Exp	DataInterchange - 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

Name	Offset	Length	Type	Owner	Description
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 DataInterchange

Using EDIW to invoke the DataInterchange Utility

YOU can use the DataInterchange for CICS transaction EDIW to invoke the DataInterchange 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 "DataInterchange Utility control information" on page 507.

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 may 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 may be referenced in a command file.

DataInterchange 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
-----
```

Interfacing to other networks and applications

This chapter describes the interfaces between your application, DataInterchange, and the network. You can use this chapter to develop programs that communicate with applications and networks.

All communication requests by DataInterchange or by applications using DataInterchange are made through the Communications API. For more information, see “Communication services” on page 407. DataInterchange includes support for multiple networks and allows you to write programs to add support for any network that is not supported. DataInterchange uses profiles that define the network (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 DataInterchange 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 4, “Using the DataInterchange application program interface”.

DataInterchange provides support for 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. DataInterchange 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 on page 551 illustrates how an API request from an application flows into the DataInterchange 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 (ADAMCTL) profile. For more information, see “Message handler” on page 559.
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 DataInterchange that you want to use.

The seven logical names you may 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 519.
- VANICICS is for CICS only, fills in several pre-defined data blocks, and then passes control to the user-supplied network program specified in the **Network program** field of the network profile. For more information, see “Special communications routine for CICS” on page 560.
- PTTOPT identifies the point-to-point network. For more information, see “Point-to-point networks” on page 549.
- VANIMQ is for the exclusive use of MQSeries send and receive.
- VANIINB1 changes a value of **G** in the **RECVTYP** field of the CMCB to a value of **N**. For more information, see “Communication Control Block (CMCB)” on page 621.
- 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:

Value	Meaning
<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 MVS 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 NETOP entries contain instructions for pulling data from various sources along with literals to create commands. For more information, see “Building network commands” on page 554.. The network program may be written in any language with any attributes because it is accessed using an MVS 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 DataInterchange. 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 (ADAMCTL). For more information, see “Message handler” on page 559.

Point-to-point networks

DataInterchange for MVS 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 554.

Activating point-to-point connections

To activate point-to-point communications, follow these steps:

1. Add a member to the user program information profile (ADAMCTL) 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 (**CMDVAL**) of this member identifies the functions that are supported by the point-to-point connection.

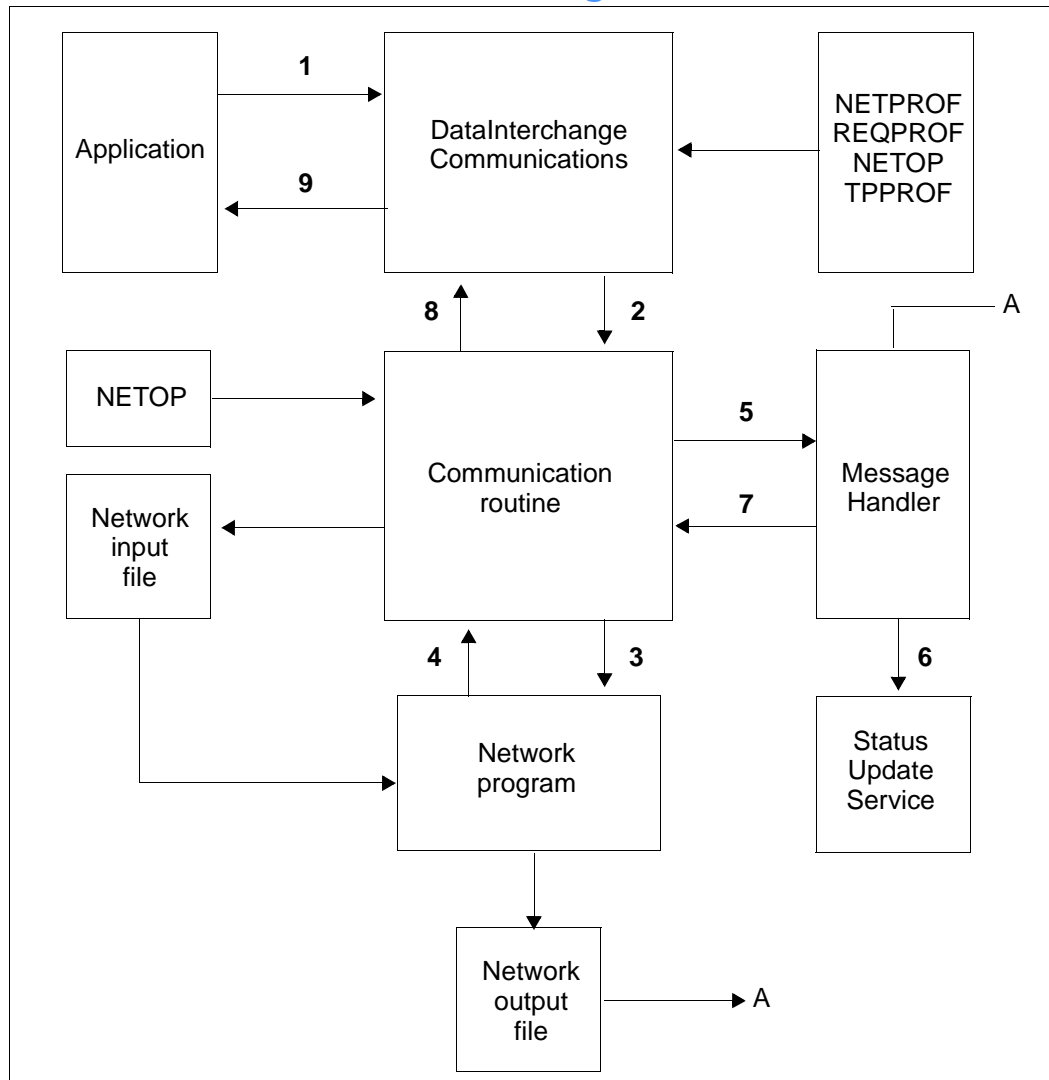
You can create the point-to-point member by copying the member FSUPPORT and changing the network ID and CMDVAL values. For details about the CMDVAL literal, see “FSUPPORT member” on page 558.

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 551) with the following exceptions:

1. The network program is called as a DataInterchange exit rather than using an MVS ATTACH. You must define the program in the ADAMCTL profile and write the program in a language supported by DataInterchange. See Chapter 5, “Exit routines,” 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.

Application-to-network flow diagram



The numbers in the diagram refer to the following sequence of events:

1. An application program requests DataInterchange communication services through the API defined in “Communication services” on page 407. 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 (FXXZccc). The communications routine name is a logical name and must be one of the following:

VANINFC	VANICICS (not applicable for this flow)
VANIINB1	VANIMQ (not applicable for this flow)
VANEXPV4	
PTTOPT	
GEISVAN	

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). NETOP members contain instructions for building commands, which are literal values combined with data from the other control blocks and profile members. The commands are written to a file as specified by the **Network input file** field from the network profile.

If transaction data is being sent (function code **211**), the file of transactions is scanned and each interchange in the file is updated to have a status of **Send started (46)**. The following fields are also associated with the interchange:

- Message name (MSGNAME)
- Message user class (ENAME)
- Message sequence number (SEQNUM)
- Network acknowledgment expected (ACKIND)
- Send date
- Send time

When commands have been built and written to the network input file and status has been updated, the program identified in the **Network program** field is invoked using the MVS ATTACH facility. The parameters passed to the network program are specified in the **Network parameters** field of the network profile. Network parameters are not passed to the network program when the communications routine is GEISVAN.

4. This is a synchronous operation where the communication routine waits until the network program is finished. When that program returns control, it is assumed that the operation is completed or that the network program is responsible for making sure the operation completes.

A return code of **0** from the network program indicates success. Return codes **1-4** indicate no serious errors. A return code of less than **0** or greater than **5** indicates the function failed completely.



NOTE: A return code of **0** is assumed when the communication routine is GEISVAN.

5. If sending transaction data is requested, the communication routine updates the status in the Transaction Store of each interchange in the file based on the return code from the network program. A return code of **0** results in a status of **Send requested (48)**. Return codes **1-4** result in a status of **Sent with errors (41)**. Any other return code results in a status of **Send request error (42)**.



NOTE: If the communication routine is VANIINB1 and the return code from the network program is zero (**0**), the communication routine does not update the status. In this case, the message handler must update the status of each interchange in the file. This is because VANIINB1 is targeted for the Expedite network program and when Expedite has a return code of **0**, it has written an entry for each interchange to the network output file (OUTMSG). The message handler for Expedite can read these records and update the status of each interchange.

However, if the return code from the network program is not **0**, VANIINB1 sets the status of each interchange to **Send request error (42)**, and the message handler must update the status for those interchanges that were successfully sent. With a nonzero return code, the assumption is that the network program may not have been able to (or chose not to) complete its processing. Therefore, the network output file (OUTMSG) may 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 (FXXZccc). For more information about message handler programs, see “Message handler” on page 559.

6. If the network output file contains status information about previously sent data, the message handler should use the status update service to update the status information in the Transaction Store. For more information, see "Update status services" on page 429.
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.

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 407.

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, "DataInterchange Control Blocks."

- 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

The table below 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 **CMDLSEQ**, **CMDFPOS**, and **CMDFLEN** fields of the NETOP profile member. The data for the command record field can either be a literal value (**CMDFVAL**) or it can be a value from one of the DataInterchange control blocks (**BLKNAME** or **BLKPOS**).

The keys to a profile member are the network ID (**NETID**), network commands ID (**NETOP**), and network commands field sequence number (**FLDSEQ**).

Network commands profile definition

Label	Length	Type	Description
NETID	8	Char	Network ID
NETOP	8	Char	Network commands ID
FLDSEQ	8	Char	Network commands field sequence number
BLKNAME	8	Char	Name of block from which data is taken
BLKPOS	4	Char	Block position
CMDLSEQ	4	Char	Command line sequence number
CMDFPOS	4	Char	Command field position
CMDFLEN	4	Char	Command field length
CMDFVAL	58	Char	Command field value

Network commands profile field descriptions

NETID

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.

NETOP

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.

FLDSEQ

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.

BLKNAME

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)	CMDFVAL field contents

BLKPOS

The starting position of the source data within the block identified in **BLKNAME**. The position is relative to 1.

CMDLSEQ

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.

CMDFPOS

The starting position in the command record where the source data should be moved, or an asterisk to use the current field position.

CMDFLEN

The length of data to move from the source location (**BLKNAME** or **BLKPOS**) to the command record (**CMDLSEQ** or **CMDFPOS**). The length of a command record is defined in the **Input record length** field of the network profile.

CMDFVAL

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 example

The table below describes the network command entries used to build the commands for sending X12 standard transactions over the network. The **NETID (IINR4)**, **NETOP (SENDX12)**, and **FLDSEQ (1-n)** fields are not shown in the table.

BLKNAME	BLKPOS	CMDLSEQ	CMDFPOS	CMDFLEN	CMDFVAL	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 **netid** FSUPPORT. The literal in the command field value (**CMDFVAL**) of the member indicates which functions are supported. The first byte corresponds to **FQUEUED**; the second byte corresponds to **FMSGGS**, and so on. Values entered in the FSUPPORT member must be in uppercase. The table below describes how to use the FSUPPORT bytes.

Byte name:	A value of Y specifies that:
FQUEUED	Queued functions are supported.
FMSGGS	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.
FAACK	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**, then 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 5, “Exit routines.”

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 (ADAMCTL). An ADAMCTL entry is not needed for the network profile entries that are distributed by DataInterchange as these programs are defined in internal DataInterchange 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 **NPSEVER** and **NPERRCD** fields.
- **CMTPPDB**
The message handler sets the **ACCTNUM** and **USERID** fields.
- **NPDB**
The message handler is invoked by the communication routine after the network program has returned control. The message handler processes the responses the network program has written to the network output file as follows:
 - Processes data in the network output file.
 - Calls the status update service to update the status in the Transaction Store, if status update messages are contained in network output file. For more information, see “Update status services” on page 429.
 - Sets the error code in the **NPERRCD** field and the severity code in the **NPSEVER** field of the CMCB if errors are noted in network output file. It also sets the **ZCCBRC** field in the CCB to indicate that the **NPSEVER** and **NPERRCD** fields have meaning. The communication routine uses the **NPERRCD** and **NPSEVER** values when logging a VN1015 error message.

Special communications routine for CICS

A special communications routine for the CICS environment (logical name VANICICS) uses a LINK interface to pass control to the network program and the message handler. If you do not plan to use Expedite/CICS to communicate with trading partners, this special communications routine may 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 VANICICS 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 *DataInterchange Administrator's Guide*. The following is a sample network profile member definition using VANICICS.

```

Profile ID      . . . : NETPROF

NETWORK ID     . . . : LU62NET
NETWORK NAME   . . . : NETWORK USING VANICICS
COMMO ROUTINE  . . . : VANICICS
NETWORK PROGRAM . . . : LU62PGM
NETWORK PGM PARMS :
NET CMD INPUT FILE:
NET CMD REC LENGTH:
TRANS DATA QUEUE : EDIQDAT
TRANS DATA REC LEN:
TIME ZONE      . . . . :
SYSTEM TYPE    . . . . :
SYSTEM LEVEL   . . . . :
MSG TEXT HEADER . . . :
NET CMD OUT FILE :
MSG PROCESSING PGM: LU62MSG
NET SEQUENCE NBR : 00000
  
```

The fields defined in the above sample are important and are described below. The other fields are ignored by DataInterchange and may be used as you like. 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 DataInterchange 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. |
| COMMO ROUTINE | To use VANICICS, it is critical that you enter VANICICS in this field to tell DataInterchange that whenever a function handled by a communications routine is invoked, VANICICS is given control. |

NETWORK PROGRAM	The name of your network program. DataInterchange 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 PROCESSING PGM	The name of your message processing program. DataInterchange will CICS LINK to this program during the processing of an UPDATE STATUS request. VANICICS does not invoke this program after other network functions, such as send or receive. The usual function of this program is to update the status in the Transaction Store.




NOTE: For VANICICS, the **MSG PROCESSING PGM** field contains the name of a physical load module that DataInterchange 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 DataInterchange use the network commands profile member, VANICICS does not. The interface to your network program is similar to the MVS 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 as follows:

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

Name:	Offset:	Length:	Type:	Address of control block:
TPPADDR	16	4	Bin	CMTPPDB
				NOTE: The CMTPPDB block only contains data 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 which are passed in the **ZFCBFUNC** field of the FCB. Valid values are:

With this code: Your network is requested to:

- 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 DataInterchange via the CICS RETURN command. DataInterchange expects your network program to indicate success or failure back to DataInterchange. Your network program does this by overwriting the incoming COMMAREA with meaningful information. The overwritten COMMAREA should be formatted as follows:

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 as follows:

In this field:	Enter:
RESPONSE	HI000
SEVERITY	00
FILLER	Blanks

No data received (network)

Your network program may issue a receive request but there may not be any data available to be received. DataInterchange 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 DataInterchange that a receive was issued but no data was returned by setting the fields in the COMMAREA as follows:

In this field:	Enter:
RESPONSE	HI000
SEVERITY	04
FILLER	Blanks

Error occurred (network)

Your network program may encounter many different errors. You can indicate to DataInterchange 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 as follows:

In this field:	Enter:
RESPONSE	Any five-character string besides 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, DataInterchange 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 as follows:

Name:	Offset:	Length:	Type:	Address of control block:
SNBADDR	0	4	Bin	SNB
CCBADDR	4	4	Bin	CCB
FCBADDR	8	4	Bin	FCB
CMCBADDR	12	4	Bin	CMCB
NPDBADDR	20	4	Bin	NPDB
TPPADDR	16	4	Bin	CMTPPDB
REQADDR	24	4	Bin	REQDB

The message handler is invoked and will always receive the same function code (252). Therefore, the **ZFCBFUNC** field in the FCB is not important. When your message handler gains control, its primary task is to update the status of the Transaction Store. You do this by using the update envelope status API documented in "Update status services" on page 429. 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 return control to DataInterchange via the CICS RETURN command. DataInterchange expects your message handler program to indicate success or failure back to DataInterchange. Your message handler program does this by overwriting the incoming COMMAREA with meaningful information. The overwritten COMMAREA should be formatted as follows:

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:

In this field:	Enter:
RESPONSE	HI000
SEVERITY	00
FILLER	Blanks

Error occurred (message handler)

Your message handler may encounter many different errors. You can indicate to DataInterchange 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:

In this field:	Enter:
RESPONSE	Any five-character string besides HI000 . The string 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 message handler program to aid in problem determination. This response code is included in the VN1022 message.
FILLER	Blanks.

Continuous receive interface (CICS only)

The continuous receive facility was designed to work with Expedite/CICS and Information Exchange, but it can be used with user-written applications. The intent of this interface is to house DataInterchange 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 DataInterchange 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 Expedite/CICS and processing that interacts with user-written receive applications as follows:

- Only continuous receives associated with Expedite/CICS are started and stopped with CICS transactions EDIR and EDIS, or with PLT programs EDICRTS and EDICRSP. Continuous receives associated with user-written receive applications are not started or stopped through DataInterchange. The user-written receive application passes the name of the continuous receive profile member to DataInterchange when a continuous receive occurs. Since this is all

the control information DataInterchange 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 DataInterchange.

- 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, DataInterchange updates the management reporting receive statistics database. If one is not supplied, management reporting statistics are not updated.
- The mailbox (requestor) profile ID is mandatory for continuous receives associated with Expedite/CICS. The continuous receive profile member ID must be a maximum of 8 characters long. Continuous receive profile member IDs associated with Expedite/CICS can have a maximum length of 16 characters.

Invoking the continuous receive interface

When a user-written receive application receives EDI data from a source (such as a leased line, different VAN, remote TD queue), it can invoke the continuous receive interface to have this EDI data processed. The EDI data must reside in a TS queue for DataInterchange 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 following format.

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 DataInterchange 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 DataInterchange to process the incoming data. For more information on the fields in the profile, refer to the continuous receive profile definition in the *DataInterchange Administrator's Guide*.
- ENVFILEx** The name of the TS queues holding the EDI data that DataInterchange is to process. The TS queues may 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 489.
- USRFLD** The data in this field is moved to the **USRFLD** field in the DataInterchange Utility control block. This data is then available to response programs started subsequently. For more information, see "DataInterchange Utility control information field descriptions" on page 509.

When DataInterchange has completed its processing, it returns control to the user-written receive application. DataInterchange updates the COMMAREA indicating success or failure. The user-written receive application is informed whether DataInterchange accepted responsibility for the data. DataInterchange 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 DataInterchange 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 DataInterchange to process. The first seven characters in the returned COMMAREA will contain one of the following values:

- HI00000** DataInterchange was able to accept ownership of the data. Whether or not DataInterchange processing was successful, the continuous receive response application must detect and report any DataInterchange processing errors.
- HI77712** The continuous receive profile member indicated that translation and deenveloping was not to be performed. The only DataInterchange 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 ENVFILEx should be backed up.
- HI88812** DataInterchange processing abended in the EDIB transaction. The data in ENVFILEx 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 ENVFILEx should be backed up.

If one of the errors above is encountered, DataInterchange attempts to log an error message to the EXPL TD queue. EXPL can be directed to a destination where 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. For inbound processing, you can use the literal keyword **&THANDLE** to map the DataInterchange archive key to the SAP IDOC. For outbound processing, you can use the special DataInterchange variable name **DISAPSEQ** to save the IDOC record sequence number on the first error encountered. The value of **DISAPSEQ** is captured in the SAP status record to indicate the first record in error.

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 DataInterchange 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. DataInterchange 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 DataInterchange 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 | ■ SEND |
| ■ ENVELOPE AND SEND | ■ TRANSLATE AND ENVELOPE |
| ■ REENVELOPE | ■ TRANSLATE AND SEND |
| ■ REENVELOPE AND SEND | ■ TRANSLATE TO STANDARD |
| ■ RESTART SEND | |

Inbound processing and SAP status

You can control the inbound processing of SAP status for functional acknowledgments by setting the **SAPUPDT** keyword to **Y** on Utility **PERFORM** statements. When you specify the **SAPUPDT** keyword and the Transaction Store is active, functional acknowledgment status is collected in the database.

To activate SAP status tracking, you must specify the **SAPUPDT** keyword on the following **PERFORM** commands:

- DEENVELOPE
- DEENVELOPE AND TRANSLATE
- RECEIVE AND DEENVELOPE
- RECEIVE AND TRANSLATE

SAP status codes supported by DataInterchange

The SAP status codes supported by DataInterchange are:

Status code	Description
04	Error within control information of EDI subsystem
05	Error during translation process
06	Translation successful
09	Error during interchange handling
10	Interchange handling successful
11	Error during dispatch
12	Dispatch successful
16	Functional Acknowledgment positive
17	Functional Acknowledgment negative
22	Dispatch successful, acknowledgment still due

You must provide the translation usage/rule required to perform the translation of the inbound and outbound IDOCs. The literal keyword &THANDLE is provided to map the DataInterchange archive key to the SAP IDOC for inbound processing, and the DataInterchange 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.

Extracting SAP status records

You can use the SAP STATUS EXTRACT command to extract SAP status records from the DataInterchange 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 matched the selection criteria.
- 796** Records were truncated in the output file.

Removing SAP status records

You can use the SAP STATUS REMOVE command to remove SAP status records from the DataInterchange database. The only error code returned by this command is:

- 796** Error occurred during processing.

Interfacing with MQSeries

You can exchange data with your trading partners using MQSeries queues in network profiles. DataInterchange prohibits the use of MQSeries queues as the target of the envelope process or as input to the deenvelope process. MQSeries queues can be used as part of a logical network where enveloped data is sent to and received from MQSeries queues. DataInterchange provides a sample network profile member named MQSAMP to assist you with the setup of this local network, as described in the following steps:

1. Ask your MQSeries Administrator to define the MQSeries queue you plan on using.
2. Define DataInterchange MQSeries Queue profile members for the queues your MQSeries administrator has created. For more information, refer to the *DataInterchange Administrator's Guide*.
3. On the Update Profile Member Panel, complete the fields listed below as follows:

In this field:	Enter:
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 DataInterchange to refer to this network.
Network name	A descriptive name of the network, such as Sample MQSeries Network. This field is optional.
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.
Network program	EDIMQSR . The physical name of the send and receive program. This program is invoked by the communication routine to process requests.
Network parameters	The parameters required by the network program. This field must match the DataInterchange MQSeries Queue profile member names created in Step 2. The SENDMQ keyword should be followed by the DataInterchange MQSeries Queue profile member name where you send data. The RECEIVEMQ keyword should be followed by the DataInterchange MQSeries Queue profile member name where you receive data. Both parameters must be delimited by blanks. If you are performing one-way communication with your trading partner, only the keyword/value combination of that direction is required.
Network input file	The ddname that contains the commands written for the network program to process. For INNB41, the name is INMSG. This field is not used for a point-to-point connection or CICS.
Input rec length	The length of records in the network input file. Generally, the record length is 80. For a point-to-point connection or CICS, leave this field blank.

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	The length of records in the TD queue. The communication routine provided by DataInterchange (VANIINB41) ignores this field and uses the logical record length you allocated for the file. For DataInterchange/MVS-CICS, 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 .
Time zone	The time zone for your location. The network specifies the allowable codes. Refer to the <i>DataInterchange Administrator's Guide</i> for a list of codes.
System type	This field is not used by any currently supported network.
System level	This field is not used by any currently supported network.
Msg text header	This field is not used by any currently supported network.
Net output file	The logical name of the file containing the network's responses to the command input file. For example, for IINB41 the name is OUTMSG. For a point-to-point connection, leave this field blank.
Message handler	The name of the program that processes messages and network acknowledgments from the network. When you send a transaction, if your network returns a message to indicate the success or failure of your send request, the message handler program interprets the returned message and updates the status of the interchange. For example, for IINB41 and IINB42, the name is INB1MSG; and for GEIS, the name is GEMSGHL. For a point-to-point connection, type NONE in this field.
Network sequence	A number that DataInterchange increments and assigns to all outbound documents. You can set or reset the value at which sequential numbering begins. The default is 00000 .
Net acks file	The name of a file (MVS ddname) where the network should write the network acknowledgements when you request a status update. The network acknowledgements are read and evaluated by the message handler program.
Script name	The name of a set of instructions that your communication software can use to process requests associated with this network profile member. The set of instructions would be part of the communication software package, and not part of DataInterchange. This field is optional.



You can override the script name in the utility commands for sending transactions.

4. Your new network profile member can be used in trading partner profiles and mailbox (requestor) profiles. For more information see Appendix A, "DataInterchange Control Blocks".
5. If you are using continuous receive, you may also perform the following tasks:
 - When using MQSeries queues with an application file, you must identify the associated MQSeries queue profile member in the data format fields (specifically, the **Application file name** and **Application file type** fields).
 - When executing a translation, you can read data to or write data from MQSeries queues (for print, report, exception, and so on).
 - When using an MQSeries queue for your application data file, you must enter the MQSeries queue profile member name and the MQ type in at least one of the following places:
 - Application file name and file type in the data format
 - Application file name and file type in the Receive Usage panel
 - Any PERFORM command that uses the **APPFILE** and **APPTYPE** keywords

For more information, see "Continuous receive using MQSeries" on page 516.

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 DataInterchange 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.

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, then you must upload the DTD file to the host. The file may be stored as either a PDS member or an HFS file.

When you use the DTD to validate a source (input) XML document, upload the DTD as text if you specify **XML EBCDIC** as **Y** (default). If you specify **XML EBCDIC** as **N**, then the DTD file must contain an XML declaration (`<?xml...>`) that includes the appropriate encoding type for the file. For more information, see the section on XML encoding considerations. For data transformation maps, DataInterchange does not validate output (target) XML documents, and so does not use DTDs for these. 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), then 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. 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, then the value is assumed to be an HFS path and the base name is appended to the path information.
2. If a DTD alias file (DD:DTDALIAS) is allocated, then the base name is first looked up in the alias file. (See below for more information)
3. If the base name is found in the alias file, then the alias name is combined with the XMLDTDS value to determine the filename of the DTD file. The search for the base name in the alias file is not case sensitive.
4. If no alias is found or the DTDALIAS file is not allocated, 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 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. The DTDALIAS file is searched for **MyPO.dtd**. For this example, we will assume it is not found.
3. 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
```

4. If 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 name making the DTD file name:

```
EDIUSER.MYDTDS(MYPO)
```

The default process may 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, the DTDALIAS file 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 **LongDTDName21.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
```

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. 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

The XML processor uses the XML Toolkit for OS/390 to parse the XML data. This parser supports many different character encodings, and follows the XML standards for auto-detection of the character encoding. However, this can sometimes cause problems when dealing with EBCDIC data.

According to the XML standard, if the XML document is not in UTF-8 (similar to ASCII for most commonly used characters) or UTF-16 format, it must begin with an XML encoding declaration (<?xml...>). For EBCDIC data, the *encoding*= attribute must be present and indicate which

encoding type is in use. If the XML data was generated as ASCII data and then converted to EBCDIC format, it is likely that the *encoding=* attribute would not be added or updated to reflect the EBCDIC conversion.

Additionally, the new line character (EBCDIC x'15) is not recognized as a valid white space character by the XML standard. However, in MVS this is often inserted into files as a record separator by editors, upload applications and other MVS applications.

To resolve these problems, the XML processor in DataInterchange can instruct the parser to override the default (auto-detected) encoding type for the XML document with a special encoding type: **ebcdic-xml-us**. This causes the parser to ignore the *encoding=* attribute in the XML declaration and use this special type instead. This encoding type is based on the IBM1140 codepage and will treat any newline characters as a carriage-return character (x'0A'), which is a valid XML white space character.

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 may contain new line characters but may not contain the correct encoding type in the XML declaration.

When you specify **XMLEBCDIC** as **N**, the XML processor determines the encoding type for both the data and external DTDs based on the normal XML auto-detection rules. Use this setting if your input XML data is in a format other than EBCDIC, such as ASCII or UTF-16. Since the value of **XMLEBCDIC** is also used to control the interpretation of the DTD files, your DTDs must contain an XML declaration (`<?xml...>`) with the proper encoding type.

When generating XML data, the XML data is always created as EBCDIC. No encoding type is specified in the default XML prolog. If you want to send the data in another format, such as ASCII or UTF-16, the translation must be done outside of DataInterchange. 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 may 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.

DataInterchange Control Blocks

This Appendix describes DataInterchange control blocks, including block layouts and field descriptions. The control blocks are provided in softcopy format in the following distribution load libraries.

Library Name	Samples
EDI.V4R1M0.SEDIASM1	Assembler and COPY
EDI.V4R1M0.SEDICBL1	COBOL and CBLCPY
EDI.V4R1M0.SEDICCC1	C and H
EDI.V4R1M0.SEDIPLI1	PL/I and INCLUDE

The names and descriptions of members that are provided in each of the libraries are listed below. Use these members as a starting point for copy books that are tailored for use by your installation.

Library member	Description
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

Library member	Description
EDIDETPI	Trading Partner Information Record
EDIFCB	Function Control Block (FCB)
EDIFFC	DataInterchange Utility C Record
EDIFFD	DataInterchange Utility D Record
EDIFFDU	DataInterchange Utility Control Information Block
EDIFFE	DataInterchange Utility E Record
EDIFFG	DataInterchange Utility G Record
EDIFFI	DataInterchange Utility I Record
EDIFFQ	DataInterchange Utility Q Record
EDIFFT	DataInterchange Utility T Record
EDIFFZ	DataInterchange 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
EDIVNNP	VANICICS commarea to network program

In addition, the EDI.V4R1M0.SEDICBL1 library contains the following example programs.

Library member:	Creates a report using the:
EDISAMR1	ENVELOPE DATA EXTRACT command. Based on Application Control Values.
EDISAMS1	ENVELOPE DATA EXTRACT command. Based on Application Control Values.
EDISAMT1	TRADING PARTNER CAPABILITY DATA EXTRACT command. About the transaction capability of trading partners.

Service Name Block (SNB)

The SNB allows you to access DataInterchange 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 DataInterchange table.

The same interface used to access DataInterchange API services is used when DataInterchange invokes a user-written exit program. User exits are defined and given logical names during the customization process. The ADAMCTL profile is then updated so that the physical load module and the implementation language can be associated with the logical name. ADAMCTL 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 DataInterchange. If you want a reentrant program, do not assign storage for the SNB from static storage.

The table below describes the layout of the SNB.

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 DataInterchange
ZSNBFLG1	27	1	Char	Reserved for DataInterchange
ZSNBFANC	28	4	Bin	Reserved for DataInterchange

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, DataInterchange 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 DataInterchange service being requested. Service names must be left-justified and padded with blanks. DataInterchange 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 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	SYNCPPOINT services

ZSNBNDX

Used internally by DataInterchange 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 DataInterchange. First flag byte.

ZSNBFLG1

Reserved for DataInterchange. First flag byte.

ZSNBFANC

Reserved for DataInterchange. First anchor position.

Common Control Block (CCB)

The CCB is used by DataInterchange to maintain status information about the current DataInterchange session. A DataInterchange 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 DataInterchange 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 DataInterchange. 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 DataInterchange 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 DataInterchange. During processing, the CCB is modified by DataInterchange. If you want a reentrant program, the storage for the CCB must not come from a static storage area.

The table below describes the layout of the CCB.

Name	Offset	Length	Type	Description
ZCCBLL	0	2	Bin	CCB length
ZCCBID	2	2	Bin	Reserved
ZCCBEYE	4	8	Char	Dump eyecatcher
ZCCBRC	12	4		Return code
ZCCBERC	16	4		Extended return code
ZCCBSID	20	8		System ID
ZCCBUID	28	8		User ID
ZCCBAID	26	8		Application ID
ZCCBCID	44	8		Error module ID
ZCCBXFID	52	2		Function ID
ZCCBLPID	54	6		Language profile ID
ZCCBCPID	68	4		Code page ID
ZCCBRSV	72	4		Reserved
ZCCBCCXP	76	4		Pointer to CCB extension
ZCCBCABP	80	4		Pointer to common area block
ZCCBDBID	84	4		MVS DB2 subsystem ID
ZCCBDBPL	88	8		MVS DB2 plan/AIX DB2 alias
ZCCBDBUI	96	8		AIX DB2 user ID
ZCCBDBPW	104	18		AIX DB2 password
ZCCBDRSV1	122	26		Reserved for DataInterchange
ZCCBRSV2	148	460		Reserved for DataInterchange

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 DataInterchange initialization call, DataInterchange 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. For more information about return codes, see Appendix B, "DataInterchange condition codes and API return codes."

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. For more information about return codes, see Appendix B, "DataInterchange condition codes and API return codes."

ZCCBSID

DataInterchange provides the system ID (a static value of **EDIV00**) during the initialize service API request.

ZCCBUID

The user ID field activated by DataInterchange during the initialize service API request. For a TSO user, this is the TSO user ID. For an MVS 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 APPDEFS profile entry to use for this DataInterchange 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 DataInterchange for error logging, but the information it contains can be used in problem determination.

ZCCBXFID

Used internally by DataInterchange for error logging, but the information it contains can be used in problem determination.

ZCCBLPID

The language profile ID. During initialization, DataInterchange verifies that a **LANGPROF** 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 DataInterchange. 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 MVS DB2 subsystem ID. Do not alter after initialization.

ZCCBDBPL

The MVS DB2 plan/AIX DB2 alias. Do not alter after initialization.

ZCCBDBUI

The AIX DB2 user ID.

ZCCBDBPW

The AIX DB2 password.

ZCCBRSV1

Reserved for DataInterchange. Do not alter after initialization.

ZCCBRSV2

Reserved for DataInterchange. 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. The table below describes the layout of the FCB.

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 DataInterchange
- 2** Terminate DataInterchange
- 5** Switch APPLID

Translation services (TRANPROC)

- 131** Production translate-to-standard
- 111** Test translate-to-standard
- 212** Production deenvelope and translate-to-application
- 211** Test deenvelope and translate-to-application
- 213** Translate a specific transaction to application
- 1000** End translation

Enveloping services (TRANPROC)

- 1** Return interchange header
- 2** Return group header
- 3** Return transaction header
- 215** Envelope transactions
- 214** Deenvelope transactions
- 990** Close and queue current envelope
- 991** Issue COMMIT request

Data extraction services (TRANPROC)

- 216** Retrieve detailed data
- 217** Retrieve transaction image
- 218** Retrieve functional acknowledgment image
- 219** Retrieve transaction acknowledgment image

Communications services (COMM)

- 211** Send transactions
- 221** Send files
- 232** Receive
- 233** Cancel
- 252** Process network acknowledgment
- 300** Return file name
- 110** Queue transaction data

Update Status Services (TRANSSRV)

- 210** Update using envelope key (account number, user ID)
- 211** Update with transaction handle
- 212** Update using alternate key (account number, user ID)
- 213** Update using envelope key (qualifier, receiver ID)
- 214** Update using alternate key (qualifier, receiver ID)
- 215** Update using envelope key (trading partner nickname)

SYNCPOINT services (SYNCSERV)

- 1** Initialize SYNCPOINT services
- 2** Request a COMMIT
- 3** Request a ROLLBACK

Translator Control Block (TRCB)

The translator control block (TRCB) is the primary control block used to convey information between an application and DataInterchange when using translation, enveloping, and data extraction services. For more information about these services, see “Translation services” on page 308, “Enveloping services” on page 365, and “Data extraction services” on page 402.

The table below describes the layout of the TRCB.

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
TRNNUM	204	4	Bin	Number of transactions in processed the interchange
SEGNUM	208	4	Bin	Number of segments in processed in the interchange

Name	Offset	Length	Type	Description
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
RSRVD2	464	1	Char	Reserved
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
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

Name	Offset	Length	Type	Description
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
GRESAGENCY	655	2	Char	Group responsible agency
RSRVD3	657	12	Char	Reserved
THXCTL	669	14	Char	Transaction control number
TTC	683	6	Char	Transaction ID
TVER	689	6	Char	Transaction version
TREL	695	6	Char	Transaction release
TST	701	10	Char	Transaction segment total
LASTINENV	711	1	Char	Last transaction in interchange
XACFIELD	712	35	Char	Application control number
FABUILT	747	1	Char	Functional acknowledgment built
QGTNUM	748	4	Bin	Total number of groups in the interchange
QTTNUM	752	4	Bin	Total number of transactions in the interchange
QSTNUM	756	4	Bin	Total number of segments in the interchange
QGT	760	6	Char	Total number of groups in the interchange
QTT	766	6	Char	Total number of transactions in the interchange
QST	772	10	Char	Total number of segments in the interchange
FASPM	782	8	Char	Functional acknowledgment standard profile member
ENVCHK	790	1	Char	Envelope status check
TRNSTAT	791	1	Char	RAWDATA transaction status
APTYPE	792	2	Char	Application file type
TSKEY	794	10	Packed	Packed transaction handle
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

Name	Offset	Length	Type	Description
BNDLFLAG	861	1	Char	Bundle flag
RAWDATA	862	1	Char	RAWDATA flag
ENVLDELAY	863	1	Char	Delayed enveloping flag
TRXACCEPT	864	1	Char	Transaction acceptable flag
TRABORT	865	1	Char	Translator abort flag
FILEID	866	8	Char	User-defined file ID
DSNAME	874	56	Char	Physical data set name
QNETID	930	8	Char	Network ID
QPTTOPT	938	1	Char	Point-to-point network flag
QSRPGM	939	1	Char	Send/Receive program flag
QDDNAME	940	8	Char	ddname for transaction file
FUNACKFLE	948	8	Char	ddname for functional acknowledgment file
QTPNICK	956	16	Char	Trading partner nickname
QRC	972	2	Bin	Queuing return code
QERC	974	2	Bin	Queuing extended return code
ERRCDES	976	20	Bin	First 10 error codes
INMEMTRANS	996	2	Bin	In-storage transactions
NOCOMMIT	998	1	Char	Commit flag
SCOPE	999	1	Char	Recovery scope
TPNICK	1000	16	Char	Trading partner nickname
CONCATENATE	1016	1	Char	Concatenation flag
ASSERTLVL	1017	1	Char	Assertion level
RAWDATAOUT	1018	1	Char	RAWDATA wanted for output
FIXEDTRX	1019	1	Char	Fixed-to-fixed transaction flag
MRREQID	1020	16	Char	Management reporting requestor ID
ERRFILTER	1036	80	Char	Initial error filter
FFILEID	1116	8	Char	File ID for fixed translations
VARTRACE	1124	1	Char	Variable trace wanted flag
EXPTRACE	1125	1	Char	Expression trace wanted flag
SSEGVAL	1126	1	Char	Service segment validation flag
TRXFACODE	1127	1	Char	Functional acknowledgment code generated for the transaction
IUSEREXIT	1128	8	Char	User exit for envelopes
IUSERAREA	1136	4	Bin	User area for IUSEREXIT
IUSERACCESS	1140	1	Char	User exit access type
IUSERTYPE	1141	2	Char	User exit program type
MAPCHAIN	1143	1	Char	Mapchain active flag

Name	Offset	Length	Type	Description
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 ERRCDDES array flag
FARC	1148	4	Bin	Functional acknowledgment return code
FAERC	1152	4	Bin	Functional acknowledgment extended return code
SAPUPDT	1156	1	Char	SAP updates requested
SAPTRX	1157	1	Char	SAP transaction indicator
SAPCLIENT	1158	3	Char	SAP client
SAPDOCNUM	1161	16	Char	SAP document number
SAPSEQ	1177	6	Char	SAP error record sequence number
ROUTCODE	1183	3	Char	Generic send usage routing code
FAEREQ	1186	1	Char	Functional acknowledgment envelope file required
BOUNDARY	1187	1	Char	Incremental translation flag
SUSBLKP	1188	4	Bin	CICS SUSPEND block pointer
CUSERDATA	1192	256	Char	User data area
APPLTPID	1448	15	Char	Application trading partner ID
EXTENDC	1464	1	Char	Extend the C record flag
VAXFLAG	1465	1	Char	Pageable translation flag
GDATE8	1466	8	Char	Group envelope 8-byte data
RECOVBAD	1474	1	Char	Recover from bad EDI standard data
RSRVD4	1475	61	Char	Reserved for DataInterchange

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 MVS, 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.

RSRVD2

Reserved.

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.

GRESAGENCY

If an EDIFACT group is being processed, this field contains the group controlling agency extracted from the group header field UNG09. For enveloping functions, this field is also an input field that identifies the controlling agency to be used when building the group header segment.

If an ICS, UCS, or X12 group is being processed, this field contains the group responsible agency extracted from the group header field GS07. For enveloping functions, this field is also an input field that identifies the responsible agency name to be used when building the group header segment.

RSRVD3

Reserved.

THXCTL

If a transaction with a **CN** data type is being processed, this field contains the transaction control number extracted from the transaction header.

TTC

If a transaction with a **TC** data type is being processed, this field contains the transaction or message ID extracted from the transaction header.

TVER

If a transaction with a **VR** data type is being processed, this field contains the transaction version extracted from the transaction header.

For enveloping functions, this field is also an input field that identifies the transaction version that should be used when building the transaction header segment.

TREL

If a transaction with an **LV** data type is being processed, this field contains the transaction release extracted from the transaction header.

For enveloping functions, this field is also an input field that identifies the transaction release that should be used when building the transaction header segment.

TST

The character representation of **SEGNUM**.

LASTINENV

Indicates whether this transaction is the last transaction of the interchange. Valid values are:

- | | |
|----------------|---|
| Y | Last transaction of the interchange |
| (other) | Not the last transaction of the interchange |

XACFIELD

The application control number that uniquely identifies the transaction to the application. This is a returned value for all translation and enveloping functions. Which field contains the application control number is indicated in the data format with an **AC** data type. This field can be overridden by providing up to eight fields that can be concatenated to form an application control number in the map ID.

FABUILT

Indicates the envelope type, if any, for the functional acknowledgment. You can control when functional acknowledgments are deenveloped. For more information, see the **ENVLDELAY** field on page 605. Valid values are:

E	UNB/UNZ
G	GS/GE
I	ICS/ICE
S	Stored, but not enveloped
T	STX/END
U	BG/EG
X	ISA/IEA

QGTNUM

When queued, the total number of groups in the interchange (**EJECT** value of **Q** during translating to an EDI standard or during enveloping). When received, the total number of groups in the interchange (**NEWENV** value of **Y** during translating to a EDI standard or during deenveloping). See **QGT** for a character representation of this field.

QTTNUM

When queued, the total number of transactions in the interchange (**EJECT** value of **Q** during translating to an EDI standard or during enveloping). When received, the total number of transactions in the interchange (**NEWENV** value of **Y** during translating to an EDI standard or during deenveloping). See **QTT** for a character representation of this field.

QSTNUM

When queued, the total number of segments in the interchange (**EJECT** value of **Q** during translating to an EDI standard or during enveloping). When received, the total number of segments in the interchange (**NEWENV** value of **Y** during translating to a EDI standard or during deenveloping). See **QST** for a character representation of this field.

QGT

The character representation of **QGTNUM**.

QTT

The character representation of **QTTNUM**.

QST

The character representation of **QSTNUM**.

FASPM

Used only by the translator. The standard profile member used in building the service segments for the functional acknowledgment.

ENVCHK

Indicates that the translator should verify a transaction's status during an envelope function for the intent of the envelope operation. Valid values are:

- 1** Envelopes transactions, and rejects this transaction if it has been enveloped before.
- 2** Reenvelopes transactions, and rejects this transaction if it has not been enveloped before.
- (other)** Does not check the status of the transaction.

This field becomes important if multiple jobs can be enveloped concurrently and you want to ensure that a transaction is not enveloped and sent twice.

TRNSTAT

Indicates the status of the transaction. Your application can use this field in conjunction with the **RAWDATA** field to determine the status of a raw data transaction or to determine the next action expected from the application. Valid values are:

- I** The data format has defined a structure that starts a transaction but the structure has not been received. Data is ignored until the starting transaction structure is recognized.
- S** The starting transaction structure has been received. If a starting structure is not defined, this value is set when the first structure is received.
- C** A transaction is in progress and the structure passed was neither the starting nor the ending structure.
- R** A transaction is in progress and a structure that defines the start of a transaction was received. The data is ignored. The application must first call the translator with an **EJECT** value of **Y** to process the current transaction, and then call the translator again with the same data to start a new transaction.
- Y** A transaction is in progress and a structure recognized as the ending structure is received. The transaction is ready to be processed and a call to the translator is required with an **EJECT** value of **Y**.

APTYPE

The type of application file identified in the **APPFILE** field. Applies only to CICS. Valid values are:

- PG** Program
- TD** TD queue identified by the first 4 characters of **APPFILE**
- TM** TS queue (in main storage)
- TS** TS queue (in auxiliary storage) identified by **APPFILE**
- TX** Transaction code identified by the first 4 characters of **APPFILE**

TSKEY

The Transaction Store handle value for a transaction in packed format. This is an input field for the translate-specific and envelope functions.

This field indicates the transaction to be translated or enveloped. For all other functions, this field is an output field that indicates the transaction key value just processed.

If handling the key in a packed format is a problem for your program, **TSKEYU** contains the key value in an unpacked format. If **TSKEY** contains all blanks or binary zeros, the value in the **TSKEYU** field is used.

TSKEYU

The Transaction Store handle value for a transaction in unpacked format. If the **TSKEY** field contains blanks or binary zeros, this field becomes the input field for the translate-specific and envelope functions.

This field indicates the transaction to be translated or enveloped. This field is a returned value for all translation, enveloping, and deenveloping functions. If this field is used as an input field, its packed value is returned in the **TSKEY** field.

MAPKEY

A returned value that identifies the map used to translate the data.

BATCHID

An input field used for all translation functions. This field is an indexed field in the Transaction Store database and provides an efficient method for retrieving transactions from the store. It can also be used to identify all the transactions that were processed during a particular execution of a job. If you do not specify a value, the system date and time are used to create a default value of *DDHHMMSS*.

ENVLDATE

The earliest date that a transaction is eligible to be enveloped and sent. If you do not specify a date, the transaction is eligible immediately, unless other conditions indicate that it is not eligible (such as a transaction in held status). This is an input field for translating application data to an EDI standard format only.

TRXLIFE

The number of days the transaction should remain in the Transaction Store before it is eligible to be purged. If this field contains zero, a default value of 30 days is used. The default value is returned in this field. This is an input field for all translating and deenveloping functions.

IMGLIFE

The number of days the transaction image should remain in the Transaction Store before it is eligible to be purged. If this field contains zero, a default value of 30 days is used. The default value is returned in the this field. This is an input field for all translating and deenveloping functions.

A transaction's life span (**TRXLIFE**) and the transaction's image life span (this field) are always the same. However, you should still set a value for this field to ensure correct processing.

HOLDFLAG

Indicates whether a transaction should be held. This is an input field for functions that add transactions to the Transaction Store. Valid values are:

- | | |
|----------------|-------------------------------|
| Y | Holds the transaction |
| (other) | Does not hold the transaction |

BNDLFLAG

Specifies the bundle status for transaction. This is an input field for translating application data to an EDI standard format. If the current trading partner is not using functional groups (the **Functional group** field in the trading partner profile contains N), changes in data that would generally cause a new group to be created are ignored, and the bundles are not terminated. Valid values are:

- | | |
|----------------|--|
| Y | The start of a bundle. This transaction becomes the controlling transaction for the bundle. All transactions that follow become part of the bundle until: <ul style="list-style-type: none">• 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. |

For translating EDI standard data to an application format, valid values are:

- | | |
|----------------|---|
| 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. |



NOTE: If RAWDATA processing has been requested and was possible, DataInterchange writes the data to the application file in raw data format. However, if RAWDATA processing was not requested or was not possible, C and D records are written to the application file. Your application program can write the records in any desired format, regardless of the setting of the **RAWDATA** field.

ENVLDELAY

Indicates whether enveloping should be delayed for transactions being translated. In any case, the transaction (or functional acknowledgment) and its image are saved in the Transaction Store. The value set for this field on the first call of a session is used for the entire session.

For translating EDI standard data to an application format, valid values are:

- Y** Does not envelope the transaction at the same time it is translated
- (other)** Envelopes the transaction at the same time it is translated

For the deenveloping and translating a file, valid values are:

- Y** Does not envelope functional acknowledgments at the same time it is translated
- (other)** Envelopes the functional acknowledgments at the same time it is translated

For related information about functional acknowledgments, see the **FUNACKFLE** field description on page 606.

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**).

OPTOPT

Indicates whether the network associated with the interchange that was written (**EJECT** is **Q**) is a point-to-point network. Valid values are:

- Y** Point-to-point network
- N** Not a point-to-point network

QSRPGM

Indicates whether the network associated with the interchange that was written (**EJECT** is **Q**) has a send/receive program defined. Valid values are:

- Y** A send/receive program is defined
- N** No send/receive program is defined

QDDNAME

The ddname of the file to which the interchange was written (**EJECT** is **Q**).

FUNACKFLE

The ddname (TSQ name in CICS) to which a functional acknowledgment interchange should be written if enveloping is not being delayed (**ENVLDELAY** is **N**). Applies only during deenveloping or translate-file functions. This field overrides the default file name in the **Trans data queue** field from the network profile.

OTPNICK

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, see “Translator Error Codes” on page 614.

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 page 607. 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.
(other)	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 DataInterchange 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 DataInterchange 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 614.

The following errors may not be filtered and attempts to do so will be ignored:

106-110, 117-118	501, 505-509
204, 207-210	601, 602, 604
301, 303, 305, 308-334	900-999
401, 403, 405-411	

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 DataInterchange use only).

EXPTRACE

Indicates whether an expression level translator trace is wanted (for DataInterchange use only).

SSEGVAl

Indicates the level of service segment validation that should take place. The service segments are the segments used when a transaction is enveloped (ISA, GS, ST, UNB, UNH, UNT, and so on). If you do not specify this field, no validation is performed. You can use the **SERVICESEGVAl** keyword on the **PERFORM** command to set this value. Valid values are:

- 1** Validates service segments for syntax only. This includes checking for mandatory data that is missing, as well as data elements that are too large or too small.
- 2** In addition to checking syntax, validates the values in the service segment data elements according to their types (only dates and times are validated), and if a validation table has been specified, checks the value of the data element against the validation table.

TRXFACODE

The type of functional acknowledgment that was generated for this transaction. Valid values are:

- A** Accepted
- R** Rejected
- E** Accepted with errors

IUSEREXIT

During send processing, the name of a user exit that is given to each interchange as it is created by DataInterchange. During receive processing, the name of a user exit that will provide the interchange to be processed by DataInterchange.

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 DataInterchange user exit program defined in the 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, then it also must be used on the TRANSLATE-TO-APPLICATION or RETRANSLATE-TO-APPLICATION commands to select those transactions stored by the deenvelope process.
- (other)** 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 DataInterchange issue periodic SUSPENDs, AICA abends can be eliminated. Valid values are:

- Y** A suspend block has been allocated and its address is in SUSBLKP.
- (other)** No suspend block has been allocated.

CLRERRS

Indicates whether the **ERRCDES** array should be reset before processing continues. Valid values are:

- Y** Resets the **ERRCDES** array before processing
- (other)** Does not reset the **ERRCDES** array before processing

FARC

The return code from the translator for the functional acknowledgment translation done during deenvolving. For more information about return codes, refer to *DataInterchange Messages and Codes*.

FAERC

The extended return code from the translator for the functional acknowledgment translation done during deenvolving. For more information about extended return codes, refer to *DataInterchange Messages and Codes*.

SAPUPDT

Indicates that SAP updates are requested (for DataInterchange use only).

SAPTRX

The SAP transaction indicator (for DataInterchange use only).

SAPCLIENT

The SAP client code (for DataInterchange use only).

SAPDOCNUM

The SAP document number (for DataInterchange use only).

SAPSEQ

The SAP error record sequence number (for DataInterchange use only).

ROUTCODE

A 3-character generic routing code provided by the application and used by DataInterchange to select a generic send usage/rule. A blank specifies a default generic send usage/rule.

FAEREO

Indicates whether a functional acknowledgment file is required. Valid values are:

- | | |
|----------------|---|
| Y | A functional acknowledgment is required. An error is produced if the file is not available. |
| (other) | A functional acknowledgment is not required. |

BOUNDARY

Indicates whether incremental translation should be used. During regular (non-incremental) translation, this field should be blank. For more information about incremental translation, see “Outbound incremental translation” on page 336.

SUSBLKP

The suspend block pointer. Applies only to the CICS API. If the API program supplies a suspend block (32 bytes, initialized with binary zeroes), this field should contain the address of the block and the **SUSBLKF** field should contain **Y**.

CUSERDATA

On send translation of C and D records, the user area provided by the user. On receive translation, this value comes from the DataInterchange variable DICUSERDATA. This field can be modified by user exits and is copied to the C record before the C record is output by DataInterchange. 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 may 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 page 152.

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 DataInterchange.

Translator Error Codes

The tables below identify the error codes generated by the translator.

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

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

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

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
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

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

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 308, “Enveloping services” on page 365, and “Data extraction services” on page 402.

The table below describes how the block is used for all functions and the initialization requirements for each.

Function code:	Initialization requirements:
1	A BLKLEN of 16 is sufficient. For more information, see “Retrieve interchange header API” on page 399.
2	A BLKLEN of 16 is sufficient. For more information, see “Retrieve group header API” on page 400.
3	A BLKLEN of 16 is sufficient. For more information, see “Retrieve transaction header API” on page 400.
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 311.
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 338.
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 341.
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 384.
215	A BLKLEN of 16 is sufficient. For more information, see “Envelope API” on page 368.
216	A BLKLEN of 16 is sufficient. For more information, see “Retrieve detailed data API” on page 404.
217	A BLKLEN of 16 is sufficient. For more information, see “Retrieve transaction image API” on page 406.
218	A BLKLEN of 16 is sufficient. For more information, see “Retrieve transaction acknowledgment image API” on page 406.
219	A BLKLEN of 16 is sufficient. For more information, see “Retrieve functional acknowledgment image API” on page 406.

Function code:	Initialization requirements:
990	A BLKLEN of 16 is sufficient. For more information, see “Close and queue interchange API” on page 382.
991	A BLKLEN of 16 is sufficient. For more information, see “Issue commit API” on page 399.
1000	A BLKLEN of 16 is sufficient. For more information, see “End translation/enveloping API” on page 384.

The following table describes how to define 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 308, “Enveloping services” on page 365, and “Data extraction services” on page 402.

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.

The table below describes shows how the block is used for all functions and the initialization requirements for each.

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 399.
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 400.
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 400.
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 311.
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 311.
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 341.
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 350.
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 341.
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 384.
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 368.

Function code:	Initialization requirements:
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 404.
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 406.
218	The DATA field contains the image record. The DATALEN field identifies the number of bytes in the record. For more information, see “Retrieve transaction acknowledgment image API” on page 406.
219	The DATA field contains the image record. The DATALEN field identifies the number of bytes in the record. For more information, see “Retrieve functional acknowledgment image API” on page 406.
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 382.
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 399.
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 transaction/enveloping API” on page 384.

The following table describes how to define 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 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)

The table below 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 NETOP profile are relative to **1**.

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 RECL
SCRIPT	82	8	Char	Script name
ENAME	90	8	Char	Envelope name
MSGNAME	98	8	Char	Message name
FILENAME	106	56	Char	File name
CANS	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
NPESCDE	193	5	Char	End session response code
NPERRCD	198	5	Char	Error code

Name	Offset	Length	Type	Description
NPSEVER	203	2	Char	Error severity
BLKTYPE	205	1	Char	Block type
FQUEUED	206	1	Char	Queued functions indicator
FMSGGS	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
FAACK	217	1	Char	Network acknowledgment indicator
FSYSMSG	218	1	Char	System messages indicator
FRCVBTP	219	1	Char	Receive by trading partner indicator
FRESTART	220	1	Char	Restart indicator
FNOUSERID	221	1	Char	Account number indicator
FACCTSEP	222	1	Char	Value used to separate account and user ID
DDCOLON	223	3	Char	DD: string
RESERV3	226	2	Char	Reserved
ADMTYPE	228	2	Char	Administrative response file type
UNIQID	230	8	Char	Unique ID returned by the network program
SAPUPDT	238	1	Char	SAP update flag
FSENTNET	239	1	Char	Skip send requested status flag
RESERV4	240	14	Char	Reserved for DataInterchange

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 page 623.

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 .
RECVMFILE	Receives non-EDI file for function codes 0132 and 0232 .
RECVX12	Receives X12 standard transactions for function codes 0132 and 0232 .
RECVEDI	Receives EDIFACT or UN/TDI standard transactions for function codes 0132 and 0232 .
RECVUCS	Receives UCS standard transactions for function codes 132 and 232 .
CANCEL	Cancels delivery of a file or message for function codes 133 and 233 .
NO-NETOP	Processes a file of responses for function code 252 , but does not invoke a network program. Used if you have network acknowledgment responses in a file that have not been processed.

REQID

The key for a member of the mailbox (requestor) profile. Communication gets the network ID from the mailbox (requestor) profile member. Your application must supply the network ID if it does not supply the requestor ID. For related information, see **NETID** on page 623.

SEQNUM

The sequence number assigned to a transaction, file, or message. For sending or queuing to send, Communication returns the sequence number. For canceling a file or message, your application supplies the sequence number of the file or message to be canceled.

CONTRCV

Indicates whether continuous receive should be active. Applies only to CICS. Valid values are:

(blank)	Starts single receive
C	Starts continuous receive
E	Ends continuous receive

FTYPE

Indicates the destination file type for received transactions. Valid values are:

MQ	MQSeries queue
TM	TS queue (main)
TS	TS queue (auxiliary)
PG	Program

FILERCV

Indicates whether a file was received. Valid values are:

Y	File was received.
(other)	File was not received.

RESERV2

Reserved.

CLRFILE

Indicates when the Communications service is to clear the file you are sending. Applies only to immediate sending of EDI standard transactions and non-EDI files. Valid values are:

Y	Clears the file after successfully sending the transaction data or non-EDI file.
U	Always clears the file.

DATAFMT

Indicates whether data is in binary format. The first character of the **Envelope name** field defines the receive class used by the Expedite product interface. Expedite will set the data type to A or E, depending on which Expedite product you are using. Valid values are:

B	Binary data
(blank)	Not binary data (default)

ACCTYP

For sending or canceling data, your application must supply a value of **D**, which indicates that the trading partner ID is an account number or user ID (Network reference: DESTTYP). For receiving data, your application must supply one of the following values (IN reference: SRCTYP):

(blank)	Receives from any source
D	Receives from a trading partner identified by an account number or user ID

DATATYP

The type of file name supplied in the **FILENAME** field. Valid values are:

- | | |
|----------|---------------|
| A | Data set name |
| D | ddname |

For sending, your application must supply the data type. You can use a ddname only for immediate sends.

For receiving, the communications service supplies the data type if:

- The application does not supply the data type.
- **NETOP** contains RECVX12, RECVEDI, or RECVUCS.
- Immediate receive (function code **232**) is requested.

The Expedite parameters for network profiles is FILEID.

RECVTYP

The type of messages to receive. Valid values are:

- | | |
|----------------|--|
| G | Receives only the first message or file that matches the trading partner nickname, the envelope name, or both. For example, receive only X12 envelopes from partner ABC. |
| (blank) | Receives all messages and files that meet the receive criteria. |

The Expedite parameter for network profiles is ALLFILES.

DCIND

The delivery class. Valid values are:

- | | |
|----------------|---|
| (blank) | Normal delivery |
| P | High priority delivery |
| I | Express delivery. May require that the receiving partner be signed on to the network when the data is sent. |

The Expedite parameter for network profiles is PRIORTY.

ACKIND

The type of acknowledgment requested. For sending, valid values are:

- | | |
|----------------|---------------------------|
| (blank) | No acknowledgments |
| R | Receipt only |
| D | Delivery only |
| B | Both receipt and delivery |
| A | Purge only |
| C | Both receipt and purge |

E	Either receipt or purge
F	Receipt and either delivery or purge
(binary zero)	Network acknowledgment code (NETACK) from the trading partner profile is returned in this field

For canceling, valid values are:

(blank)	No acknowledgments
H	Only header information in the acknowledgment
T	Both header and text information in the acknowledgment

The Expedite parameter is ACK.

RESRECL

Indicates whether the received data set records and the output records must be the same length. Valid values are:

E	Applies only to IINB41. Ends the session with an error if the length of the output record is greater than the length of the data set record.
S	Splits the output records to fit the length defined for the data set records.

You do not need to set this field if the storage format (**STGFRMT** from the trading partner profile) contains **C** or **D**. In these cases, the length of the output record is set to the length of the data set record.

The Expedite parameter is RESRECL.

SCRIPT

The name of the script that your communications software should follow when processing requests for service. The script would be part of your communication software package and not part of DataInterchange.

ENAME

The envelope name. For sending and canceling, if your application supplies a null value (binary zero), Communications uses the value in the **Message user class** field from the mailbox (requestor) profile and returns it here. If the mailbox (requestor) profile does not supply a value, the default is **#IDI#**.

For receiving, if your application supplies a null value, Communications uses the value in the **Message user class** field from the mailbox (requestor) profile and returns the message user class of the first file received.

The Expedite parameter is ACK.

MSGNAME

The message name. You may define any value that fits in the field length, or use blanks. For sending or canceling, your application supplies the message name. The default is **#IDI#**.

For receiving, communications returns the message name of the first file received.

The Expedite parameter is MSGNAME.

FILENAME

The name of a file containing data to be sent or into which data is to be received from the network. The **Data type** field (**DATATYP**) indicates whether the name is a data set name or a ddname.

Your application must supply the file name for sending non-EDI files (function codes **121** and **221**). You can use a ddname only for immediate sends. For sending EDI standard transactions (function code **211**), this field is optional. If you do not specify a file name, Communications uses the ddname in the **Transaction data queue** field from the network profile member. When sending EDIFACT or UN/TDI requests, an **E** is appended to the ddname. When sending UCS requests, a **U** is appended to the ddname. The ddname is used as supplied for sending X12 requests. Communications also sets the **Data type** field (**DATATYP**) to **D**.

For receiving non-EDI files, your application must supply the file name. You can use a ddname only for immediate receives. For receiving EDI standard transactions, this field is optional. If you do not specify a file name, Communications uses the ddname in the **Receive file name** field from the mailbox (requestor) profile member. DataInterchange also sets the **Data type** field (**DATATYP**) to **D**.

The Expedite parameter for network profiles is FILEID.

CANS D

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

The Expedite parameter for network profiles is STARTDATE.

CANST

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

The Expedite parameter for network profiles is STARTTIME.

CANED

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

The Expedite parameter is ENDDATE.

CANET

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

The Expedite parameter is ENDTIME.

TMZONE

The time zone for cancellation requests. Valid values are:

- L** Local time
- G** Greenwich mean time

The Expedite parameter is TIMEZONE.

MODEM

The type of modem used.

NPSSCDE

The network program start-session response code.

NPESCDE

The network program end-session response code.

NPERRCD

The network program error code.

NPSEVER

The network program error severity code.

BLKTYPE

The type of data blocks passed to communications for sending messages. Applies only to sending messages using function codes **141** and **241**. Valid values are:

- | | |
|----------------|--|
| H | Data blocks are larger than 32-K bytes. |
| (other) | Data blocks are smaller than 32-K bytes. |

FQUEUED

Indicates whether the network supports queued functions. Valid values are:

- | | |
|----------------|-----------------------------------|
| Y | Supports queued functions |
| (other) | Does not support queued functions |

FMSGGS

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)** Does not support CANCEL functions

FCLASS

Indicates whether the network supports user message classes. Valid values are:

- Y** Supports queued functions
- (other)** Does not support queued functions

FAACK

Indicates whether the network supports network acknowledgments. Valid values are:

- Y** Supports network acknowledgments
- (other)** Does not support network acknowledgments

FSYSMSG

Indicates whether the network supports system messages. Valid values are:

- Y** Supports system messages
- (other)** Does not support system messages

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 DataInterchange use only).

FSENTNET

Indicates whether the status should be set directly to **Sent to network** (skipping over **Send requested** status). This is the case with network program EDIMQSR (the MQSeries network program). Valid values are:

- | | |
|----------|--|
| Y | Skips the Send requested status and sets status to Sent to network |
| N | Does not skip the Send requested status |

RESERV4

Reserved.

Trading Partner Profile Block (TPPDB)

The following table 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 NETOP profile are relative to **1**.

Name	Offset	Length	Type	Description
BLKLEN	0	2	Bin	Length of TPPDB
RESERV1	2	2	Bin	Reserved
BLKNME	4	8	Char	Block name
TPNICKNM	12	16	Char	Trading partner nickname
NETID	28	8	Char	Network ID
SYSQUAL	36	1	Char	Intersystem address qualifier
SYSID	37	8	Char	Intersystem ID
ACCTNUM	45	32	Char	Requestor's network account number
USERID	77	32	Char	Requestor's network user ID
ENVLQUAL	109	4	Char	Interchange qualifier
ENVLID	113	35	Char	Interchange sender/receiver ID
CONAME	148	40	Char	Company name
ADDR1	188	40	Char	Company address line 1
ADDR2	228	40	Char	Company address line 2
PHONE	268	25	Char	Contact phone number
CONTACT	293	30	Char	Contact name
PASSWORD	323	14	Char	Interchange password for send
RCVPASS	337	14	Char	Interchange password for receive
SECUID	351	8	Char	Network security profile member
NETCLS	359	1	Char	Network message class
NETCHG	360	1	Char	Network charges code
NETACK	361	1	Char	Network acknowledgment code
NETVCHK	362	1	Char	Destination verification code
NETRETN	363	3	Char	Mailbox retention period
NETEDIO	366	1	Char	Option for storing received data
NETEDIP	367	1	Char	Special processing requested for received data
STGFRMTO	368	1	Char	Storage format override
MACHTYPE	369	1	Char	Machine type
STGFRMT	370	1	Char	Storage format
EOTID	371	1	Char	End of text/message delimiter

Name	Offset	Length	Type	Description
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
TPDATA LINE	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
TPENV SUF	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 DataInterchange
TPSUPAD3	546	40	Char	Company address line 3
TPSUPCTY	586	30	Char	City name
TPSUPST	616	2	Char	State code
TPSUPPST	618	15	Char	Postal code
TPSUPCON	633	30	Char	Country code
TPSUPFAX	663	25	Char	Fax number
TPSUPU3	688	40	Char	Comment line 3
TPSUPU4	728	40	Char	Comment line 4
TPSUPU5	768	40	Char	Comment line 5
TPSUPU6	808	40	Char	Comment line 6
TPSUPU7	848	40	Char	Comment line 7
TPSUPU8	888	40	Char	Comment line 8
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 DataInterchange

Name	Offset	Length	Type	Description
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 DataInterchange

TPPDB field descriptions

BLKLEN

A 2-byte binary field that contains the length of the TPPDB data block. A TPPDB is 1532 bytes.

RESERV1

Reserved.

BLKNME

The name of this block.

TPNICKNM

The name you use to refer to the trading partner. The value in this field must identify a member from the trading partner profile.

NETID

The network ID. The value in this field must match the key field of a member of the network profile.

SYSQUAL

Indicates whether intersystem addressing is required for this trading partner. For the AT&T Global Network, must contain the value **I** if intersystem addressing is required (IN reference: DTBLTYP). Enter the ID of the other system in the **SYSID** field.

SYSID

For intersystem addressing, the ID of the system responsible for the receiver's account. The ID is limited to 3 characters.

The Expedite parameter for network profiles is SYSID.

ACCTNUM

The account number that the network assigns to the trading partner. Applies only to sending or receiving non-EDI files and messages. The entry must be left-justified. For sending and receiving EDI standard transactions using ISA/IEA envelopes, the last position must be blank. The combined value of this field and the **USERID** field must be a unique value. This field is required if you want to use network acknowledgments.

USERID

The user ID that the network assigns your trading partner. Applies only to sending or receiving non-EDI files and messages. The entry must be left-justified. The combined value of this field and the **ACCTNUM** field must be a unique value. Together, the account number and user ID make up the trading partner ID in the interchange envelope except for UCS (BG/EG) envelopes. For UCS envelopes, the phone number contains the trading partner ID. This field is required if you want to use network acknowledgments.

ENVLQUAL

The type of interchange ID used in the **ENVLID** field. The EDI standard defines these codes. If this field or the **Interchange ID** field (**ENVLID**) is blank, the enveloper takes the qualifier from the envelope profile member.

ENVLID

The ID used to fill in the interchange receiver ID field when you send to this partner, and to identify the interchange sender when you receive from this partner. If you leave this field blank, the enveloper uses the account number and user ID (or phone number for BG/EG interchanges).

CONAME

The name of the trading partner's company. The company name may be used as envelope data by using the **CO** envelope data type.

ADDR1

Line 1 of the trading partner's address.

ADDR2

Line 2 of the trading partner's address.

PHONE

The trading partner's telephone number. When enveloping type **U** (BG/EG) interchanges, if the **Interchange ID** field is blank, the value in this field is used as the interchange receiver ID. When de-enveloping type **U** interchanges, if the **Interchange ID** field is blank, the value in this field is used as the interchange sender ID.

CONTACT

The name of the person you speak with when dealing with this trading partner.

PASSWORD

The password that you and your trading partner agreed upon for sending to this trading partner. The value in this field corresponds to the **PW** data type in the interchange envelope.

RCVPASS

The password that you and your trading partner agreed upon for receiving from this trading partner. If this value matches the interchange password (**PW** data type) that was received, translation occurs.

SECUID

The name of the network security profile member that specifies the encryption and authentication processes that apply to EDI data. For sending, the trading partner usage/rule specifies the network security profile member. If the send usage/rule does not specify a member, the member specified here is used.

NETCLS

Indicates any special status of the data being sent. Applies only to send requests. Valid values are:

(blank)	Normal status
T	Test status

The Expedite parameter for network profiles is MODE.

NETCHG

Indicates how charges are shared between sender and receiver. Applies only to send requests. Valid values are:

- 1** Receiver pays all charges.
- 2** Receiver pays all charges if agreed to, or charges are split between sender and receiver.
- 3** Receiver pays all charges if agreed to, or charges are split between sender and receiver if agreed to. Otherwise, the sender pays all charges (default).
- 4** Charges are split between sender and receiver if agreed to. Otherwise, the sender pays all charges.
- 5** Charges are split between sender and receiver.
- 6** Sender pays all charges.

The Expedite parameter for network profiles is CHARGE.

NETACK

Indicates which network acknowledgments are requested. Applies to send requests when the acknowledgment indicator in the CMCB contains binary zeros. For related information, see **ACKIND** on page 625. The network specifies the acceptable values. Valid values are:

(blank)	No acknowledgments
R	Receipt only
D	Delivery only
B	Both receipt and delivery
A	Purge only

- C** Both receipt and purge
 - E** Either receipt or purge
 - F** Receipt and either delivery or purge
- The Expedite parameter for network profiles is ACK.

NETVCHK

Indicates whether the destination is verified before sending occurs. Valid values are:

- N** Does not verify the destination (default)
- Y** Requires verification
- F** Requests verification, and sends even if the destination is not verified (useful for intersystem addressing)

If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

The Expedite parameter for network profiles is VERIFY.

NETRETN

The number of days that data is to be kept in the network mailbox before it is purged, if it is not received. Enter blanks or zeroes to use the default number. For Expedite Base/MVS, the valid range is **001-180**. For Expedite/CICS, the valid range is **001-099**, left-justified.

If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

NETEDIO

Indicates whether you want EDI segments stored in the receiving file as separate records. You provide the file name in the mailbox (requestor) profile. Valid values are:

- Y** Ends records at the segment delimiter (default)
- N** Does not end records at the segment delimiter

If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

The Expedite parameter is EDIOPT.

NETEDIP

Indicates whether the EDI data you receive has special EDI processing (breaking records by the segment delimiter). Valid values are:

- Y** Performs EDI processing if the common data header indicates that the data is in EDI standard format (default)
- N** Omits EDI processing, regardless of the common data header

If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

The Expedite parameter for network profiles is AUTOEDI.

STGFRMTO

Indicates whether you want to use the storage format defined in the common data header. Valid values are:

- Y** Uses the storage format as defined in the common data header (default)
- N** Ignores the storage format defined in the common data header

If there is no common data header, the format indicated in the **Storage format** field is used. If your request does not specify a trading partner, the trading partner information is taken from the mailbox (requestor) profile.

The Expedite parameter for network profiles is DLMOVERRIDE.

MACHTYPE

This field is not currently used.

STGFRMT

Indicates to the network how data is stored for free-form messages and files. Applies only to sending or receiving non-EDI files.

When determining what codes to select, consider the type of data you want to send and how the file is received. Contact a representative of each network you are using for all available codes. For example, if you are using:

For Expedite Base/MVS (IEBASE), valid values are:

- C** Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Select this option to send files containing program source code that is defined with variable length records. Output records do not include the carriage return and line-feed characters.
- L** Precedes each record with a 2-byte hexadecimal record length. Select this option when sending data in fixed format or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
- N** Stores data as it is received. Output records are built based on the record length of the data set allocated to receive the data.

For Expedite/CICS, valid values are:

- A** Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Select this option to send files containing program source code that is defined with variable length records. Output records do not include the carriage return and line-feed characters.
- L** Precedes each record with a 2-byte hexadecimal record length. Select this option when sending data in fixed format or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
- O** Other (free-form).

The Expedite parameter for network profiles is DELIMITED.

EOTID

The character that signifies the end of the message text to the network. EOTID applies only to sending or receiving free-form messages.

The Expedite parameter for network profiles is ENDSTR.

LOGENV

Indicates whether EDI standard data will be logged. Applies only when the **Log standard data** field from the 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. DataInterchange 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 DataInterchange.

TPDATALINE

The phone number used to connect your computer to talk directly to your trading partner's computer.

TIMEOUT

The maximum allowable time that the data line for communications can be idle without being dropped. If you specify a trading partner when requesting network activity (send or receive), the value for this field is taken from the trading partner profile. Otherwise, the value for this field is taken from the mailbox (requestor) profile.

SEGMENTED

Indicates whether you want EDI segments to be stored in the output file as separate records. Valid values are:

- Y** Ends records at the segment delimiter
- N** Does not end at the segment delimiter (default)

SUFFIX

A 2-character suffix for the ddname used to store the results of a fixed-to-fixed translation. The basic part of the ddname is taken from the **Application file name** field of the target data format.

TPENVSUF

A 2-character suffix for a generic EDI standard envelope profile member name. The basic part of the name is taken from the **Send or Receive usage override** field.

TPGENRCV

A code to indicate whether generic receive usages/rules are allowed for this trading partner. Valid values are:

- Y** Allows generic receive usages/rules
- (other)** Does not allow generic receive usages/rules

TPCMPRES

The Expedite Base/MVS compression code. Applies only during send processing. Valid values are:

- Y** Compresses the data.
- N** Does not compress the data.
- T** Expedite Base/MVS uses its own table to decide whether to compress the data.

TPRSRV1

Reserved for DataInterchange.

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 DataInterchange.

DESCRIPT

The description of this profile member.

LOGLOCK

The profile member logical lock flag (for DataInterchange 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 DataInterchange.

Communication Data Block (DATBLK)

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 DATBLK. Use the **BLKTYPE** field in the CMCB to indicate which definition you are using.

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

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

DATBLK field descriptions

BLKLEN

The length of the data block, including this field. The **DATBLK** 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 the table below 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 NETOP profile are relative to **1**.

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 DataInterchange
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. DataInterchange supplies the VANICICS, VANIINB1, VANIMQ, and PTTOPT programs. Any programs you write must be defined in a member of the 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:

This filename is used:	For 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, DataInterchange expects to find an allocation for the ddname SENDPOE.

DATLRECL

The length of records in the TD queue. For MVS, the logical record length that you allocate for the file.

The Expedite parameter for network profiles is VANIINB1.

TMZONE

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

The Expedite parameter is TIMEZONE.

SYSTYP

This field is not used by any currently supported network.

SYSLVL

This field is not used by any currently supported network.

TXTHDR

This field is not used by any currently supported network.

CMDOUT

The network command output file containing the network's responses to the command input file.

MSGROUT

The name of the program that processes messages from the network.

The Expedite parameter for network profiles is INBIMSG.

SEQNUM

A sequential number assigned to all outbound documents.

NETACKFILE

The name of a file (MVS 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 DataInterchange.

FILLER

Reserved.

DESCRIPT

The description of this profile member.

LOGLOCK

The profile member logical lock flag (for DataInterchange 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)

The table below 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 NETOP profile are relative to 1.

Name	Offset	Length	Type	Description
BLKLEN	0	2	Bin	Length of REQDB
RESERV1	2	2	Bin	Reserved
BLKNME	4	8	Char	Block name
REQID	12	16	Char	Requestor ID
NETID	28	8	Char	Network ID
ACCTNO	36	32	Char	Network account number
USERID	68	32	Char	Network user ID
PASSWD	100	16	Char	Network password
MSGUCL	116	8	Char	Network message user class
INDDNAME	124	8	Char	Receive file name
NETCLS	132	1	Char	Network message class
NETCHG	133	1	Char	Network charge code
NETACK	134	1	Char	Network acknowledgment
NETVCHK	135	1	Char	Destination verification
NETRETN	136	3	Char	Mailbox retention period
NETEDIO	139	1	Char	EDI receive option
NETEDIP	140	1	Char	EDI processing override
STGFRMTO	141	1	Char	Storage format override
STGFRMT	142	1	Char	Storage format
NETCMDMBR	143	8	Char	Net commands PDS member
TIMEOUT	151	4	Char	Communication line timeout value
NTACKPGM	155	8	Char	Remote Network Acknowledgments processing program
ALTNETPHONE	163	32	Char	Alternate data line phone
COMPRESS	195	1	Char	Compression
PRIORITY	196	1	Char	Delivery priority
FILLER	197	15	Char	Reserved for DataInterchange
DESCRIPT	213	30	Char	Member description
LOGLOCK	243	1	Char	Logical lock flag
LASTUID	244	17	Char	User ID that performed the latest update
LASTUDT	261	4	Bin	Date and time of the latest update

REQDB field descriptions

BLKLEN

A 2-byte binary field that contains the length of the REQDB data block. A REQDB is 264 bytes.

RESERV1

Reserved.

BLKNME

The name of the mailbox (requestor) profile block. **EDIREQDB**.

REQID

The name (key) used to refer to this requestor.

NETID

The name (key) that identifies the network. The value in this field must match the name of a member in the network profile.

ACCTNO

The account number that the network assigns to the requestor. The entry must be left-justified. For sending and receiving EDI standard transactions using ISA/IEA enveloping, the last position must be blank.

USERID

The user ID that the network assigns to the requestor. The entry must be left-justified.

PASSWD

The requestor's password for using the network. When connecting to IBM services, the first 8 characters are the current password, and the second 8 characters are the new password (used for changing the password).

MSGUCL

The message user class. A user-defined code that trading partners agree to use for identifying classes of information to be sent or received. Examples of classes are DEPT01, X12, MSG, FILE, EDI, and UCS. Leave this field blank to indicate that all information for the mailbox is to be sent or received.

INDDNAME

The name of the file into which information is received from the network. The translator processes EDI standard transactions from this file.

NETCLS

Indicates the status of the data being sent. Applies only to sending. Valid values are:

(blank)	Normal status
T	Test status

If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used.

The Expedite parameter for network profiles is MODE.

NETCHG

Indicates how charges are shared between sender and receiver. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

- | | |
|----------|--|
| 1 | Receiver pays all charges. |
| 2 | Receiver pays all charges if agreed to, or charges are split between the sender and receiver. |
| 3 | Receiver pays all charges if agreed to, or charges are split between the sender and receiver if agreed to. Otherwise, the sender pays all charges (default). |
| 4 | Charges are split between the sender and receiver if agreed to. Otherwise, the sender pays all charges. |
| 5 | Charges are split between the sender and receiver. |
| 6 | Sender pays all charges. |

The Expedite parameter for network profiles is CHARGE.

NETACK

Indicates which network acknowledgments (receipt, delivery, purge) you want to receive when sending to this trading partner. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

- | | |
|----------------|--------------------------------------|
| (blank) | No acknowledgments |
| R | Receipt only |
| D | Delivery only |
| B | Both receipt and delivery |
| A | Purge only |
| C | Both receipt and purge |
| E | Either purge or delivery |
| F | Receipt and either delivery or purge |

The Expedite parameter for network profiles is ACK.

NETVCHK

Indicates whether the destination is verified before sending occurs. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

- N** No verification (default)
- Y** Verification required
- F** Attempt to verify, but send even if the destination is not verified (useful for intersystem addressing)

The Expedite parameter for network profiles is VERIFY.

NETRETN

The number of days that data is to be kept in the network mailbox before it is purged, if it is not received. Enter blanks or zeroes to use the default number. For Expedite Base/MVS, the valid range is **001-180**. For Expedite/CICS, the valid range is **01-99**, left-justified. If your request specifies a trading partner, the value is taken from the trading partner profile, and the value in this field is not used.

NETEDIO

A value that indicates whether you want to store EDI segments in the receiving file as separate records. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

- Y** Ends records at the segment delimiter (default)
- N** Does not end records at the segment delimiter

The Expedite parameter is EDIOPT.

NETEDIP

Indicates whether EDI data you receive has special EDI processing (breaking records by the segment delimiter). If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

- Y** Performs EDI processing if the common data header indicates that the data is in EDI standard format (default)
- N** Omits EDI processing, regardless of the common data header

The Expedite parameter for network profiles is AUTOEDI.

STGFRMTO

Indicates whether you want to use the storage format defined in the common data header. If there is no common data header, the format indicated in the storage format field is used. If your request specifies a trading partner, the value is taken from the trading partner profile and the value in this field is not used. Valid values are:

- Y** Uses the storage format as defined in the common data header (default)
- N** Ignores the storage format defined in the common data header

The Expedite parameter for network profiles is DLMOVERRIDE.

STGFRMT

Indicates to the network how data is stored for free-form messages and files.

Valid values for several IBM products are listed below. Contact a representative of each network you are using for all available values.

For Expedite Base/MVS (IEBASE), valid values are:

- C** Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Program source code defined with variable length records is the type of file generally sent with this option. Output records do not include the carriage return and line-feed characters.
- L** Precedes each record with a 2-byte hexadecimal record length. Select this option when sending data in fixed format or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
- N** Stores data as it is received. Output records are built based on the record length of the data set allocated to receive the data.

For Expedite/CICS, valid values are:

- A** Stores each record with a carriage return and line-feed character and uses the end-of-file character. These characters are represented and stored as hexadecimal values 0D0A (CRLF) and 1A (EOF). Program source code defined with variable length records is the type of file generally sent with this option. Output records do not include the carriage return and line-feed characters.
- L** Precedes each record with a 2-byte hexadecimal record length. Select this option when sending a data set defined in fixed format or when sending binary data. The output record is determined by the value in the first 2 bytes that contain the record length.
- O** Other (free-form).

The Expedite parameter for network profiles is DELIMITED.

NETCMDMBR

The name of a PDS member that will be allocated to the ddname of EDINTCMD. This member will contain the commands that you want to pass to a network. DataInterchange 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 DataInterchange.

TIMEOUT

The maximum allowable time that the data line for communications can be idle without being dropped. If you specify a trading partner when requesting network activity (send or receive), the value for this field is taken from the trading partner profile. Otherwise, the value for this field is taken from the mailbox (requestor) profile.

NTACKPGM

The name of a program that will be used to process network acknowledgments from a secondary network. This program is used only if you are using a gateway as your primary network to connect to a secondary network, and have requested and received network acknowledgments (into the **Network acknowledgment file**) from the secondary network.



NOTE: When a gateway is used to connect to another VAN, the gateway is referred to as the primary network because it is the network with which DataInterchange interfaces. The other VAN is referred to as the secondary or remote network because DataInterchange goes through the gateway to reach the other network. When the gateway is used to connect directly to a trading partner's site or when the gateway is used as the only network, there is no secondary network.

ALTNETPHONE

The alternate phone number to dial to connect to your network.

COMPRESS

Indicates to Expedite Base/MVS whether to compress the data. Applies only to sending. Valid values are:

- Y** Compresses the data.
- N** Does not compress the data.
- T** Uses the Expedite Base/MVS compression table to decide whether to compress the data.

PRIORITY

Indicates to Expedite Base/MVS and Expedite/CICS how to prioritize message delivery. Valid values are:

- (blank)** Normal delivery
- P** High priority

FILLER

Reserved for DataInterchange.

DESCRIPT

The description of this profile member.

LOGLOCK

The profile member logical lock flag (for DataInterchange 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.

DataInterchange condition codes and API return codes

The DataInterchange Utility places return codes generated by utility-related errors in the Utility Control Information Block (FFUS). For more information, see “DataInterchange Utility control information” on page 507.

In MVS, condition codes are returned in the **CCBERC** field in the FFUS. For MVS batch users, condition codes are returned as the Job Step Condition Code.

In CICS, condition codes are returned in the **CCBRC** and **CCBERC** fields in the FFUS. The CCBRC field contains the utility severity code (**UTILSEV**). The CCBERC field contains the utility condition code (**UTILCCODE**).

Do not confuse the **CCBRC** and **CCBERC** fields in the FFUS with the **ZCCBRC** and **ZCCBERC** fields in the CCB.

This Appendix contains the condition codes, return codes, and extended return codes that DataInterchange can pass to an application. The tables in this chapter refer to the DataInterchange utility Severity and Condition codes. The codes are arranged in the following groups:

- DataInterchange utility severity codes and condition codes including:
 - General function messages of the DataInterchange Utility
 - Translation messages
 - Communications messages
 - Combination Command messages
- API return codes including:
 - Initialization return codes
 - Translation return codes
 - Communications return codes
 - Ending DataInterchange return codes
 - Negative return codes

For more information and recommended actions, refer to *DataInterchange Messages and Codes*.

DataInterchange condition codes

The DataInterchange Utility returns the most serious return code encountered during a job step. In MVS, your JCL can include logic that tests the return code and, if appropriate, ends the job. If a nonzero job step condition code is returned, examine the audit trail report (ddname PRTFILE) for details.

In those cases where you want to ignore certain condition codes, you can use the keywords **IFCC** and **SETCC** on any PERFORM statement to override up to 10 different condition codes. For more information on overriding condition codes, see General DataInterchange Utility Condition Codes below.



CAUTION: Exercise caution to avoid overriding a meaningful error and thus causing unpredictable results.

The following table describes condition codes for the DataInterchange Utility functions of communication and translation.

Severity code	Condition code	Message ID	Description
0000	0000	None	Processing completed successfully for all PERFORM commands.
0008	0003	None	Translation error. See "Translation condition codes" on page 661. This error may indicate that records were written to the exception file. See CCEXCEPTION value Y ; message ID FF0194.
0008	0016	None	Edit Services could not successfully initialize.
0008	0020	FF0486	Command only supported in CICS.
0008	0032	None	Cannot open the print file (ddname PRTFILE in MVS).
0008	0036	FF0550	Report continuous receive status error.
0008	0048	TS0100 - TS0230	An error occurred while parsing the command language input.
0008	0064	FF0410	A command file name was not passed into the DataInterchange Utility (CICS only).
0008	0080	FF0411	Cannot open the command file (ddname SYSIN in MVS).
0008	0088	FF0477 - FF0478	An error has occurred attempting to invoke a response application.
0008	0096	FF0200	The DataInterchange Utility attempted to obtain storage, but could not.
0008	0112	FF0412	An error occurred while attempting to read from the command file.
0008	0120	None	Service Director could not successfully initialize.

Severity code	Condition code	Message ID	Description
0008	0128	FF0413	Too many commands were contained in the command file.
0008	0144	FF0414	An error occurred while attempting to close the command file.
0008	0160	FF0404	An error occurred while attempting to release a storage location.
0008	0176	Multiple	An error occurred in the Transaction Store while selecting transactions. (Check for repository messages.)
0008	0184	FF0415	There are no transactions in the Transaction Store to process.
0008	0192	FF0416	A selection criteria was specified, but no transactions in the store meet the criteria.
0008	0208	FF0417	A query file name was not passed into the DataInterchange Utility (CICS only).
0008	0224	FF0418	Cannot open the query file (ddname EDIQUERY in MVS).
0008	0240	FF0419	An error occurred while attempting to write to the query file.
0008	0256	FF0420	An error occurred while attempting to close the query file.
0008	0272	FF0422 - FF0425	An error occurred during the processing of a PURGE command.
0008	0288	FF0426 - FF0429	An error occurred during the processing of an UNPURGE command.
0008	0300	CR0040 + VM1017 or VN1019	An error occurred in Expedite/CICS during start and stop of continuous receive.
0008	0303	CR0010 + PS0301	An error occurred while attempting to retrieve a continuous receive profile during a start or stop of continuous receive.
0008	0304	FF0430 - FF0433	An error occurred during the processing of a HOLD command.
0008	0320	FF0434 - FF0437	An error occurred during the processing of a RELEASE command.
0008	0336	FF0510	An error occurred during the processing of a PRINT command.
0008	0464	FF0438 - FF0443 - FF0571	An error occurred during the processing of an ENVELOPE command.

Severity code	Condition code	Message ID	Description
0008	0480	FF0444 - FF0449 - FF0571	An error occurred during the processing of a REENVELOPE command.
0008	0496	FF0421	A request to send or receive data was issued, but a requestor ID was not supplied.
0008	0512	FF0453	An error occurred while attempting to retrieve a requestor ID.
0008	0544	FF0450	A request to send or receive data was issued, but a requestor ID was not supplied that matches a network for which data was enveloped.
0008	0560	MS0010 - MS0020	An error occurred while attempting to retrieve a message from the message file.
0008	0576	FF0401	An error occurred while attempting to write to the print file.
0008	0592	FF0451	A command requiring selection criteria was issued, but no selection criteria were specified.
0008	0608	FF0452	A requestor ID was specified, but it could not be found.
0008	0624	FF0454 - FF0462	An error occurred during the processing of a REMOVE TRANSACTIONS command.
0008	0632	FF0573	Remove log error.
0008	0634	FF0575	Load log error.
0008	0636	FF0577	Unload log error.
0008	0640	FF0300	Cannot open the tracking file (ddname FFSTRAK in MVS).
0008	0656	FF0302	An error occurred while attempting to write to the tracking file.
0008	0672	FF0300	Cannot open the exception file (ddname FFSEXP in MVS).
0008	0688	FF0302	An error occurred while attempting to write to the exception file.
0008	0704	EI0002 - EI0063	An error occurred during the processing of an EXPORT command.
0008	0720	EI0002 - EI0063	An error occurred during the processing of an IMPORT command.
0008	0736	FF0467	The command file does not contain any commands to process.

Severity code	Condition code	Message ID	Description
0008	0752	Multiple	An error occurred during the processing of a CLOSE MAILBOX command. (Check for communication errors.)
0008	0768	FF0469	Elapsed time exceeded for REMOVE TRANSACTIONS.
0008	0784	FF047	No records match selection criteria for DATA EXTRACT command.
0008	0816	FF0421	Mandatory REQID keyword not provided.
0008	0832	FF0474	Error reading NETPROF profile member.
0008	0848	FF0475	Error encountered by message handler program.
0008	0864	FF0525	Error during UPDATE STATUS command.
0008	0880	FF0527, FF0529, FF0531, or FF0534	Error during management reporting command.
0008	0896	FF0546	The delete profile request failed.
0008	0903	FF0194	Records written to the exception file. (See CCEXCEPTION value X.)
0008	0912	FF0480	No message handler specified.
0008	0928	FF0481	No network output file specified.
0008	0976	FF0542	Reconstruct command failed.
0008	0992	FF0548	The query profile request failed.

TRANSFORM request return codes

The results of the TRANSFORM request are posted in the return code and extended return code fields of the CCB. Only the highest code value is posted by the message broker. The following table describes TRANSFORM return codes.

Severity code	Condition code	Message ID	Description
0000	0000	FF0585	Successful data transformation.
0008	0004	FF0584	The highest error condition encountered during data transformation was 4. Examine the Utility audit file messages to determine whether the transformation was acceptable.

Severity code	Condition code	Message ID	Description
0008	0008	FF0584	The highest error condition encountered during data transformation was 8. Examine the event log or audit file messages. This error usually means the transformation was not successful.
0008	0012	FF0584	The highest error condition encountered during data transformation was 12. Examine the event log or audit file messages. This error means the transformation was not successful.
0012	-0099	FF0583	An abend or other serious exception occurred within the Message Broker. Rerun the transformation, specifying TRACELEVEL as A3 . Then contact Customer Support providing them with the trace output, the input data, and an export file containing the map and all associated objects.

Translation condition codes

This section categorizes all translation errors by level, describes send and receive translation considerations, and tabulates all translator errors with their corresponding DataInterchange Utility condition codes. The applicable DataInterchange PERFORM commands are:

- For send translation
 - TRANSLATE TO STANDARD
 - TRANSLATE AND ENVELOPE
 - TRANSLATE AND SEND
- For receive translation:
 - DEENVELOPE
 - TRANSLATE TO APPLICATION
 - RETRANSLATE TO APPLICATION
 - DEENVELOPE AND TRANSLATE
 - RECEIVE AND TRANSLATE

The highest error encountered is reflected in the DataInterchange Utility condition code as follows:

Code	Description
0	Warnings or normal conditions
1	Data element errors
2	Segment errors
3	Transaction errors
4	Group envelope errors
5	Interchange envelope errors
6	Invalid data errors
>120	Environmental and program errors

The translator continues processing in all cases except severe errors (environmental and program errors). For example, if the translator encounters an interchange level error code of **5**, it continues by processing the next interchange. Similarly, if an unacceptable translation error occurs, the translator skips the current transaction and goes on to the next transaction. The acceptable error level is defined in the usage/rule. You can indicate if a level of **0**, **1**, or **2** is acceptable. When a program error occurs, the translator aborts immediately. The DataInterchange Utility records the errors in the audit trail file (print file) and event log.



NOTE: The DataInterchange Utility does not return the information records you requested for translation errors with a level of **3** or greater.

Outbound translation considerations

When errors occur that exceed the acceptable error level you defined for the transaction, the translation of the current transaction is stopped. Transactions that are not successfully translated are written to the exception file. If the DataInterchange Utility encounters one of the errors described in this section and the error is defined as unacceptable, then DataInterchange writes the entire

transaction to the exception file. Also, if C and D records are being used, the return codes in the control record are updated accordingly. Data records in the exception file are copies of your input records.

**NOTES:**

1. Depending on the error, the exception file may not contain all the untranslated records. You may have to use your original application files to correct and reprocess the untranslated transactions. The audit trail report contains information that can help you recover the untranslated transactions. The event log is also an important source of diagnostic information.
2. Initialization errors, such as failure to open a file, result in SYSPRINT error messages.

Additionally, the DataInterchange Utility can generate errors prior to translator invocation. The following table shows the condition codes that may be caused by problems with an application file (keyword APPFILE).

Severity code	Condition code	Message ID	Description
0008	0006	FF0131	The input APPFILE is empty.
0008	0009	FF0152	A transaction in APPFILE does not contain a valid record code.
0008	0011	FF0151	A transaction in APPFILE does not contain a C record.
0008	0017	FF0150	A transaction in APPFILE does not contain a D record.

Inbound translation considerations

When errors occur that exceed the acceptable error level you chose when defining the transaction, the translation of the current transaction is stopped. Transactions that are not successfully translated are not written to the exception file. If the DataInterchange Utility encounters one of the errors described in this section and the error is defined as unacceptable, then DataInterchange writes the transaction to the Transaction Store, but produces no application output data.

**NOTES:**

1. Application output is only written to the exception file in the event that the application file(s) cannot be opened.
2. You can use the RETRANSLATE TO APPLICATION command to reprocess the untranslated transactions. The audit trail report contains information that can help you recover the untranslated transactions. The event log is also an important source of diagnostic information.

Translation condition code tables

The next nine tables list the possible translator errors. Each table represents a category as follows.

Error category	Example
Warning conditions	Structure not used, end of file
Data element errors	Data element too short or too long
Segment errors	Mandatory segment missing, unrecognized segment
Transaction syntax errors	Control number mismatch, segment count incorrect
Transaction environmental errors	Translation usage/rule not found
Group errors	Control number mismatch, password invalid
Interchange errors	Control number mismatch, trading partner not known
Invalid data errors	Not an interchange
Environmental or program errors	Database error, insufficient virtual storage

For more information about a specific error message, refer to *DataInterchange Messages and Codes*.

Warnings and normal condition codes

The following table describes normal conditions that do not set a condition code and warnings.

Severity code	Condition code	Message ID	Unique code	Description
0000	0000	None	0	Processing of requested function completed normally.
0000	0000	TR0841	2	Structure not defined in the data format.
0000	0000	TR0403	3	Structure not used in the translation.
0000	0000	TR0404	4	Structure defined as part of parent (not passed separately).
0000	0000	TR0405	5	Raw data structure could not be identified.
0000	0000	TR0406	6	Raw data structure received, but structure defined to start the translation not yet received.
0000	0000	TR0407	7	Mismatching loop ID values in LS and LE segments.
0000	0000	TR0408	8	LS segment does not have a corresponding LE segment.

Severity code	Condition code	Message ID	Unique code	Description
0000	0000	TR0409	9	No translation usage/rule provided for hierarchical code and parent hierarchical code combination.
0000	0000	TR0824	505	Transmission of envelope to communications failed.
0000	0000	TR0401	1	End of file, no more envelopes in the envelope queue.

Data element condition codes

The following table describes the errors that occur at the data element level. The translation might still be acceptable based on the acceptable error level established in the send or receive usage record.

Severity code	Condition code	Message ID	Unique code	Description
0008	0001	None	0	The highest severity of error detected during translation of this transaction is at the data element level.
0008	0001	TR0001	101	A mandatory data element is blank for a segment containing data in other data elements.
0008	0001	TR0002	102	Data element is too long.
0008	0001	TR0003	103	Data element is too short.
0008	0001	TR0004	104	Code in ID type field not found in validation table.
0008	0001	TR0005	105	Code in ID type field not found in translation table.
0008	0001	TR0006	106	User exit for data element failed. Exit routine returned an error.
0008	0001	TR0007	107	Invalid date format. Unable to reformat date according to customized date edit number.
0008	0001	TR0008	108	Data element conversion failed. Format of data in input buffer conflicts with translation usage/rule.
0008	0001	TR0009	109	Standard length exceeds application length.
0008	0001	TR0010	111	Paired conditionality for segment not satisfied.
0008	0001	TR0011	112	Required conditionality for segment not satisfied.

Severity code	Condition code	Message ID	Unique code	Description
0008	0001	TR0012	113	Mutually exclusive conditionality for segment not satisfied.
0008	0001	TR0013	114	Conditionality for segment not satisfied.
0008	0001	TR0014	110	Paired conditionality for segment not satisfied.
0008	0001	TR0015	115	Mandatory composite field missing.
0008	0001	TR0016	116	Data element validation failed.
0008	0001	TR0017	117	Attempt to increment accumulator will exceed maximum size.
0008	0001	TR0018	118	Attempt to add a value to an accumulator will exceed maximum size.
0008	0001	TR0023	119	Data in data format has been overlaid.
0008	0001	TR0024	120	Field data has been received that was not defined for this transaction.

Segment condition codes

The following table describes the errors that occur on the segment level. The translation might still be acceptable based on the acceptable error level established in the send or receive usage record.

Severity code	Condition code	Message ID	Unique code	Description
0008	0002	None	0	The highest severity of error detected during translation of this transaction is at the segment level.
0008	0002	TR0019	207	Error getting storage while processing binary segment.
0008	0002	TR0020	208	Error opening a file while processing binary segment.
0008	0002	TR0021	209	Error reading a file while processing binary segment.
0008	0002	TR0022	210	Error writing a file while processing binary segment.
0008	0002	TR0050	201	A segment contains more elements than the EDI standard allows.
0008	0002	TR0051	202	Unrecognized segment ID. The segment is not defined for the transaction.
0008	0002	TR0052	203	A mandatory segment for this trading partner is missing.

Severity code	Condition code	Message ID	Unique code	Description
0008	0002	TR0053	204	Occurrences of a repeating segment in the input data exceed the maximum use count for the structure in the data format.
0008	0002	TR0054	205	Loop repetition count exceeded. Loop repeats more times than definition specifies.
0008	0002	TR0055	206	Segment repetition count exceeded. Segment repeats more times than definition specifies.
0008	0002	TR0056	211	Unexpected segment data received.
0008	0002	TR0057	215	Application data received out of sequence.
0008	0002	TR0058	212	Creation of EDI standard loop has been aborted.
0008	0002	TR0059	213	Creation of a repeating segment has been aborted.
0008	0002	TR0060	214	Creation of a segment has been aborted.

Transaction condition codes

The following table describes errors that indicate a serious problem with the syntax of a transaction. Such errors can occur when DataInterchange is unable to locate a translation usage/rule or control string needed to process the transaction.

Severity code	Condition code	Message ID	Unique code	Description
0008	0003	None	0	The highest severity of error detected during translation of this transaction is at the transaction set level.
0008	0003	TR0025	326	Duplicate transaction within group or interchange.
0008	0003	TR0101	302	Transaction set control numbers do not match in header and trailer.
0008	0003	TR0103	304	Transaction set trailer contains invalid segment count.
0008	0003	TR0105	315	Network security profile member could not be found for encrypted transaction.
0008	0003	TR0106	316	Authentication failed.
0008	0003	TR0107	317	S2S segment without S2E. Incomplete security segments for received transaction.

Severity code	Condition code	Message ID	Unique code	Description
0008	0003	TR0108	318	S2E segment without S2S. Incomplete security segments for received transaction.
0008	0003	TR0207	506	Invalid envelope definition. Envelope definition damaged.
0008	0003	TR0208	507	Invalid envelope segments. Envelope definition damaged.
0008	0003	TR1257	335	Mandatory composite missing in service segment.
0008	0003	TR1258	336	Service segment data element too long.
0008	0003	TR1259	337	Service segment data element value not defined in validation table.
0008	0003	TR1260	338	Service segment data element value not consistent with data type.
0008	0003	TR1261	339	Mandatory data element missing in service segment.
0008	0003	TR1262	340	Service segment data element too small.

Transaction environmental errors

The following table describes the condition codes for environmental errors that occur during transaction translation. Such errors can occur when DataInterchange is unable to locate a translation usage/rule or control string needed to process the transaction.

Severity code	Condition code	Message ID	Unique code	Description
0008	0003	TR0104	305	Receive map not found.
0008	0003	TR0109	319	Attempt to receive translate (TRANSLATE TO APPLICATION) a transaction that was previously sent translated (TRANSLATE TO STANDARD).
0008	0003	TR0110	320	Invalid transaction handle.
0008	0003	TR0111	321	Attempt to envelope a received transaction.
0008	0003	TR0112	322	Attempt to translate a transaction that does not have a group or interchange segment associated with it.
0008	0003	TR0113	324	Attempt to envelope a transaction from a bundle without enveloping the controlling transaction.

Severity code	Condition code	Message ID	Unique code	Description
0008	0003	TR0114	325	Attempt to envelope a transaction that was not translated successfully.
0008	0003	TR0115	327	Transaction control number assigned by application not valid.
0008	0003	TR0116	328	Transaction control number assigned by application is a duplicate.
0008	0003	TR0117	330	Call to transaction services for details failed.
0008	0003	TR0118	331	Error attempting to get image from Transaction Store.
0008	0003	TR0119	332	Request to get an image that could not be found.
0008	0003	TR0120	333	Request to get an image but the base has not been established.
0008	0003	TR0121	334	Attempt to envelope a transaction with an incorrect status.
0008	0003	TR0155	406	Authentication routine required but not provided.
0008	0003	TR0156	407	Encryption routine required but not provided.
0008	0003	TR0157	408	Filtering routine required but not provided.
0008	0003	TR0818	307	The translator could not write to the event log. The log might be full.
0008	0003	TR0820	308	Send map not found for this trading partner, application, and direction.
0008	0003	TR0821	306	Data format not found.
0008	0003	TR0822	309	Control string not found for this transaction.
0008	0003	TR0825	310	Trading partner profile member not found.
0008	0003	TR0826	311	Sender ID in envelope profile member is blank.
0008	0003	TR0830	312	Input data block contains no data. Application data length is 0.
0008	0003	TR0834	313	Overran output buffer. User's output buffer is too small for the data.

Severity code	Condition code	Message ID	Unique code	Description
0008	0003	TR0848	329	Failed to load a user exit routine.
0008	0003	TR0850	323	EDI standard delimiters are not unique. Update delimiters in EDI standard envelope or trading partner profile.
0008	0003	SA0042	314	Access denied to function within resource.

Group condition codes

The following table describes the errors that indicate a serious problem with an entire functional group within an interchange. None of the transactions in the group are processed.

Severity code	Condition code	Message ID	Unique code	Description
0008	0004	None	0	An error is detected in the functional group header or trailer. No transactions in the functional group are translated. Processing continues.
0008	0004	TR0100	301	Transaction set header is missing or invalid.
0008	0004	TR0102	303	Transaction set trailer is missing or invalid.
0008	0004	TR0107	409	S1S segment without S1E.
0008	0004	TR0108	410	S1E segment without S1S.
0008	0004	TR0151	402	Functional group control numbers do not match in header and trailer.
0008	0004	TR0153	404	Functional group trailer contains invalid transaction count.
0008	0004	TR0154	405	Authentication failed for group.
0008	0004	TR0155	406	Authentication routine required but not provided.
0008	0004	TR0156	407	Encryption routine required but not provided.
0008	0004	TR0157	408	Filtering routine required but not provided.
0008	0004	TR0158	411	There is a duplicate group within the interchange.
0008	0004	TR0207	506	Envelope is not defined correctly. Envelope definition damaged.
0008	0004	TR0208	507	Envelope is not defined correctly. Envelope definition damaged.

Severity code	Condition code	Message ID	Unique code	Description
0008	0004	TR1257	412	Mandatory composite missing in service segment.
0008	0004	TR1258	413	Service segment data element too long.
0008	0004	TR1259	414	Service segment data element value not defined in validation table.
0008	0004	TR1260	415	Service segment data element value not consistent with data type.
0008	0004	TR1261	416	Mandatory data element missing in service segment.
0008	0004	TR1262	417	Service segment data element too small.

Interchange condition codes

The following table describes the errors indicating the translator was able to isolate an interchange to process, but there is a serious problem with that interchange. None of the transactions in the interchange are processed.

Severity code	Condition code	Message ID	Unique code	Description
0008	0005	None	0	An error is detected in the interchange header or trailer. No transactions in the interchange are translated. Processing continues.
0008	0005	TR0150	401	Functional group header is missing or invalid.
0008	0005	TR0152	403	Functional group trailer is missing or invalid.
0008	0005	TR0201	501	Interchange header contains invalid sender ID. Cannot locate trading partner profile member.
0008	0005	TR0203	502	Interchange control numbers do not match in header and trailer.
0008	0005	TR0205	503	Interchange trailer contains invalid functional group count.
0008	0005	TR0206	504	Password does not match password in trading partner profile.
0008	0005	TR0207	506	Interchange is not defined correctly, or the interchange is badly damaged and cannot be parsed.
0008	0005	TR0208	507	Received interchange segment contains invalid data.

Severity code	Condition code	Message ID	Unique code	Description
0008	0005	TR0209	508	Expected delimiter not found. Encrypted data not immediately followed by segment delimiter.
0008	0005	TR0210	509	Envelope definition not found for received interchange.
0008	0005	TR0211	510	A duplicate interchange was detected and is being skipped.
0008	0005	TR1257	511	Mandatory composite missing in service segment.
0008	0005	TR1258	512	Service segment data element too long.
0008	0005	TR1259	513	Service segment data element value not defined in validation table.
0008	0005	TR1260	514	Service segment data element value not consistent with data type.
0008	0005	TR1261	515	Mandatory data element missing in service segment.
0008	0005	TR1262	516	Service segment data element too small.

Invalid data condition codes

The following table describes the errors that indicate that the translator cannot identify the data it is reading from the file as EDI standard data. The translator expects the ISA, UNB, SCH, ICS, BG, or GS segment to signal the start of EDI standard data. Any data before, between, or after any of these valid segments is flagged as an error.

Severity code	Condition code	Message ID	Unique code	Description
0008	0006	None	0	Invalid data found in the input file. Data does not match the format required by the EDI standard.
0008	0006	TR0200	601	Interchange envelope header is missing or invalid.
0008	0006	TR0204	602	Interchange envelope trailer is missing or invalid.
0008	0006	TR0842	60	No EDI standard data found in input file.
0008	0006	TR0202	604	Interchange header found while looking for a trailer.

Environmental and program error condition codes

The following table describes the errors considered so serious that the translator logs the appropriate message for the error and stops processing immediately.

Severity code	Condition code	Message ID	Unique code	Description
0012	0121	TR1201	916	A program error occurred during anchor processing.
0012	0122	TR1202	917	Unable to release main storage.
0012	0123	TR1203	918	Unable to read repository.
0012	0125	TR1205	919	QSAM failed to read file.
0012	0126	TR1206	920	QSAM failed to close file.
0012	0127	TR1207	927	DataInterchange ended because of severe error generating functional acknowledgments.
0012	0128	TR1208	928	DataInterchange ended because of severe error updating the management reporting databases.
0012	0129	TR1209	929	Error reading envelope control string.
0012	0130	TR0810	901	Profile read for update failed.
0012	0131	TR0811	902	Profile write failed.
0012	0132	TR0812	903	Parameter count is invalid.
0012	0135	TR0815	904	A request to get main storage failed.
0012	0136	TR0816	905	Invalid function code.
0012	0137	TR0817	906	The transaction processor did not end because it is not active. This error also occurs if input records are out of sequence, such as a C record followed by a blank record.
0012	0147	TR0827	908	You cannot perform this function now. Function code not valid at this time.
0012	0148	TR0828	909	Input data block size invalid. Must be at least 32 K.
0012	0149	TR0829	910	Output data block size invalid. Must be at least 32 K.
0012	0152	TR0832	911	Transaction Store call failed.
0012	0156	TR0836	912	Repository read failed.
0012	0158	TR0838	913	QSAM failed to open file.
0012	0159	TR0839	914	Mailbox (requestor) profile not found.
0012	0160	TR0840	915	Standards profile member not found.

Severity code	Condition code	Message ID	Unique code	Description
0012	0163	TR0843	907	Receive file name in mailbox (requestor) profile is blank.
0012	0164	SA0042	921	Access denied to function within resource.
0012	0166	TR0846	922	Raw data control string not found.
0012	0167	TR0847	923	Failed to find network security profile member.
0012	0168	TR0848	924	Failed to load a user exit routine.
0012	0169	TR0849	925	Error returned by user exit routine.
0012	0171	TR0851	926	An data format ID is required when raw data is specified.
0012	0172	TR1252	930	Error reading target ADF control string.
0012	0173	TR1253	931	No beginning/ending structure.
0012	0174	TR1254	932	No internal trading partner ID value.
0012	0175	TR1255	933	User exit (IUSEREXIT) not defined.
0012	0176	TR1256	934	IUSEREXIT returned an error.
0012	0177	TR1257	935	Mandatory composite missing in service segment.
0012	0178	TR1258	936	Service segment data element too long.
0012	0179	TR1259	937	Service segment data element value not defined in validation table.
0012	0180	TR1260	938	Service segment data element value not consistent with data type.
0012	0181	TR1261	939	Mandatory data element missing in service segment.
0012	0182	TR1262	940	Service segment data element too small.
0012	0183	TR1263	941	Database error obtaining lock.

Communication condition codes

DataInterchange Utility condition codes are specialized for communication functions. Condition codes are not shared between MVS and CICS. Each environment has its own set of codes. These codes apply to the following PERFORM commands:

- SEND
- RECEIVE
- Combination commands that include SEND or RECEIVE

MVS condition codes

The following table describes job step condition codes for sending and receiving in MVS.

Condition code	Message ID	Description
0000	FF0020 or FF0030	Successful transmission.
0001	VN1015	This message ID is also used under more serious circumstances. A code of 0001 indicates that all valid data was processed successfully. A code of 0015 indicates that one or more envelopes were not processed, and processing ended prematurely.
0002	VN1040 and FF0142	No data received on a receive request.
0002	FF0140	QSAM error prevented completion of request to clear the file (send requests).
0003	VN1041	Not able to process the network command output file.
0003	FF0111	Network profile not found.
0004	VN1004, VN1005, PS0010 - PS0310	Error in Profile Services.
0005	FF0140	Network profile does not provide name of send/receive program.
0005	CM0005	Cannot pass control to VANIINB1.
0006	SA0042	No authority to perform network functions.
0007	CM0006	Network ID is invalid for the MVS environment.

Condition code	Message ID	Description
0015	VN1015	Invalid parameter passed to network program. One or more envelopes were not sent/received. Processing ended prematurely. Examine the network command output file for errors.
<i>nnnn</i>	Multiple	Other severity 0008 condition codes are possible, but unlikely. The code reflects the last three characters of the communications extended return code. <i>Example:</i> Communications returns 8 and 1012 (for message ID VN1012); the DataInterchange Utility returns 0012 . For more information, see “Communication nonsevere errors” on page 692.
<i>nnnn</i>	Multiple	When the communications return code is 0012 , all condition codes are 120 plus the last three characters of the communications extended return code. <i>Example:</i> Communications returns 12 and 1016 (for message ID VN1016); the DataInterchange Utility returns 0136 . For more information, see “Communication severe errors” on page 693.

CICS condition codes

The following table describes condition codes for sending and receiving in CICS.

Severity code	Condition code	Message ID	Description
0000	0000	FF0020 or FF0030	Successful transmission.
0004	0002	FF0142 and VN1040	No data received on a receive request.
0008	0003	FF0111	Network profile not found.
0008	0004	VN1004, VN1005, PS0010 - PS0310	Error in Profile Services.
0008	0007	CM0006	Network ID is invalid for the CICS environment.
0008	0017	VN1017	Error occurred during the execution of an Expedite/CICS command. Examine the print file or the event log for the Expedite/CICS error.
0008	0019	VN1019	A timeout error occurred while DataInterchange was waiting for Expedite/CICS to complete a continuous receive termination request.

Severity code	Condition code	Message ID	Description
0008	0020	VN1020	A timeout error occurred while DataInterchange was waiting for Expedite/CICS to complete a request to receive network acknowledgments.
0008	0021	VN1021	A timeout error occurred while DataInterchange was waiting for Expedite/CICS to complete a single receive request.
0008	<i>nnnn</i>	Multiple	Other severity 0008 condition codes are possible, but unlikely. The code reflects the last three characters of the communications extended return code. <i>Example:</i> Communications returns 8 and 1012 (for message ID VN1012); the DataInterchange Utility returns 0012 . For more information, see “Communication nonsevere errors” on page 692.
0012	<i>nnnn</i>	Multiple	When the communications return code is 0012 , all condition codes are 120 plus the last three characters of the communications extended return code. <i>Example:</i> Communications returns 12 and 1016 (for message ID VN1016); the DataInterchange Utility returns 0136 . For more information, see “Communication severe errors” on page 693.
0008	0300	CR0010 and PS0301	Error occurred while attempting to retrieve a continuous receive profile during a start or stop of continuous receive. Examine the event log for PS0301 and take the action indicated.
0008	0303	CR0040 and VM1017 or VN1019	Error occurred in Expedite/CICS during start and stop of continuous receive.

Combination command condition codes

When there is a problem processing a combination command, DataInterchange supplies one condition code for your application. This condition code might not specifically identify the problem area. If the condition code is insufficient for problem identification, restart the DataInterchange using the commands separately. This section describes the error conditions that can occur during combination command processing.

TRANSLATE AND SEND errors

When you issue a TRANSLATE AND SEND command to DataInterchange, several error conditions can occur. If a severe error occurs during translation, DataInterchange ignores the send request, and the utility condition code reflects the translator-extended return code. If an error that is not severe occurs during translation, DataInterchange attempts to complete the send process. If the send is successful, the utility condition code reflects the translator-extended return code. If an error occurred during the send, the utility condition code is a communications condition code, regardless of translation errors.

ENVELOPE AND SEND errors

When you issue an ENVELOPE AND SEND or REENVELOPE AND SEND command, errors can occur during the enveloping process. When an enveloping error occurs, the utility condition code returned is **464** (ENVELOPE) or **480** (REENVELOPE), and the send is not done. If enveloping or reenveloping is successful, the send is issued. The utility condition code is set to a communications condition code if an error occurs during the send.

RECEIVE Errors

When you issue a RECEIVE AND DEENVELOPE or RECEIVE AND TRANSLATE command, several error conditions can occur. If an error occurs during a receive, a communications condition code returns. If a receive is successful, deenveloping (and translation) is done. If an error occurs during this process, the corresponding translator extended return code is given as the utility condition code. When multiple WHERE clauses are specified, giving multiple requestor IDs, the first error (except for an empty mailbox) terminates the command, and this error is given as the utility condition code.

API return codes

The following sections describe the API return codes.

Initialization return codes

The results of the initialization request are posted in the return code and extended return code fields of the CCB. The following table describes initialization return codes.

Return code	Extended code	Description
00	00	Initialization was successful.
04	04	No service table defined (failure to locate FXXZIN load module).
04	08	Insufficient virtual storage to load programs and tables.
04	12	Failure initializing the EDIT service. This may be caused by an incorrect ZCCBLPID value (one that cannot be matched with a LANGPROF member). In DB2 installations, this error may 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), the DB2 plan must be rebound. Besides EDIPSMD, there are two other DataInterchange DB2 load modules: EDIRPML and (for CICS) EDICRIN that could generate DB2 timestamp problems if out of sync.
04	16	A DataInterchange session for the same user is already active.
04	1024	Access to the system is denied.

Negative return codes

On any request for a DataInterchange service, you can receive a negative return code in the **ZCCBRC** field. A negative return code indicates that the service requested was never invoked, and that an error occurred that prevented the service from being called. The following table describes the negative return codes.

Return code	Extended code	Description
-1		The CCB is not valid for one of the following reasons: <ul style="list-style-type: none"> The CCB is not initialized with the environmental initialization function request. The CCB has been destroyed. The program is not using the same CCB used on the initialization request.
-2		The requested service is not known to DataInterchange. The name in the ZSNBNAME field of the SNB does not identify a logical service. Logical service names must be padded with blanks.
-3		The requested service is known, but could not be found in any library, and therefore was not loaded.
-4		Inability to establish environment needed by service because of insufficient virtual storage.
-5	04	The parameter count from the SNB is less than 2. Any call to the service director must pass the SNB and CCB as parameters.
-5	08	The language the service is written in is not supported.
-8		An abend has been detected by DataInterchange/MVS-CICS. The extended return code is the EBCDIC representation of the CICS abend code.

DataInterchange termination return codes

The following table describes errors that can occur when you terminate the DataInterchange environment.

Return code	Description
0	Termination was successful.
>0	Termination failed, try again.
<0	Not a valid CCB address or an incorrect name in the ZSNBNAME field.

Translation return codes

Many translator error conditions can be detected and reported. The following table lists the categories of errors, with specific examples:

Error	Example
Warning conditions	Structure not used, end of file
Data element errors	Data element too short or too long
Segment errors	Mandatory segment missing, unrecognized segment
Transaction errors	Control number mismatch
Group errors	Control number mismatch, password invalid
Interchange errors	Control number mismatch, trading partner not known
Invalid data errors	Not an interchange
Environmental or program errors	File read error, insufficient virtual storage

More than one error can occur during translation of a transaction. The extended return code indicates the level at which an error occurred, such as the data element or segment level. The most serious errors are returned in the **ZCCBRC** and **ZCCBERC** fields.

A particular translation error (return code and extended return code combination) can have many causes. Each cause has a unique message ID and a unique error code. DataInterchange logs each message when it detects an error.

API programs can use the **ERRCDES** field of the TRCB to retrieve unique codes. For more detailed information on these errors, see "Translator Control Block (TRCB)" on page 586.

When translation errors occur during receive, the functional acknowledgment to the sender, if requested, indicates the error code identified by the message. Refer to your EDI standards documentation for codes returned in functional acknowledgments. Use the Transaction Store Facility option on the Administrator's Menu to obtain information about functional acknowledgments. For more information on the Transaction Store Facility, refer to the *DataInterchange Administrator's Guide*.

Normal or warning conditions

The following table describes the return codes that DataInterchange uses when it detects no errors.

Return code	Extended code	Msg ID	Unique code	Description
00	00	None	0	Processing of requested function completed normally.
00	00	TR0841	2	Structure not defined in the data format.
00	00	TR0403	3	Structure not used in this translation usage/rule.
00	00	TR0404	4	Structure defined as part of parent (not passed separately).
00	00	TR0405	5	Raw data structure could not be identified.

Return code	Extended code	Msg ID	Unique code	Description
00	00	TR0406	6	Raw data structure received but structure defined to start the translation not yet received.
00	00	TR0407	7	Mismatching loop ID values in LS and LE segments.
00	00	TR0408	8	LS segment does not have a corresponding LE segment.
00	00	TR0409	9	No translation usage/rule provided for hierarchical code and parent hierarchical code combination.
00	00	TR0824	50	Transmission of envelope to communications failed.
04	01	TR0401	1	End of file, no more envelopes in network queue to receive.

Data element errors

The following table describes the errors that occur at the data element level. The translation might still be acceptable depending on the acceptable error level established in the send or receive usage record.

Return code	Extended code	Message ID	Unique code	Description
08	01	None	0	The highest severity of error detected during translation of this transaction is at the data element level.
08	01	TR0001	101	A mandatory data element is blank for a segment containing data in other data elements.
08	01	TR0002	102	Data element is too long.
08	01	TR0003	103	Data element is too short.
08	01	TR0004	104	Code in ID type field not found in validation table.
08	01	TR0005	105	Code in ID type field not found in translation table.
08	01	TR0006	106	User exit for data element failed. Exit routine returned an error.
08	01	TR0007	107	Invalid date format. Unable to reformat date according to customized date edit number.

Return code	Extended code	Message ID	Unique code	Description
08	01	TR0008	108	Data element conversion failed. Format of data in input buffer conflicts with translation usage/rule.
08	01	TR0009	109	Standard length exceeds application length.
08	01	TR0010	111	Paired conditionality for segment not satisfied.
08	01	TR0011	112	Required conditionality for segment not satisfied.
08	01	TR0012	113	Mutually-exclusive conditionality for segment not satisfied.
08	01	TR0013	114	Conditionality for segment not satisfied.
08	01	TR0014	110	Paired conditionality for segment not satisfied.
08	01	TR0015	115	Mandatory composite field missing.
08	01	TR0016	116	Data element validation failed.
08	01	TR0017	117	Attempt to increment accumulator will exceed maximum size.
08	01	TR0018	118	Attempt to add a value to an accumulator will exceed maximum size.
08	01	TR0023	119	Data in data format has been overlaid.
08	01	TR0024	120	Field data has been received that was not defined for this transaction.

Segment errors

The following table describes the errors that occur on the segment level. The translation might still be acceptable depending on the acceptable error level established in the send or receive usage record.

Return code	Extended code	Msg ID	Unique code	Description
08	02	None	0	The highest severity of error detected during translation of this transaction is at the segment level.
08	02	TR0019	207	Error getting storage while processing binary segment.
08	02	TR0020	208	Error opening a file while processing binary segment.
08	02	TR0021	209	Error reading a file while processing binary segment.

Return code	Extended code	Msg ID	Unique code	Description
08	02	TR0022	210	Error writing a file while processing binary segment.
08	02	TR0050	201	A segment contains more elements than the EDI standard allows.
08	02	TR0051	202	Unrecognized segment ID. The segment is not defined for the transaction.
08	02	TR0052	203	A mandatory segment for this trading partner is missing.
08	02	TR0053	204	Occurrences of a repeating segment in the input data exceed the maximum use count for the structure in the data format.
08	02	TR0054	205	Loop repetition count exceeded. Loop repeats more times than definition specifies.
08	02	TR0055	206	Segment repetition count exceeded. Segment repeats more times than definition specifies.
08	02	TR0056	211	Unexpected segment data received.
08	02	TR0057	215	Application data received out of sequence.
08	02	TR0058	212	Creation of EDI standard loop has been aborted.
08	02	TR0059	213	Creation of a repeating segment has been aborted.
08	02	TR0060	214	Creation of a segment has been aborted.

Transaction syntax errors

The following table describes the errors that indicate a serious problem with the syntax of a transaction. Such errors can occur when DataInterchange is unable to locate a translation usage/rule or control string needed to process the transaction. An API program receives a single 8,3 return code, indicating that a transaction is being skipped.

Return code	Extended code	Msg ID	Unique code	Description
08	03	None	0	The highest severity of error detected during translation of this transaction is at the transaction set level.
08	03	TR0025	326	Duplicate transaction within group or interchange.
08	03	TR0101	302	Transaction set control numbers do not match in header and trailer.

Return code	Extended code	Msg ID	Unique code	Description
08	03	TR0103	304	Transaction set trailer contains invalid segment count.
08	03	TR0105	315	Network security profile member could not be found for encrypted transaction.
08	03	TR0106	316	Authentication produced an MAC value that does not match the received value.
08	03	TR0107	317	S2S segment without S2E. Incomplete security segments for received transaction.
08	03	TR0108	318	S2E segment without S2S. Incomplete security segments for received transaction.
08	03	TR0207	506	Envelope is not defined correctly. Envelope definition damaged.
08	03	TR0208	507	Envelope is not defined correctly. Envelope definition damaged.
08	03	TR1257	335	Mandatory composite missing in service segment.
08	03	TR1258	336	Service segment data element too long.
08	03	TR1259	337	Service segment data element value not defined in validation table.
08	03	TR1260	338	Service segment data element value not consistent with data type.
08	03	TR1261	339	Mandatory data element missing in service segment.
08	03	TR1262	340	Service segment data element too small.

Transaction environmental errors

The following table describes the errors that indicate environment-related problems with a transaction. Such errors can occur when DataInterchange is unable to locate a translation usage/rule or control string needed to process the transaction. An API program receives a single 8,3 return code, indicating that a transaction is being skipped.

Return code	Extended code	Msg ID	Unique code	Description
08	03	TR0104	305	Receive map not found.
08	03	TR0109	319	Attempted to receive and translate (TRANSLATE TO APPLICATION) a transaction that was previously sent and translated (TRANSLATE TO STANDARD).
08	03	TR0110	320	Invalid transaction handle.

Return code	Extended code	Msg ID	Unique code	Description
08	03	TR0111	321	Attempted to envelope a received transaction.
08	03	TR0112	322	Attempted to translate a transaction that does not have a group or interchange segment associated with it.
08	03	TR0113	324	Attempted to envelope a transaction from a bundle without enveloping the controlling transaction.
08	03	TR0114	325	Attempted to envelope a transaction that was not translated successfully.
08	03	TR0115	327	Transaction control number assigned by application not valid.
08	03	TR0116	328	Transaction control number assigned by application is a duplicate.
08	03	TR0117	330	Call to transaction services for details failed.
08	03	TR0118	331	Error attempting to get image from Transaction Store.
08	03	TR0119	332	Request to get an image that could not be found.
08	03	TR0120	333	Request to get an image but the base has not been established.
08	03	TR0121	334	Attempt to envelope a transaction with an incorrect status.
08	03	TR0155	406	Authentication routine required but not provided.
08	03	TR0156	407	Encryption routine required but not provided.
08	03	TR0157	408	Filtering routine required but not provided.
12	16	TR0816	???	Invalid function code. Function code is not: 1, 2, 3, 111, 131, 211, 212, 214, 215, or 1000.
08	17	TR0817	???	Transaction processor did not terminate because it was not active.
08	03	TR0818	307	The translator could not write to the event log. The log might be full.
08	03	TR0820	308	Send map not for this trading partner, application, and direction.
08	03	TR0821	306	Data format not found.
08	03	TR0822	309	Control string not found for this transaction.
08	03	TR0825	310	Trading partner profile member not found.

Return code	Extended code	Msg ID	Unique code	Description
08	03	TR0826	311	Sender ID in envelope profile member is blank.
08	03	TR0830	312	Input data block contains no data. Application data length is 0.
08	03	TR0834	313	Overran output buffer. User's output buffer is too small for the data.
08	03	TR0848	329	Failed to load a user exit routine.
08	03	TR0850	323	EDI standard delimiters are not unique. Update delimiters in EDI standard envelope or trading partner profile.
08	03	SA0042	314	Access denied to function within resource.

Group translation errors

The following table describes the errors that indicate a serious problem with an entire functional group within an interchange. None of the transactions in the group are processed. An API program receives a single 8,4 return code, indicating that a group is being skipped.

Return code	Extended code	Msg ID	Unique code	Description
08	04	None	0	An error is detected in the functional group header or trailer. No transactions in the functional group are translated. Processing continues.
08	04	TR0100	301	Transaction set header is missing or invalid.
08	04	TR0102	303	Transaction set trailer is missing or invalid.
08	04	TR0107	409	S1S segment without S1E.
08	04	TR0108	410	S1E segment without S1S.
08	04	TR0151	402	Functional group control numbers do not match in header and trailer.
08	04	TR0153	404	Functional group trailer contains invalid transaction count.
08	04	TR0154	405	Authentication failed for group.
08	04	TR0155	406	Authentication routine required but not provided.
08	04	TR0156	407	Encryption routine required but not provided.
08	04	TR0157	408	Filtering routine required but not provided.
08	04	TR0158	411	There is a duplicate group within the interchange.

Return code	Extended code	Msg ID	Unique code	Description
08	04	TR0207	506	Envelope is not defined correctly. Envelope definition damaged.
08	04	TR0208	507	Envelope is not defined correctly. Envelope definition damaged.
08	04	TR1257	412	Mandatory composite missing in service segment.
08	04	TR1258	413	Service segment data element too long.
08	04	TR1259	414	Service segment data element value not defined in validation table.
08	04	TR1260	415	Service segment data element value not consistent with data type.
08	04	TR1261	416	Mandatory data element missing in service segment.
08	04	TR1262	417	Service segment data element too small.

Interchange translation errors

The following table describes the errors indicating the translator was able to isolate an interchange to process, but there is a serious problem with that interchange. None of the transactions in the interchange are processed. An API program receives a single 8,5 return code, indicating that an interchange is being skipped.

Return code	Extended code	Msg ID	Unique code	Description
08	05	None	0	An error is detected in the interchange header or trailer. No transactions in the interchange are translated. Processing continues.
08	05	TR0150	401	Functional group header is missing or invalid.
08	05	TR0152	403	Functional group trailer is missing or invalid.
08	05	TR0201	501	Interchange header contains invalid sender ID. Cannot locate trading partner profile member.
08	05	TR0203	502	Interchange control numbers do not match in header and trailer.
08	05	TR0205	503	Interchange trailer contains invalid functional group count.
08	05	TR0206	504	Password does not match password in trading partner profile.

Return code	Extended code	Msg ID	Unique code	Description
08	05	TR0207	506	Interchange is not defined correctly or the interchange is badly damaged and cannot be parsed.
08	05	TR0208	507	Received interchange segment contains invalid data.
08	05	TR0209	508	Expected delimiter not found. Encrypted data not immediately followed by segment delimiter.
08	05	TR0210	509	Envelope definition not found for received interchange.
08	05	TR0211	510	A duplicate interchange was detected and is being skipped.
08	05	TR1257	511	Mandatory composite missing in service segment.
08	05	TR1258	512	Service segment data element too long.
08	05	TR1259	513	Service segment data element value not defined in validation table.
08	05	TR1260	514	Service segment data element value not consistent with data type.
08	05	TR1261	515	Mandatory data element missing in service segment.
08	05	TR1262	516	Service segment data element too small.

Invalid data errors

The following table describes the errors that indicate that the translator cannot identify the data it is reading from the file as EDI standard data. The translator expects an ISA, UNB, SCH, ICS, BG, or GS segment to signal the start of EDI standard data. Any data occurring before, between, or after any of these valid interchanges is flagged as an error.

Return code	Extended code	Msg ID	Unique code	Description
08	06	None	0	Invalid data found in the input file. Data does not match the format required by the EDI standard.
08	06	TR0200	601	Interchange envelope header is missing or invalid.

Return code	Extended code	Msg ID	Unique code	Description
08	06	TR0204	602	Interchange envelope trailer is missing or invalid.
08	06	TR0842	603	No EDI standard data found in input file.
08	06	TR0202	604	Interchange header found while looking for a trailer.

Environmental and program errors

The following table describes the errors considered so serious that the translator logs the appropriate message for the error and stops processing (**TRABORT** field in the TRCB is set to **Y**).

Return code	Extended code	Msg ID	Unique code	Description
12	10	TR0810	901	Profile read for update failed.
12	11	TR0811	902	Profile write failed.
12	12	TR0812	903	Parameter count is invalid.
12	15	TR0815	904	A request to get main storage failed.
12	16	TR0816	905	Invalid function code.
12	17	TR0817	906	The transaction processor did not end because it is not active. This error also occurs if input records are out of sequence, such as a C record followed by a blank record.
12	27	TR0827	908	You cannot perform this function now. Function code not valid at this time.
12	28	TR0828	909	Input data block size invalid. Must be at least 32K.
12	29	TR0829	910	Output data block size invalid. Must be at least 32K.
12	32	TR0832	911	Transaction Store call failed.
12	36	TR0836	912	Repository read failed.
12	38	TR0838	913	QSAM failed to open file.
12	39	TR0839	914	Mailbox (requestor) profile not found.
12	40	TR0840	915	Standards profile member not found.
12	43	TR0843	907	Receive file name in mailbox (requestor) profile is blank.
12	44	SA0042	921	Access denied to function within resource.
12	46	TR0846	922	Raw data control string not found.
12	47	TR0847	923	Failed to find network security profile member.

Return code	Extended code	Msg ID	Unique code	Description
12	48	TR0848	924	Failed to load a user exit routine.
12	49	TR0849	925	Error returned by user exit routine.
12	51	TR0851	926	An data format ID is required when raw data is specified.
12	01	TR1201	916	A program error occurred during anchor processing.
12	02	TR1202	917	Unable to release main storage.
12	03	TR1203	918	Unable to read repository.
12	05	TR1205	919	QSAM failed to read file.
12	06	TR1206	920	QSAM failed to close file.
12	07	TR1207	927	DataInterchange ended because of severe error generating functional acknowledgments.
12	08	TR1208	928	DataInterchange ended because of severe error updating the management reporting databases.
12	09	TR1209	929	Error reading envelope control string.
12	52	TR1252	930	Error reading target ADF control string.
12	53	TR1253	931	No beginning/ending structure.
12	54	TR1254	932	No internal trading partner ID value.
12	55	TR1255	933	User exit (IUSEREXIT) not defined.
12	56	TR1256	934	IUSEREXIT returned an error.
12	57	TR1257	935	Mandatory composite missing in service segment.
12	58	TR1258	936	Service segment data element too long.
12	59	TR1259	937	Service segment data element value not defined in validation table.
12	60	TR1260	938	Service segment data element value not consistent with data type.
12	61	TR1261	939	Mandatory data element missing in service segment.
12	62	TR1262	940	Service segment data element too small.
12	63	TR1263	941	Database error obtaining lock.

Communication return codes

This section describes the codes that can be returned by the communication routine. Program your application to inspect the return code after each call and take appropriate action. When the communications routine receives an error from the network program, it logs an error (message ID VN1015) and returns with extended return code of **15**, and a return code of **4**, **8**, or **12** depending on the severity of the error.

The network message handler must process the network output file, and set the **NPSEVER** and **NPERRCD** fields of the CMCB to indicate the severity and error code for any errors contained in the network output file. For more information, see “Message handler” on page 559. Communications then logs the VN1015 message with a symptom string that includes:

```
NETWORK PROGRAM_RC=xx NETWORK PROGRAM_ERC=yyyy
```

Where: **Is the value of:**

xx NPSEVER

yyyy NPERRCD

Communication return codes are divided into four categories. Each type of return code contains several extended return codes that provide detailed information about the error.

Return code	Explanation
0	Successful completion of the communication request.
4	A warning message. DataInterchange does not end a send or receive operation for this return code.
8	An error has occurred and communication processing might discontinue.
12	A severe error has occurred and communication processing has ended.

Communication extension codes

The following tables describe the meanings of the extension codes for each communication return code.

Communication warnings

Return code	Extended code	Description
04	0000	Default values used for trading partner profile.
04	0001	Network profile does not provide name of send/receive program.
04	0002	QSAM error prevented completion of request to clear the file (applies to sending only).
04	1015	Invalid parameter passed to network program. Check network profile, trading partner profile, or communication interface control block.
04	1040	No data received on a receive request.
04	1041	Not able to process the network command output file.

Communication nonsevere errors

Return code	Extended code	Description
08	0001	No profile data passed.
08	0002	Profile ID not found.
08	0003	Profile key not found.
08	1001	Invalid function code passed.
08	1002	Trading partner ID not found in communication interface control block.
08	1003	Profile ID not found.
08	1004	Profile key not found.
08	1007	Open file error.
08	1012	Mismatched network command and network ID.
08	1013	Invalid account type passed.
08	1014	Invalid data type passed.
08	1015	Invalid parameter passed to network program.
08	1024	A restart was requested but the network is not in a restart situation; request was terminated.
08	1025	A restart is required for the network; request was terminated.
08	1026	A restart was requested but restart is not supported; request was terminated.

Communication severe errors

Return code	Extended code	Description
12	0004	Profile services error.
12	0005	Failed to start communication routine.
12	1005	Profile services error.
12	1006	Get storage failed.
12	1008	Write file error.
12	1009	Close file error.
12	1010	Release storage failed.
12	1011	Network program not found.
12	1015	Invalid parameter passed to the network program.
12	1016	Network program execution failed.
12	1018	An error occurred while attempting a restart.

Status update return codes

The results of a status update request are posted in the return code and extended return code fields of the CCB. The following table describes the status update return codes.

Return code	Extended code	Description
0	0	Status update was successful.
8		Status update failed.
8	210	No interchange was found.
8	211	Internal program error.
8	212	Error attempting to update the database.

SYNC initialization return codes

The results of the initialization request are posted in the return code and extended return code fields of the CCB. The following table describes the initialize SYNC return codes.

Return code	Extended code	Description
0	0	Initialization was successful.
12	12	Initialization failed due to insufficient virtual storage.

COMMIT request return codes

The results of COMMIT requests are posted in the return code and extended return code fields of the CCB. The following table describes the COMMIT return codes.

Return code	Extended code	Description
0	0	COMMIT work successful.
4	1	COMMIT ignored. The request to COMMIT was ignored for one of the following reasons: <ul style="list-style-type: none">• An interval of -1 was specified for SYNCPOINT.• The interval has not been reached yet.
12	N	COMMIT failed. The SQLCODE from DB2 indicating why the COMMIT was not honored.

ROLLBACK request return codes

The results of ROLLBACK requests are posted in the return code and extended return code fields of the CCB. The following table describes the ROLLBACK return codes.

Return code	Extended code	Description
0	0	ROLLBACK work successful.
12	N	ROLLBACK work failed. The SQLCODE from DB2 indicating the reason for ROLLBACK failure.

Using sample JCL

DataInterchange Utility (EDIUTIL) JCL

This section describes the DataInterchange Utility JCL. Sample Utility JCLs are distributed in the JCL distribution library (EDI.V4R1M0.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 may 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 may cause the file allocation to be changed to a size that is too small to accommodate the data written to it during repetitive DataInterchange processing. If you decide to use the RLSE keyword with these files, make sure to delete and re-allocate these files before each execution of DataInterchange.

EDIQUERY	PURCORD
FFSEXCP	REQ1DD
FFSTRAK	REQ2DD
FFSWORK	RPTFILE
INVOICE	QDATA

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 DataInterchange Utility is 4096 K.

```
//EDIUTIL JOB (INSTALLATION DEPENDENCIES, REGION=4096K)
//*
//*****
/* This sample JCL will invoke the DataInterchange 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 than English is installed, change all      *
/*      "EDIENU." to "EDI###.", where "###" represents the proper      *
/*      language identifier value shown in the program directory.      *
/*      *****

```

Section 2 DataInterchange Utility parameters

When invoking the DataInterchange Utility, certain parameters are passed in. These parameters are keyword-oriented. The keywords should be separated by at least one blank character. Valid keywords are:

APPLID	The application ID. The default is EDIFFS . This keyword also identifies the log file as specified in the ACTLOGS profile.
LANGID	The language ID. The default is ENU . The value supplied must match an entry in the LANGPROF profile. This parameter is used to control date formats, decimal notations, and other language-specific fields.
SYSID	The installation-defined DataInterchange system ID that controls access to various components of DataInterchange. The default is DIENU . The value in this field is part of the resource name defined using RACF or another resource control product.
DLM	The delimiter used in place of left and right parentheses to enclose DataInterchange 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 may come from the EDITSIN data set.

For more information, see “DataInterchange/MVS and DB2 attachment” on page 294.

In DB2, change the parameter PLAN=EDIENU41 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 DI/MVS RACF installation.
/*      2. If necessary, change the "APPLID=EDIFFS" parm to match
/*      the desired DI/MVS 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 DataInterchange 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 LNKLIST, or loading all necessary programs into the link pack area).

In DB2, STEPLIB must also include the DB2 load library.

```
//STEPLIB DD DSN=EDI.V4R1M0.SEDILMD1,DISP=SHR <--EDI LOAD LIBRARY
// DD DSN=SYS1.SNA.LOADLIB,DISP=SHR <--COMM. LOAD LIBRARY
```

Section 4 PRTFILE

PRTFILE is required for all DataInterchange Utility functions. Error messages generated during function processing and a summary report are written to PRTFILE. This data set contains fixed block addressing (FBA) or variable block addressing (VBA) records with a logical record length of 132.

```
//*****
/* Specify the Audit print file. In this case the summary report      *
/* generated by the utility will be returned to JES. You may 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 DataInterchange 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, DataInterchange 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.TRADING.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 NETPROF 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 *
/*           may not satisfy YOUR network requirements, so it    *
/*           is suggested that you verify this allocation.        *
/******
//QDATA      DD DSN=NETWORK.HOLDFILE.NAME,DISP=MOD
//*
```

Section 10 Network communications

This group of JCL statements is used when sending to or receiving from a network. The JCL shown here is required for the AT&T Global Network. Each network has its own JCL requirements.

Section 10a Communicating with Expedite Base/MVS using IEBASE

This section contains JCL statements for using the Expedite Base/MVS network program IEBASE. The network profiles supplied by DataInterchange are IINB41 (IEBASE Release 4.1) and IINB42 (IEBASE Release 4.2 and higher). The files described here are required by Expedite Base/MVS. INMSG is the network input file and contains commands written by DataInterchange for processing by Expedite Base/MVS. OUTMSG is the network output file and contains the responses from Expedite Base/MVS after processing. INB1STAT is used during status updating to hold network acknowledgments. For more information about these Expedite Base/MVS files, refer to the *Expedite Base/MVS Programming Guide*.

```

//*****
/* Specify the IGN Expedite Base/MVS network program (IEBASE)    *
/* required work files. The network profile name is IINB42.      *
/*                                                                *
//*
```

```

/* DSNAME      DESCRIPTION                                RECFM      LRECL      *
/*
/* * ERRORMSG   ERROR DESCRIPTIONS FILE                  FB          80      *
/* * ERRORTXT   EXTENDED ERROR DESCRIPTIONS FILE         FB          80      *
/* * INB1STAT   STATUS INFORMATION FILE                   V          255     *
/* * INMSG      MESSAGE COMMAND FILE                     FB          80      *
/* * INPRO      PROFILE COMMAND FILE                     FB          80      *
/* * OUTMSG     MESSAGE RESPONSE FILE                    FB          80      *
/* * OUTPRO     PROFILE RESPONSE FILE                    FB          80      *
/* *
/* * NOTE:      INPRO in sample JCL needs to be customized for your      *
/* *            system.                                                  *
/* *
/* *****
/* *
/* ERRORMSG DD DSN=SYS1.EXPBASE,EXPMSG( ERRORMSG ),DISP=SHR
/* ERRORTXT DD DSN=SYS1.EXPBASE,EXPMSG( ERRORTXT ),DISP=SHR
/* INB1STAT DD DSN=NETWORK.INB1STAT,DISP=OLD
/* INMSG    DD DSN=SYS1.EXPBASE,EXPMSG( INMSG ),DISP=SHR,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, then a qualifier table (QUTTABLE) and one or more destination tables (TTABLExx) may be necessary. The ddname of the destination table(s) may be specified in the data of the qualifier table and can be any valid ddname. If the qualifier table (QUTTABLE) is not used, then the following default ddnames are used for the destination table(s):

Envelope Type	ddname
X12	TTABLExx - where xx is the two-character qualifier (ISA07 element of the interchange envelope)
UCS	TTABLE01
UN/TDI	TTABLE
EDIFACT	TTABLExx - where xx is the first two characters of the qualifier (UNTO:2 element of the interchange envelope)

```

//QUTTABLE DD DSN=USERFILE.IN.ANYNAME.SEQ.FILE,DISP=SHR
//TTABLExx DD DSN=USERFILE.IN.ANYNAME.SEQ.FILExx,DISP=SHR
/*

```

Section 10b Communicating with Expedite Base/MVS using IEBase and Comm-Press

this section contains JCL statements for using the Expedite Base/MVS network program IEBase with Comm-Press. In addition to the files mentioned in Section 10a Communicating with Expedite Base/MVS using IEBase above, these files are required if you are using Expedite Base/MVS with compression. For more information about these Expedite Base/MVS files, refer to the *Expedite Base/MVS Programming Guide*.

```

/* *****
/* * If you are using Expedite Base/MVS with Comm-Press, then specify *
/* * the following work files:                                         *
/* *
/* * DSNAME      DESCRIPTION                                RECFM      LRECL      *
/* *

```

```

/* * COMPPDS      COMPRESSION PDS FILE                FB          80      *
/* * COMPWRK      COMPRESSION WORK FILE               FB          80      *
/* * INMSGC       MESSAGE COMMAND FILE                FB          80      *
/* * INMSGR       MESSAGE RESPONSE FILE               FB          80      *
/* * OUTMSGC      MESSAGE COMMAND FILE                FB          80      *
/* * OUTMSGR      MESSAGE RESPONSE FILE               FB          80      *
/* * SYSUT1       TEMPORARY WORK FILE FOR COMPRESSION FB          80      *
/* *
/* * COMPTRC      COMPRESSION TRACE FILE
/* * CPLOOKUP     COMPRESSION LOOKUP TABLE           FB          80      *
/* *
/* * NOTE:  COMPTRC is required when BASE(Y) is used on the trace
/* *        command. CPLOOKUP is required when using COMPRESS(T).
/* *
/* *****
/* *
/* * COMPPDS  DD DSN=NETWORK.COMPPDS,DISP=(NEW,CATLG),
/* *           DCB=(RECFM=FB,LRECL=80,BLKSIZE=0,DSORG=PO),
/* *           UNIT=SYSALLDA,SPACE=(CYL,(10,2,10))
/* * COMPWRK  DD DSN=NETWORK.COMPWRK,DISP=(NEW,CATLG),
/* *           DCB=(RECFM=V,LRECL=84,BLKSIZE=23440,DSORG=PS),
/* *           UNIT=SYSALLDA,SPACE=(TRK,(10,1))
/* * INMSGC   DD DSN=&&INMSGC,DISP=(NEW,DELETE),UNIT=SYSALLDA,
/* *           DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(1,1))
/* * INMSGR   DD DSN=&&INMSGR,DISP=(NEW,DELETE),UNIT=SYSALLDA,
/* *           DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),SPACE=(TRK,(1,1))
/* * OUTMSGC  DD SYSOUT=*
/* * OUTMSGR  DD DSN=NETWORK.OUTMSGR,DISP=(NEW,CATLG),
/* *           DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
/* *           UNIT=SYSALLDA,SPACE=(TRK,(10,1))
/* * SYSUT1   DD DSN=&&SYSUT1,DISP=(NEW,DELETE,DELETE),
/* *           UNIT=SYSDA,DCB=BLKSIZE=23476,SPACE=(CYL,(1,1))
/* * COMPTRC  DD SYSOUT=*
/* * CPLOOKUP DD DSN=NETWORK.CPLOOKUP,DISP=(NEW,CATLG),
/* *           DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
/* *           UNIT=SYSALLDA,SPACE=(TRK,(10,1))
/* *

```

Section 11 EDI standards

For receiving and deenveloping transaction data, this section of JCL defines the EDI standard data file to receive into and deenvelope from. The ddname should match one of the following:

- The **Receive file name** field value as specified in the mailbox (requestor) profile (REQPROF) associated with the **REQID** keyword in the PERFORM command. There must be a ddname here for each requestor that can be processed during the run.
- The value specified with the **FILEID** keyword in the PERFORM command. The value specified here overrides the **Receive file name** value from the mailbox (requestor) profile.

This data set can contain fixed-length or variable-length records with a logical record length of 80 or greater.

```

/* *****
/* * If this job step includes receiving and/or deenveloping of
/* * transaction data:
/* *
/* * Specify the Network Transaction files to receive into and/or
/* * deenvelope from.
/* *
/* * The ddname MUST match the receive file name in the Requestor
/* *

```

```

/* profile or the FILEID override keyword in the EDISYSIN input      *
/* command language statements.                                     *
/* A ddname is required for each unique ddname specified by a      *
/* Requestor ID (receive ddname obtained from Requestor profile).  *
/* Use DISP=MOD to append data to the file                          *
/* DISP=OLD to replace the file                                     *
/* *****                                                         *
/*                                                                    *
/*REQ1DD      DD DSN=REQ1.TRANS,DISP=MOD
/*REQ2DD      DD DSN=REQ2.TRANS,DISP=MOD
/*

```

Section 12 ddname

This section defines the ddname for receiving data. The ddname is specified in the **Application file name** field in the data format. You must specify a ddname for each data format that can be processed during the run. The name specified in the data format can be overridden with a value in the **Application file name** field in the receive usage/rule. If there is no applicable DD statement, the application data is written to the exception file (FFSEXCP). In the example below, INVOICE identifies a file that holds data in application format when translating from EDI standard format.

This data set can contain fixed or variable length records. The logical record length must be large enough to hold the largest application record.

```

/* *****                                                         *
/* If translating to application formats:                             *
/*                                                                    *
/* Specify the sequential application (output) files                 *
/*                                                                    *
/* ddnames MUST be supplied for every Application Data Format        *
/* that can be received in this job and they must match the        *
/* ddname specified in defining the Application 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 may request.                  *
/* *****                                                         *
/*                                                                    *
/*INVOICE     DD DSN=INVOICE.DATA.FILE,DISP=MOD
/*PURCORD     DD DSN=PURCORD.DATA.FILE,DISP=MOD
/*

```

Section 13 Report file

This section defines the file that contains reports generated by the PRINT command and by various other PERFORM commands.

This data set can contain fixed block addressing (FBA) or variable block addressing (VBA) records with a logical record length of 132.

```

/* *****                                                         *
/*                                                                    *
/*                               PLEASE READ!!!!                     *
/*                                                                    *
/* The following DD assignments (up to the EDISYSIN DD assignment)  *
/* are used for specific commands which most likely will not be    *

```



```

/* used too often. Please read the comments carefully to determine   *
/* if the ddnames should be allocated or removed.                     *
/******
/*
/******
/* If using any of the PRINT commands:                                *
/*
/* Specify the output file to write the reports out to.              *
/*
/* Use DISP=MOD to append to the existing file                       *
/*     DISP=OLD to replace the file                                   *
/* Allocation parameters should indicate a record length of 133.     *
/******
/*
//RPTFILE    DD DSN=REPORT.FILE,DISP=MOD
/*

```

Section 14 Output file

This section defines the file that contains output from command processing. EDIQUERY identifies a file that contains output from QUERY commands and from various other PERFORM commands.

This data set should contain variable length records with a block size of 32760.

```

/******
/* If using the following commands:                                   *
/*     TRADING PARTNER PROFILE DATA EXTRACT                         *
/*     TRADING PARTNER CAPABILITY DATA EXTRACT                     *
/*     TRANSACTION ACTIVITY DATA EXTRACT                           *
/*     NETWORK ACTIVITY DATA EXTRACT                               *
/*     TRANSACTION DATA EXTRACT                                    *
/*     ENVELOPE DATA EXTRACT                                       *
/*     QUERY                                                         *
/*
/* Specify the output file to write the extracted data or matching   *
/* handles to.                                                       *
/*
/* Use DISP=MOD to append to the existing file                       *
/*     DISP=OLD to replace the file                                   *
/*
/* Allocation parameters should indicate a variable record format    *
/* with a block size of 32760. This will allow any type of data     *
/* to be written to the query file without the possibility of       *
/* records being truncated.                                          *
/******
//EDIQUERY   DD DSN=QUERY.FILE,DISP=MOD
/*

```

Section 15 Export/Import statements

This section of JCL is required for exporting and importing.

```

/******
/* If using the EXPORT or IMPORT commands:                           *
/*
/* 1) Allocate the export/import DI control file.                   *
/*
/* 2) Allocate the export/import user supplied control file.        *
/*     This control file describes what data is to be exported or   *
/*     imported. An allocation of fixed format and record length 84 *

```

```

/*      should be used. This file can also be allocated inline.          *
/*      Make sure to use the same ddname allocated with the              *
/*      CTLFILE keyword in the EDISYSIN input command language           *
/*      statements. In this sample CTLFILE(EXIMCTL) would be used.       *
/*                                                                           *
/* 3) Allocate the DD statements for the export/import files              *
/* themselves. All the ddnames can point to a single file                 *
/* or they can each point to a separate file. This sample               *
/* JCL shows the ddnames pointing to separate files.                     *
/* Allocation of the dataset(s) should have a record format              *
/* of variable, a record length of 4089, and a block size of 4093.*
/* Use DISP=MOD to append to the existing file(s)                        *
/*      DISP=OLD to replace the file(s)                                   *
/******
/*

```

Section 15a Export/Import control statements

This section defines the control statements describing what data should be exported or imported. This ddname must match the value specified in the **CTLFILE** keyword on the PERFORM command. The export and import control statements are described in “Export/Import control file (CTLFILE)” on page 187.

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

```

/******
/* Step 1 - Allocate user supplied control file
/******
/*
//EXIMCTL DD DSN=EXPORT.IMPORT.CTLFILE,DISP=OLD

```

Section 15b Export/Import files

The remaining EDIEIxxx DD statements define files to which data is exported, or from which data is imported. You can assign all DD statements to a single physical file, or separate them as shown in this example. These data sets contain variable length records with a logical record length of 4089.

```

/******
/* Step 2 - Allocate export/import files
/******
/*
//EDIEIADF DD DSN=EXPORT.IMPORT.ADF,DISP=MOD
//EDIEICST DD DSN=EXPORT.IMPORT.CST,DISP=MOD
//EDIEIPRF DD DSN=EXPORT.IMPORT.PRF,DISP=MOD
//EDIEISTD DD DSN=EXPORT.IMPORT.STD,DISP=MOD
//EDIEITBL DD DSN=EXPORT.IMPORT.TBL,DISP=MOD
//EDIEITPT DD DSN=EXPORT.IMPORT.TPT,DISP=MOD
/*

```

Section 16 Functional acknowledgment overrides

This section defines override data that the translator uses to generate functional acknowledgments. FAENV is an optional file processed during deenvelope functions. For detailed information about the format of this file, see “Enveloping options file for functional acknowledgments (FAENV)” on page 182.

This data set can contain fixed or variable length records. The logical record length must be as large as the largest record in the file.

```

//*****
/* If deenveloping data:
/*
/* Specify the sequential file which contains overrides
/* that you want used when functional acknowledgments are
/* generated for the data that is being deenveloped.
/*
/* This file is optional and if not specified the envelope data
/* for functional acknowledgments will be taken from the standard
/* profile member specified for the transactions.
/*
/* Suggested: DCB=(RECFM=V,LRECL=255)
//*****
//FAENV      DD DSN=FA.OVERRIDES.FILE,DISP=SHR
//

```

Section 17 Perform command file

This section defines the file from which the Utility PERFORM commands are read. If EDISYSIN is not defined, the Utility checks SYSIN for the PERFORM commands. For more information on these commands, see Chapter 2, “DataInterchange commands and keywords.”.

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 DataInterchange Utility will first look for PERFORM
/* commands in EDISYSIN. If there is no EDISYSIN defined,
/* the Utility will look in SYSIN.
//*****
//EDISYSIN DD *
* An asterisk in column one denotes a comment line..
* -----
* Here is a sample of command language input to translate and
* envelope transactions from application file APDATA01 and generate
* optional records.
* -----
PERFORM TRANSLATE AND ENVELOPE
WHERE APPFILE(APDATA01) OPTRECS(IEGTQ)

* -----
* Here is a sample of command language input to send queued
* transactions on behalf of requestor ID ROBOX
* -----
PERFORM SEND WHERE REQID(ROBOX)

* -----
* Here is a sample of command language input to receive transactions
* from requestor ID ROBOX
* -----
PERFORM RECEIVE WHERE REQID(ROBOX)

```

```

* ----- *
* Here is a sample of command language input to deenvelope and
* translate received transactions associated with requestor ID ROBOX
* ----- *
PERFORM DEENVELOPE AND TRANSLATE
WHERE REQID(ROBOX)
/*

```

Section 18 Pageable translation work file

This section identifies the pageable translation work file, EDIVAX. EDIVAX should be defined as a temporary data set. Space allocation is dependent on the amount of data that is paged. Pageable translation is specified by using the **PAGE** keyword available on all translate type commands, and on all envelope/deenvelope type commands. For information regarding EDIVAX space allocation and pageable translation in general, see the **PAGE** keyword description on page 152.

```

/*****
/* 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=SYSALLDA,SPACE=(CYL,25)
/*

```

Section 19 DB2 parameters

This section is for DB2 only. It is typical in a DB2 environment that batch application programs (such as the DataInterchange 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 DataInterchange Utility parameters are specified in SYSTSIN. See "Section 2 DataInterchange Utility parameters" on page 696 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 (EDIENU41) 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 DI/MVS RACF installation.      *
/* 5. If necessary, change the parm "APPLID=EDIFFS" to match      *
/* the desired DI/MVS application ID.      *
/* 6. If necessary, change the parm "PLAN=EDIENU41" 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 (EDIENU41) -
    PARM('LANGID=ENU SYSID=DIENU APPLID=EDIFFS PLAN=EDIENU41 SYSTEM=DSN')
    END
/*

```

Section 20 XML parameters

This section is for XML processing. The EDI.XML parameters (in the last line of code) should be edited as appropriate for your installation of XML Toolkit Version 1 Release 2, replacing SYS1 with the high level qualifier for your system.

```

//STEPLIB DD DSN=EDI.V4R1M0.SEDILMDI,DISP=SHR
          DD DSN=EDI.V4R1M0.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

The table below lists outlines the sections of JCL required for the different DataInterchange 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.

Command	JCL sections						
DEENVELOPE	6	11	16	17			
DEENVELOPE AND TRANSLATE	6	7*	11	12	16*	17	
ENVELOPE	6	7*	8	9	17		
ENVELOPE AND SEND	6	7*	8	9	10	17	
ENVELOPE DATA EXTRACT	14	17					
EXPORT	15	17					
HOLD	17						
IMPORT	15	17					
NETWORK ACTIVITY DATA EXTRACT	14	17					
PRINT	13	17					
PURGE	17						
QUERY	14	17					
RECEIVE	10	11	17				
RECEIVE AND DEENVELOPE	6	10	11	16*	17		
RECEIVE AND SEND	6	7*	9	11	16*	17	
RECEIVE AND TRANSLATE	6	10	11	12	16*	17	
RECVFILE AND SEND	6	7*	8	9	10	11	17
REENVELOPE	6	7*	8	9	17		
RELEASE	17						
REMOVE TRANSACTIONS	17						
RETRANSLATE TO APPLICATION	6	12	17				
SEND	9	10	17				
TRADING PARTNER CAPABILITY DATA EXTRACT	14	17					
TRADING PARTNER PROFILE DATA EXTRACT	14	17					
TRANSACTION ACTIVITY DATA EXTRACT	14	17					

Command	JCL sections						
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.V4R1M0.SEDIINS1). For additional information, see "Reporting and extracting data" on page 7.

```
//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 (ARCFIL) AND THE HOLD FILE (HLDFILE) DD    *
// * STATEMENTS AS NECESSARY.                                          *
// * 3. IF NECESSARY, CHANGE THE PARM "DIENU" IN THE STEP "ARCHIVE"    *
// * FOR YOUR DI/MVS 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 ARCFIL,    *
// * AND ALL OTHER ENTRIES WILL BE COPIED TO THE HLDFILE.            *
// * *****
//CREATE EXEC PGM=IEFBR14
//ARCFIL DD DSN=EDIENU.V4R1M0.ARCFIL,DISP=(NEW,CATLG),
// DCB=(RECFM=V,LRECL=4096,BLKSIZE=0),
// UNIT=SYSALLDA,SPACE=(CYL,(10,5))
//HLDFILE DD DSN=EDIENU.V4R1M0.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.V4R1M0.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)
//ARCFIL DD DSN=EDIENU.V4R1M0.ARCFIL,DISP=SHR
//HLDFILE DD DSN=EDIENU.V4R1M0.HLDFILE,DISP=SHR
// *
//SYSIN DD *
PERFORM UNLOAD LOG ENTRIES WHERE APPLID(EDIFFS)
LOGDATE(10/01/01) TO(12/31/01)
ARCHIVEFILE(ARCFIL) HOLDFILE(HLDFILE)
// *
//SYSTSIN DD *
DSN SYSTEM (DSN)
RUN PROG (EDIFFUT) -
PLAN (EDIENU41) -
PARM('LANGID=ENU SYSID=DIENU APPLID=EDIFFS PLAN=EDIENU41 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 TABLESPACE (EDIENU41.EDIELOG)
UNLDDN (UNLD)
WORKDDN (UNLD)
REORG INDEX (EDIENU41.EDIELOGX)
/*

```


Performance considerations

This Appendix provides performance information on DataInterchange/MVS and DataInterchange/MVS-CICS to help you plan your system requirements.

The following considerations and tuning techniques are highly recommended to achieve optimum DataInterchange/MVS performance:

- General optimization techniques for either batch or CICS
- DataInterchange/MVS considerations for batch
- DataInterchange/MVS-CICS considerations for CICS only
- File maintenance techniques
- Transaction Store query techniques
- VS COBOL II Field Exit considerations

General optimization techniques

- If you are a high-volume EDI customer using a DB2 repository:
 - Define multiple DASD volumes to DB2 storage group EDISTG01 to facilitate spreading DB2 data sets across multiple packs.
 - Define large DASD allocations for Transaction Store DB2 tables EDITSAU, EDITSTH, EDITSTI, EDITSTU, EDITSEV, EDITSGP which are heavily referenced.
 - Monitor the status of DB2 Table data sets on a regular basis (at least twice monthly, more often if possible). Use LISTCAT and DB2 Utilities such as CHECK, RUNSTAT, and REORG to monitor and tune the DB2 table data sets.
 - Add volumes to DB2 storage groups, increase table space DASD, and reorganize DB2 Table data sets when CI, and especially CA, splits are encountered.
- Define the Master Catalog for the high-level data set name qualifier on a separate pack from all other MVS data sets.

- Log only application data images and EDI standard data images during the testing period. The trading partner profile (TPPROF) controls logging of interchange images. The usage/rule controls the logging of application images. Additionally, do not log profile access attempts if not necessary for your environment. The **Log** action on the Profile menu controls this.
- Log only events during the testing period. The ACTLOGS profile, member EDIFFS, **Log** flag controls whether normal events are logged. Error conditions and events that change a transaction's status are recorded even if logging is turned off.
- Minimize translator error messages by setting the &DIERRFILTER keyword.
- Use commands that combine several functions in one; for example, rather than using TRANSLATE TO STANDARD and then ENVELOPE, use TRANSLATE AND ENVELOPE to have both functions performed at the same time.
- If concurrency is an issue and multiple DataInterchange 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, may be advisable here to help concurrency and improve throughput. This can be accomplished using the RECOVERY keyword.
- Specify that DataInterchange 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.

DataInterchange/MVS considerations

- Define FFSWORK data set in the DataInterchange/MVS 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 DataInterchange/MVS 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.)

$$\begin{array}{rclclcl} \text{Virtual storage} & = & \text{Largest application} & + & \text{EDI interchange} & + & 4 \text{ MB overhead} \\ \text{required} & & \text{transaction image} & & \text{length} & & \end{array}$$

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 DataInterchange 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.

EDI standard	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. DataInterchange obtains most storage above the 16 MB line.



NOTE: The MVS 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 MVS JCL documentation.

DataInterchange/MVS-CICS considerations

- For CICS/ESA installations, the DataInterchange Persistent Environment can be used. The Persistent Environment is designed to minimize read operations involved in translating data.
- Make sure transaction EDIX is enabled. This is done with CEMT I TRAN(EDIX).
- The DataInterchange 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 DataInterchange translations involving numerous trading partners, you can develop HOT-DI applications to drive DataInterchange.
- 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.)

$$\begin{array}{rclcl} \text{Virtual storage} & = & \text{Largest application} & + & \text{EDI interchange} & + & 4 \text{ MB overhead} \\ \text{required} & & \text{transaction image} & & \text{length} & & \end{array}$$

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 DataInterchange 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.

Standard	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. DataInterchange obtains most storage above the 16 MB line.

In CICS, most storage will be obtained from extended DSA. (This is a CICS 4.1 EDSALIN SIT parameter.) You must also consider other CICS activity, such as concurrent DataInterchange/MVS-CICS tasks. Concurrent tasks each require 3 MB of additional overhead.

Maximize throughput by using concurrent DataInterchange/MVS-CICS 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 DataInterchange/MVS data sets on a regular basis. Small data sets result in better performance.
- Do not track unneeded, outdated transactions in the Transaction Store database. Performance is degraded when a database query range includes these unimportant records. Execute PURGE and REMOVE commands regularly to delete unneeded records. Also, use an appropriate purge interval. For more information on the file content and space calculations, see Appendix F, “Space calculation examples.”
- Specify a WHERE clause for REMOVE commands to reduce the scope of the database query to transactions with a PURGE - DATE EXPIRED status. Before transactions can be removed, you must run a database query to determine the remove eligibility of expired transactions. Narrowing the query to only eligible transactions improves performance. For example, if the default purge interval of 30 days is used, transactions added in the last 30 days are not eligible for removal. The command could be: REMOVE TRANSACTIONS WHERE HANDLE(*-999) TO(*-30).
- If the Transaction Store information is not used or needed for your environment, turn off the writing of these records through options in the APPDEFS profile. However, be aware that if ENVELOPE or TRANSLATE TO APPLICATION is used, it is essential that Transaction Store records be written as this ensures their retrieval.
- Archive event log entries regularly. For more information on the archive utility, see Appendix C, “Using sample JCL.” Or, if records therein are not needed, use the sample install JCL to DELETE, DEFINE, and LOAD the log file. Sample JCL is contained in target library SEDIINS1. Member EDIJVSDF contains JCL to delete and define the log. Only run the applicable job step for the event log in question.

Transaction Store query techniques

The WHERE option specified during a database query plays a significant role in performance. A database query takes place with various PERFORM statements, the most common of which are ENVELOPE, TRANSLATE TO APPLICATION, PRINT, PURGE, and REMOVE. These Transaction Store utility functions require a search and retrieval of information contained in the Transaction Store.

The following techniques apply to all PERFORM commands that require a database selection/query. These same techniques can be utilized with the interactive Transaction Store facility. The following list contains criteria and the order in which the criteria are checked. (All fields at the same indentation level are checked in the order listed.)

```
-----
Transaction Handle
Batch ID
Direction

All information in Transaction Handle Record(s)
Table:  EDITSTH

Application Control field
Direction
Standard Transaction ID
Transaction Handle
Date added to store
Time added to store
```

```

Trading Partner Nickname
Internal Trading Partner ID
Network ID
Envelope Type
Earliest purge date
Earliest envelope date

All information in Application Record(s)
Table:  EDITSAU

Application ID
Application 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 DataInterchange 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.

VS COBOL II field exit considerations

If DataInterchange/MVS translation performance degrades after adding a VS COBOL II field exit, it is probably because of a tremendous increase in the EXCP count. This degradation can often be attributed to the LIBKEEP runtime parameter. By default, the runtime parameter LIBKEEP equals N which causes the runtime environment to reinitialize every time a program or sub-program is executed. When a field exit is called by DataInterchange thousands of times, the exception count goes up and performance goes down. The execution option of **LIBKEEP** can be changed to Y in macro IGZEOPD, thus keeping the runtime environment in memory after a GOBACK from the field exit. This is a global change and affects all COBOL applications using this run time library. Consider the implications:

1. A system programmer needs to balance exceptions with region size constraints. The more sophisticated a COBOL program is, the longer it takes to initialize the runtime environment. In the case of DataInterchange field exits, this probably is minimal because these exits typically only work with storage. However, keeping the environment in storage may effect the region sizes of other larger COBOL applications.
2. Consider the environmental issue of both COBOL OS/VS and VSCOBOL II. Because LIBKEEP may keep the COBOL II runtime in memory, COBOL OS/VS programs may access this rather than using strictly COBOL OS/VS. The only impact is that this forces a NO END JOB response in a COBOL OS/VS program.

The problem of the IGZEOPD macro being global can be bypassed by using a separate CSI for a copy of the VS COBOL II run time library. A special run time library can be installed where **LIBKEEP** equals Y. This library can then be STEPLIBed in the DataInterchange/MVS job, so it is not accessed by the rest of the MVS system. This eliminates the impact on other COBOL applications.

The LIBKEEP and RTEREUS options are similar with the exception that RTEREUS provides reusability for concurrent jobs. Both are runtime options and not compiler or link-edit options. For more information about runtime parameters, refer to the *VS COBOL II Application Programming Guide*.

Notices

This publication was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Trademarks and service marks

The following terms in this publication are trademarks or service marks of the International Business Machines Corporation in the United States and other countries:

- Expedite
- IBM
- IBM Global Services
- Information Exchange
- MQSeries

Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or servicemarks of others. All other trademarks and service marks are the property of their respective owners.



Glossary

A

AAR. Association of American Railroads. Represents the railroad industry in areas such as standards, public relations, and advertising.

acknowledgment. See *functional acknowledgment*, *network acknowledgment*.

action bar. The area at the top of a panel that contains choices currently available in the application that is running. Compare to the *function key area*, which contains actions common to all programs.

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 DataInterchange 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 DataInterchange and sets specific DataInterchange 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.

CLIST. See *Command list*.

command line. The line at the bottom of the panel that provides an alternate way of requesting services rather than using the *Action* column of the panel body.

code list. A table, supplied by DataInterchange or defined by the user, that contains all acceptable values for a single data field.

Command list (CLIST). A list of commands and statements designed to perform a specific function for the user.

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

DASD. Direct access storage device.

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. The DataInterchange product; 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 WindowsTM-based parameter entry mechanism (DataInterchange Client), and a translator.

DataInterchange Client. A WindowsTM-based product for entry of parameters needed by the DataInterchange translator.

DataInterchange/MVS. The DataInterchange product used on the host; pieces include a TSO parameter entry mechanism and a translator.

DataInterchange/MVS-CICS. The CICS-based DataInterchange product.

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. DataInterchange Client; the Windows™-based, client/server interface for DataInterchange.

dictionary. See *data dictionary*.

direct access storage device (DASD). A device in which access time is effectively independent of the location of the data.

distribution libraries. Supplied partitioned data sets on tape containing one or more components used to transfer data to a new system.

distribution tape. A magnetic tape that contains the distribution libraries for installing a new system.

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 DataInterchange 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 DataInterchange.

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. See *UN/EDIFACT*.

electronic data interchange (EDI). A method of transmitting business information over a network, between business associates who agree to follow approved national or industry standards in translating and exchanging information.

electronic transmission. The means by which information is transferred between parties, such as over a public network.

element. See *data element*.

element separator. A character that separates the data elements in a segment. See also *data element delimiter*.

encryption. The encoding and scrambling of data. Data is encrypted by the sender and decrypted by the receiver using a predetermined program and unique electronic key.

event. An occurrence that is important to a user's computer tasks, such as a software error, sending a transaction, or acknowledging a message.

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that was derived from, and is a subset of SGML. It is used to represent structured documents and data.

F

field. See *data field*.

floating segment. A segment of an EDI standard that may exist in many positions relative to other segments.

forward translation table. A user-defined table that translates data values that differ between trading partners. For example, if a manufacturer and supplier have different part numbers for the same item, each company can use its own part number and have it converted to the other company's part number during translation. Forward translation tables translate local values to standard values.

functional acknowledgment. An electronic acknowledgment returned to the sender to indicate acceptance or rejection of EDI transactions.

functional group. One or more transaction sets of a similar type transmitted from the same location, enclosed by functional group header and trailer segments.

function key. A key that causes a specified sequence of operations to be performed when it is pressed. Generally used to refer to keys labelled F_n , where n is a number from 1 to 24.

function key area. Two lines at the bottom of the panel that list the active function keys for the panel.

G

GDDM. Graphical data display manager.

global variable. A variable that is shared among all instances of all documents within a translation session.

graphical data display manager (GDDM). An IBM-licensed program that allows your monitor to define and display pictures.

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

ICS. International Control Segments.

import. The process of taking DataInterchange objects exported on another DataInterchange system and incorporating them into the receiving system.

International Control Segments Interactive System Productivity Facility (ISPF). An IBM-licensed program that serves as a full-screen editor and dialog manager.

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.

ISPF. Interactive System Productivity Facility.

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

Link Pack Area (LPA). In MVS, an area of main storage containing reenterable routines from system libraries. Their presence in main storage saves loading time.

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.

LPA. Link pack area.

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 DataInterchange to define your mailboxes and any associated preferences.

map. A set of instructions that indicate to DataInterchange 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.

member. A collection of data for one entry in a profile. For example, a member of the trading partner profile contains data about one trading partner.

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 DataInterchange Client logs messages about errors that occur within the client.

MQSeries. A product of the IBM company; used to implement messaging and queueing of data groups.

MQSeries Queue profile. Represents a relationship between a logical name and a physical MQSeries queue name.

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.

MVS. Multiple virtual storage.

N

network acknowledgment. A response from the network indicating the status of an interchange envelope, such as sent or received.

network commands. The commands that you want DataInterchange to pass to your network, defined in the network commands profile. In the host product, this file is named NETOP.

Network Profile. The DataInterchange Client terminology for NETPROF members on DataInterchange Host.

O

ODETTE. Organization for Data Exchange through Teletransmission in Europe.

P

panel body. The area in the middle of the panel that contains entry fields, lists of selectable items, menu choices, and scrollable text.

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.

PDF. Program Development Facility.

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.

quiesce. To end a process by allowing operations to complete normally.

quiescing. The process of bringing a device or a system to a halt by rejection of new requests for work.

R

RACF. Resource access control facility.

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 DataInterchange uses to process your data.

requestor. See *mailbox*.

Resource Access Control Facility (RACF). An IBM-licensed program that provides for access control by identifying and verifying the users to the system, authorizing access to protected resources, logging the detected unauthorized attempts to enter the system, and logging the detected accesses to protected resources.

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 DataInterchange translator, such as control strings, code lists, translation tables and profiles.

S

SAF. System Authorization Facility.

security administrator. The person who controls access to business data and program functions.

SAP. A German company named Systeme, Anwendungen, and Produkte specializing in application software. A major product, SAP R/3, is a component-based architecture/application that integrates business processes, such as sales, materials management, and distribution.

SAP R/3 supports an EDI interface subsystem. SAP R/3 generates application data in the SAP R/3 Intermediate Document (IDOC) layout. This data is then sent to the EDI subsystem via a file transfer product, such as FTP or TCP/IP.

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. One 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.

SMP/E. System Modification Program Extended.

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 DataInterchange, rather than constant values. For example, to use today's date, you enter &DATE.

SQL. Structured query language.

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.

System Modification Program Extended (SMP/E). An IBM-licensed program used to install software and software changes on OS/VS1 and OS/VS2 systems.

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.

Time Sharing Option (TSO). A component of the IBM MVS operating system that allows users full access to MVS functionality, but shares machine resources across users.

Time Sharing Option Extensions (TSO/E). The base for all TSO enhancements. It provides MVS users with additional functions, improved usability, and better performance.

TPT. Trading partner transaction. See *map*.

trading partner profile. The profile that defines your trading partners, including information about network account numbers, user IDs, who pays for network charges, etc.

trading partners. Business associates, such as a manufacturer and a supplier, who agree to exchange information using electronic data interchange.

trading partner transaction. See *map*.

trailer. A control structure that indicates the end of an electronic transmission.

transaction. A single business document, such as an invoice. See also *EDI transaction*.

transaction set. A group of standard data segments, in a predefined sequence, needed to provide all of the data required to define a complete transaction, such as an invoice or purchase order. See also *EDI transaction set*.

Transaction Store. The file that contains the results of translations and a history of translation activity.

transform. The process of converting a document from one format to another.

transient data queue (TD). A sequential data set used by the Folder Application Facility in MVS/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.

TSO. Time Sharing Option.

TSO/E. Time Sharing Option.

TSQ. See *temporary storage queue*.

XML. See Extensible Markup Language.

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 DataInterchange 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

WINS. Warehouse Information Network Standard.

WindowsTM. Microsoft's graphical operating system under which DataInterchange Client runs.

X

X12. A common EDI standard approved by the American National Standards Institute.



Index

Symbols

&ASSERT literal 123
&BOUNDARY literal 336

Numerics

000 record 192
0C1/0C2 record 191
1Y1 record 212
1Y2 record 213
1Y3 record 213
1Y4 record 214
1Y5 record 214
1Y6 record 214
1Y7 record 215
1Y8 record 215
1Y9 record 216
1YA record 216
2W1 record 218
2W2 record 219
2W3 record 219
2W4 record 220
2W5 record 220
2W6 record 221
2W7 record 221
2W8 record 221
2W9 record 222
2WA record 222
2WB record 223
3T5 record 232
3T6 record 233
3T7 record 234

3V1 record 226
3V2 record 227
3V3 record 228
3V9 record 229
3VA record 229
3VB record 230
3VC record 230
3VD record 230
3VE record 231
7A1 record 210
7P1 record 194
7P2 record 194
7P3 record 209
7P4 record 209
7Z1 record 210

A

abends, CICS return codes 295
account key
 alternate 434
 full 433
acknowledgment images data extracts, record
 layout 288
activity log profile 247
administrative data
 exporting 12
 importing 12
AJ1 record 242
AJ2 record 242
AJ3 record 243
API
 business tasks 302

- cancel send 424
 - parameters 424
 - close and queue interchange
 - parameters 382
 - commit work parameters 439
 - data extraction parameters 402
 - deenveloping 384
 - parameters 385
 - end translation/enveloping 384
 - parameters 384
 - ENVELOPE AND SEND command 303
 - enveloping 368, 369
 - parameters 369
 - environmental services
 - initializing 304
 - initializing parameters 305
 - initializing request results 305
 - terminating 306
 - terminating parameters 306
 - terminating request results 307
 - get envelope parameters 441
 - initialize SYNC parameters 438
 - internal calls 427
 - issue commit 399
 - issue commit parameters 399
 - languages 292
 - link edit 292
 - process network acknowledgments
 - parameters 427
 - put envelope parameters 442
 - queue standard data 427
 - receive parameters 421
 - receive transactions 420
 - restart receive parameters 421
 - restart receive transactions 420
 - restart send transactions 412
 - parameters 413
 - retrieve detailed data 404
 - retrieve group header parameters 400
 - retrieve interchange header parameters 399
 - retrieve transaction acknowledgment image 406
 - retrieve transaction header 400
 - parameters 401
 - retrieve transaction image 406
 - return codes 678
 - commit 694
 - communication 691
 - communication nonsevere 692
 - communication severe 693
 - communication warnings 692
 - data element 681
 - group 686
 - initializing 678
 - interchange 687
 - invalid data 688
 - negative 679
 - normal 680
 - rollback 694
 - segment 682
 - status update 693
 - SYNC initializing 693
 - termination 679
 - transaction environmental 684, 689
 - transaction program 689
 - transaction syntax 683
 - translation 680
 - warnings 680
 - return filename parameters 426
 - rollback work parameters 439
 - send files parameters 417
 - send transactions 412
 - parameters 413
 - stub programs 292
 - translation services
 - first call request 322
 - overrides 323
 - functions 310
 - translate specific 341
 - parameters 342
 - translate-file-to-application 350
 - parameters 350
 - translate-to-application 338, 341
 - testing 341
 - translate-to-standard 311, 317
 - parameters 317
 - translate-to-standard testing 315
 - update status parameters 430
 - utility service
 - initializing HOT-DI 306
 - parameters 306
 - writing custom programs 302 to 303
- application
- control record 229
 - data extracts
 - record layouts 287
 - default profile 247
 - file 175
 - path to network program 408
 - processing flow to network 551
 - terminal attached 496
 - TRCB fields returned 346, 357, 391
 - envelopes 374

TRODB fields returned 347, 358
 application program interface. *See* API. 291
 applications
 response 521
 continuous receive 522
 DataInterchange Utility 521
 transaction 523
 response program types 521
 APPLID
 initializing environmental services 304
 MVS environment 15
 archiving
 event log entries 6
 Assembler
 calls 299
 associated objects
 data formats 217
 EDI standards 211
 export/import 186
 maps 224
 authentication exit
 languages
 Assembler 472
 C 472
 COBOL 471
 parameters 469
 authentication exits 469
 automatic enveloping 312

B

batches 335, 364, 379
 BK2 record 231
 bundles 335, 364, 379
 business tasks, API 302

C

C and D records
 input file 175
 C language, calls 298
 C records
 field descriptions 259
 layout 257
 call exit 481
 calls
 Assembler 299
 C language 298
 samples 298
 COBOL 295
 samples 295

 first call for transaction 321, 344, 345, 354
 first call of session 318, 342, 351
 get envelope 457
 internal 427
 last call for transaction 327, 350, 363
 last call of session 331, 350, 363, 379, 395
 PL/I 296
 samples 297
 put envelope 457
 subsequent 327, 349, 363
 transaction status fields 325
 cancel request
 CMCB fields returned 425
 CMCB initialization 424
 returned information 425
 cancel send
 API 424
 parameters 424
 case sensitive keywords 173
 categories
 reporting 7, 8
 CCB
 field descriptions 582
 layout 581
 overview 291
 CCB returned fields
 call exit 481
 cancel request 425
 commit work request 439
 get data 479
 get envelope 459
 initialize SYNC request 438
 put data 480
 put envelope 459
 receive 422
 return filename request 427
 rollback request 440
 send files 419
 send transactions 415
 update status request 431
 CICS
 abend return codes 487
 abends returns codes 295
 call utility services 504
 communication 560
 condition codes 675
 continuous receive 565, 566
 DB2 setup 497
 thread pools 498
 envelope queue alternatives 489
 environment 485

- control information 486
- DataInterchange Utility 485
- HOT-DI 500
- invoking 486
- results 487
- terminating 504
 - HOT-DI 505
- initializing DataInterchange 501
- network
 - program control 561
 - ZFCBFUNC field 562
- network profile 560
- performance profile layout 256
- persistent environment 14
- post-enveloping programs 489
- pre-enveloping programs 489
- recovery 491
 - application 496
 - DataInterchange 495
 - DataInterchange utility 493
- regions, separate DataInterchange 497
- return codes 504
 - abends 295
- running DataInterchange in a separate region 497
- serial file processing 490
- special considerations 715
- startup considerations 499
- storage mechanisms 487
- TS queues, processing multiple 489
- unit of work 491, 493
 - application 496
 - DataInterchange 495
- clauses
 - SELECTING 18
 - WHERE 18
- close and queue interchange API 382
 - parameters 382
- CLOSE MAILBOX command 23
 - examples 23
- clusters 335, 364, 379
- CMCB
 - field descriptions 622 to 631
 - initialization 413, 418, 421, 424, 426, 428
 - layout 621
 - output fields 411
 - overview 621
- CMCB returned fields
 - cancel request 425
 - file data 419
 - receive request 423
 - return filename request 427
 - transaction data 415
- COBOL
 - authentication exit definition 471
 - calls 295
 - samples 295
 - encryption exit definition 466
 - field exit definition 451
 - filtering exit definition 476
 - special considerations 720
 - translation exit definition 455
- codes
 - condition 655, 656
 - CICS communication 675
 - communication 674
 - MVS communication 674
 - translation 661
 - data element 664
 - environment 672
 - group 669
 - interchange 670
 - invalid data 671
 - segment 665
 - transaction 666
 - transaction environmental 667
 - warning and normal 663
 - return 655, 656
 - TRANSFORM 659
- combination commands
 - condition codes 677
 - DEENVELOPE AND TRANSLATE 26
 - ENVELOPE AND SEND 31
 - RECEIVE AND DEENVELOPE 66
 - RECEIVE AND SEND 68
 - RECEIVE AND TRANSLATE 69
 - RECONSTRUCT AND SEND 72
 - RCVFILE AND SEND 74
 - REENVELOPE AND SEND 77
 - TRANSLATE AND ENVELOPE 106
 - TRANSLATE AND SEND 108
- command examples
 - CLOSE MAILBOX 23
 - DEENVELOPE 25
 - DEENVELOPE AND TRANSLATE 27
 - DELETE PROFILE 28
 - ENVELOPE 30
 - ENVELOPE AND SEND 32
 - ENVELOPE DATA EXTRACT 34
 - EXPORT 36
 - GLB DUMP 37
 - GLB TRACE 38

- HOLD 40
- IMPORT 41
- LOAD LOG ENTRIES 42
- NETWORK ACTIVITY DATA EXTRACT 43
- PRINT ACKNOWLEDGMENT IMAGE 46
- PRINT ACTIVITY SUMMARY 48
- PRINT EVENT LOG 50
- PRINT STATUS SUMMARY 52
- PRINT STATUS SUMMARY2 54
- PRINT TRANSACTION DETAILS 56
- PRINT TRANSACTION IMAGE 58
- PROCESS NETWORK ACKS 59
- PURGE 61
- QUERY 63
- QUERY PROFILE 64
- RECEIVE 65
- RECEIVE AND DEENVELOPE 67
- RECEIVE AND SEND 68
- RECEIVE AND TRANSLATE 70
- RECONSTRUCT 71
- RECONSTRUCT AND SEND 72
- RCVFILE 73
- RCVFILE AND SEND 74
- REENVELOPE 76
- REENVELOPE AND SEND 78
- RELEASE 80
- REMOVE LOG ENTRIES 81
- REMOVE STATISTICS 82
- REMOVE TRANSACTIONS 84
- REPORT CONTINUOUS RECEIVE STATUS 85
- RESET STATISTICS 86
- RESTART RECEIVE 87
- RESTART SEND 88
- RETRANSLATE TO APPLICATION 90
- SAP STATUS EXTRACT 91
- SAP STATUS REMOVE 92
- SEND 93
- SENDFILE 94
- START CONTINUOUS RECEIVE 95
- STOP CONTINUOUS RECEIVE 96
- TRADING PARTNER CAPABILITY DATA EXTRACT 98
- TRADING PARTNER PROFILE DATA EXTRACT 100
- TRANSACTION ACTIVITY DATA EXTRACT 101
- TRANSACTION DATA EXTRACT 104
- TRANSFORM 105
- TRANSLATE AND ENVELOPE 107
- TRANSLATE AND SEND 109
- TRANSLATE TO APPLICATION 111
- TRANSLATE TO STANDARD 112
- UNPURGE 115
- UPDATE STATISTICS 116
- UPDATE STATUS 117
- command language
 - format 18
 - syntax 17
 - validation 19
- command statements
 - SELECTING clause 18
 - WHERE clause 18
- commands
 - and optional records 272
 - building network 554
 - CLOSE MAILBOX 23
 - combination 66, 68, 69, 72, 74, 77, 106, 108
 - condition codes
 - ENVELOPE AND SEND 677
 - RECEIVE 677
 - TRANSLATE AND SEND 677
 - continuous receive 12
 - data management 6
 - DB2 file 174
 - DEENVELOPE 24
 - DEENVELOPE AND TRANSLATE 26
 - DELETE PROFILE 28
 - ENVELOPE 29
 - ENVELOPE AND SEND 31
 - ENVELOPE DATA EXTRACT 33
 - enveloping 368
 - EXPORT 36
 - exporting 12
 - file 173
 - GLB 14
 - GLB DUMP 37
 - GLB TRACE 38
 - HOLD 39
 - IMPORT 41
 - importing 12
 - inbound processing 5
 - LOAD LOG ENTRIES 42
 - map commands record 231
 - NETWORK ACTIVITY DATA EXTRACT 43
 - network file 175
 - network profile 555
 - command example 557
 - layout 246
 - overview 17
 - persistent environment 14
 - PRINT ACKNOWLEDGMENT IMAGE 45

PRINT ACTIVITY SUMMARY 47
PRINT EVENT LOG 49
PRINT STATUS SUMMARY 51
PRINT STATUS SUMMARY2 53
PRINT TRANSACTION DETAILS 55
PRINT TRANSACTION IMAGE 57
printing 8, 45, 47, 49, 51, 52, 53, 54, 55, 56, 57,
59
PROCESS NETWORK ACKS 59
profile maintenance 12
PURGE 60
QUERY 62
QUERY PROFILE 64
RECEIVE 65
RECEIVE AND DEENVELOPE 66
RECEIVE AND SEND 68
RECEIVE AND TRANSLATE 69
RECEIVE in CICS 5
RECONSTRUCT 71
RECONSTRUCT AND SEND 72
RECVFILE 73
RECVFILE AND SEND 74
REENVELOPE 75
REENVELOPE AND SEND 77
RELEASE 79
REMOVE LOG ENTRIES 81
REMOVE STATISTICS 82
REMOVE TRANSACTIONS 83
REPORT CONTINUOUS RECEIVE
STATUS 85
RESET STATISTICS 86
RESTART RECEIVE 87
RESTART SEND 88
RETRANSLATE TO APPLICATION 89
SAP STATUS EXTRACT 91
SAP STATUS REMOVE 92
SEND 93
fixed-to-fixed translation 4
SENDFILE 94
START CONTINUOUS RECEIVE 95
STOP CONTINUOUS RECEIVE 96
syntax 17
TRADING PARTNER CAPABILITY DATA
EXTRACT 97
TRADING PARTNER PROFILE DATA
EXTRACT 99
TRANSACTION ACTIVITY DATA
EXTRACT 101
TRANSACTION DATA EXTRACT 103
TRANSFORM 105
TRANSLATE AND ENVELOPE 106

TRANSLATE AND SEND 108, 302
TRANSLATE TO APPLICATION 110
TRANSLATE TO STANDARD 112
UNLOAD LOG ENTRIES 113
UNPURGE 114
UPDATE STATISTICS 116
UPDATE STATUS 117
commarea
CICS 561
continuous receive 566
return codes 567
message handler 564
performance monitor user exit 542
comments
trading partner 210
commit work API parameters 439
commits
commit workAPI
CCB fields returned 439
issue commit API 399
return codes 694
common control block (CCB) 291
Comm-Press JCL sample 700
communication
CICS routine 560
CMCB fields 411
FSUPPORT member 558
parameters 554
return codes 691
API warnings 692
nonsevere 692
severe 693
communication data block
See DATABLE. 644
communication services
functions 409
overview 407
return codes 411
communications
condition codes
CICS 675
MVS 674
IEBASE
JCL samples 699, 700
compression exit
languages
Assembler 478
C 477
COBOL 476
parameters 474
compression exits 474

- concatenating data 405
 - condition codes 655
 - combination commands 677
 - communication 674
 - CICS 675
 - MVS 674
 - DataInterchange 656
 - overriding 22
 - translation 661
 - data element 664
 - environment 672
 - group 669
 - interchange 670
 - invalid data 671
 - segment 665
 - transaction 666
 - transaction environmental 667
 - warning and normal 663
 - contact
 - trading partner 209
 - trading partner details 210
 - continuous receive
 - CICS interface 565
 - invoking 566
 - commands 12
 - MQSeries 516
 - outside Expedite/CICS 520
 - processing received data 518
 - profile layout 255
 - recovering 537
 - reporting status 85
 - return codes 567
 - selection criteria 517
 - sessions 534
 - cleanup 536
 - starting and stopping 534
 - unrecoverable 536
 - special considerations 516
 - starting 95
 - status
 - codes 13
 - reporting 13
 - sent to network 519
 - status reporting
 - format 13
 - stopping 96
 - control blocks 577
 - CCB
 - field descriptions 582
 - layout 581
 - See CCB
 - CMCB
 - field descriptions 622 to 631
 - layout 621
 - common (CCB) 291
 - DATABLK
 - field descriptions 644
 - layout 644
 - FCB
 - field descriptions 584
 - layout 584
 - See FCB.
 - function (FCB) 292
 - NPDB
 - field descriptions 646 to 648
 - layout 645
 - overview 291
 - REQDB
 - field descriptions 650 to 654
 - layout 649
 - service name (SNB) 291
 - SNB
 - field descriptions 579
 - layout 579
 - See SNB.
 - TPPDB
 - field descriptions 634 to 643
 - layout 632
 - TRCB
 - field descriptions 590 to 613
 - layout 586
 - TRIDB
 - field descriptions 617
 - layout 616
 - TRODB
 - field descriptions 619
 - layout 618
 - update status (USDB) 435
 - control information
 - DataInterchange Utility 507
 - format 507
 - control information, DataInterchange Utility
 - field descriptions 509 to 515
 - control numbers 209
 - CTLFILE 187
 - custom programs
 - API 302 to 303
- D**
- D records
 - layout 268, 269

- data
 - concatenation 405
 - condition codes
 - translation, invalid data 671
 - return codes
 - API invalid data 688
 - sending
 - non-EDI 94
- data elements
 - composite
 - notes record 216
 - condition codes
 - translation 664
 - detail
 - record 215
 - header
 - record 214
 - return codes
 - API 681
- data extract exit 482
- data extraction
 - API parameters 402
 - parameters 402
 - services overview 402
 - SNB initialization 402
 - TRCB fields returned 404
 - transaction detail 405
 - TRCB initialization 403
 - TRIDB initialization 403
 - TRODB fields returned, transaction detail 406
 - TRODB initialization 403
- data extracts
 - common key
 - transaction store 281
 - network activity 278
 - trading partner capability 277
 - trading partner profiles 276
 - transaction activity 279
 - transaction store categories 280
- data formats
 - associated objects 217
 - export/import 217
 - export/import file sample 217
 - records
 - dictionary 218
 - field 221
 - format header 219
 - format record ID 219
 - header details 221
 - loop 220
 - loop details 222
 - record 220
 - record details 222
 - structure 221
 - structure details 223
- data management 6
- data management commands 6
- data modes
 - multiple unit of work 316
 - raw data 316
 - translate-to-standard 315
- data transformation
 - map rule 234
- DATABLK
 - field descriptions 644
 - layout 644
- DataInterchange Utility 15
 - CICS abend return codes 487
 - commands overview 17
 - determining results 487
 - invoking 486
 - keywords overview 17
 - overview 1
 - passing control information 486
 - record formats 257
 - terminating 504
- DATATYP,
 - special considerations 414
- DATE keyword 19
- DB2 294
 - attachment 294
 - command file 174
 - deadlock processing 437
 - recovery 437
 - sample JCL 706
 - setup considerations 497
 - thread pools 498
 - timeout processing 437
 - translation tables 308
- ddname, JCL samples 702
- DEENVELOPE AND TRANSLATE command 26
 - examples 27
- DEENVELOPE command 24
 - examples 25
- deenveloping
 - API 384
 - FCB initialization 386
 - parameters 385
 - SNB initialization 385
 - special considerations 339
 - trading partner profiles 397
 - transactions 66

- translate-to-application 338, 339
 - TRCB initialization 386
 - TRIDB initialization 389
 - TRODB initialization 389
- default
 - message name 417
 - message user class 416
- defining
 - EDI1 TD queue 519
 - EDI2 and EDI2 TD queues 529
- DELETE PROFILE command 28
 - examples 28
- deleting profiles 12
- destination files
 - JCL samples 697
- details records
 - data format header 221
 - data format loop 222
 - data format record 222
 - data format structure 223
 - EDI standard data element 215
 - EDI standard segment 214
 - EDI standard transaction 213
 - global variable 231
 - trading partner contact 210
 - XML DTD 243
- diagrams
 - HOT-DI
 - inbound processing 503
 - initializing 501
 - outbound processing 506
 - load module 293
 - receive, deenvelope, and translate 340
 - translation, envelope and send 314
- dictionary
 - data format 218
 - record 212
 - XML data 242
- DIERRFILTER variable 21
- DLM keyword 15
- DTDs
 - details record 243
 - headers record 242
 - resolution 573
- E**
- E record layout 274
- EDI standards
 - associated objects 211
 - composite data element header record 216
 - data element detail record 215
 - data element header record 214
 - dictionary record 212
 - envelopes 365
 - export/import 211
 - export/import file sample 212
 - JCL samples 701
 - segment detail record 214
 - segment header record 214
 - segment notes record 215
 - transaction detail record 213
 - transaction header record 213
 - transaction notes record 216
- EDI1 TD queue, defining 519
- EDI2 and EDI3 TD queues, defining 529
- EDICRIN 567
- EDIEIADF 190
- EDIEICST 190
- EDIEIDDF 190
- EDIEIPRF 190
- EDIEISTD 190
- EDIEITBL 190
- EDIEITPT 190
- EDIFACT profile layout 249
- EDIQUERY 180
- EDISYSIN 173
- EDITSIN 174, 294
- EDIVAX 182
- EDIW transaction 544
- element records, map 228
- encoding, XML considerations 574
- encryption exit 464
 - languages
 - Assembler 468
 - C 467
 - COBOL 466
 - parameters 464
- end translation/enveloping API 384
 - parameters 384
- envelope
 - API 368, 369
 - CICS queue alternatives 489
 - EDI standards 365
 - file 176
 - file overrides 177
 - key 432
- ENVELOPE AND SEND command 31
 - examples 32
- ENVELOPE command 29
 - examples 30
- envelope data

- reporting 35
- ENVELOPE DATA EXTRACT command 33
 - examples 34
- envelope transaction 373
- enveloping
 - API 368, 369
 - automatic during translation 312
 - commands 368
 - data file
 - JCL samples 699
 - FCB initialization 370
 - initializing 370
 - interchange status fields 326
 - overview 311
 - parameters 369
 - SNB initialization 370
 - sorting transactions 368
 - special considerations 379
 - trading partner profiles 396
 - transactions 373
 - TRCB initialization 370
 - TRIDB initialization 372
 - TRODB initialization 372
- enveloping options file 182
- enveloping services
 - functions 368
 - overview 365
 - parameters 369
- environment
 - persistent
 - enabling/disabling 527
 - multiple MVS subtasks 526
 - multiple regions 527
 - MVS space requirements 526
- environment, persistent 526
- environmental condition codes
 - translation 672
- environmental services
 - initializing 304
 - parameters 305
 - request results 305
 - overview 304
 - terminating parameters 306
 - terminating request results 307
 - utility service API parameters 306
- error codes
 - translator 614
 - field-level 614
 - group-level 615
 - interchange-level 615
 - segment-level 614
 - transaction-level 614
 - warnings 614
- errors
 - filtering 21
 - TRXABORT field 356
 - TRXACCEPT field 356
- event logs
 - archiving entries 6
 - removing entries 6
- events logging report 50
- exception file 177
- exit routines 443
 - authentication
 - Assembler definition 472
 - C definition 472
 - COBOL definition 471
 - parameters 469
- call 481
- compression
 - Assembler definition 478
 - C definition 477
 - COBOL definition 476
 - parameters 474
- data extract 482
- encryption
 - Assembler definition 468
 - C definition 467
 - COBOL definition 466
 - parameters 464
- field 444, 446
 - Assembler definition 450
 - C definition 450
 - COBOL definition 451
 - receive parameters 448
 - send parameters 447
- filtering parameters 475
- get data
 - parameters 479
- get/put envelope 457
 - parameters 458
- languages 444
- linkage editor 445
- message handler 559
- performance monitor 541
- post-translation 444, 453
- pre-translation 444, 453
- put data
 - parameters 480
- return codes 444
- security 444, 460
 - authentication 469

- compression 474
 - enabling during receive 462
 - enabling during send 461
 - encryption 464
 - parameters 462
 - support 479
 - transaction 453
 - Assembler definition 456
 - C definition 456
 - COBOL definition 455
 - translation parameters 453
 - Expedite/CICS
 - continuous receives 520
 - interface 531
 - export
 - associated objects 186
 - EXPORT command 36
 - examples 36
 - export/import
 - B1 field descriptions 238
 - B2 field descriptions 240
 - common control record 191
 - common end of group record 192
 - data formats 217
 - EDI standards 211
 - file sample 225
 - data format 217
 - EDI standards 212
 - table definitions 237
 - XML definitions 241
 - JCL samples 703, 704
 - maps 224
 - profiles layouts 244
 - queues 528
 - record sequence 193
 - table definition 237
 - table definitions 237
 - sample 237
 - table entry definition 240
 - trading partner profile header record 194
 - trading partner profile record 194
 - XML definitions 241
 - exporting
 - administrative data 12
 - commands 12
 - control file 187
 - field descriptions 187
 - data categories 186
 - files 190
 - data area 192
 - record sequence 190
 - maps 12
 - profiles 193
 - trading partner information 12
 - utility overview 186
- F**
- FAENV 182
 - example 184
 - field descriptions 183
 - file format 184
 - FCB
 - initialization
 - deenvolving 386
 - envolving 370
 - translate-file-to-application 351
 - translate-to-application 342
 - translate-to-standard 318
 - layout 584
 - overview
 - FFSEXCP 177
 - FFSTRAK 178
 - JCL samples 698
 - FFSWORK 181
 - JCL samples 698
 - field descriptions
 - control information
 - DataInterchange Utility 509 to 515
 - field exit 446
 - languages
 - Assembler 450
 - C 450
 - COBOL 451
 - logical name 444
 - receive parameters 448
 - send parameters 447
 - fields
 - data format 221
 - starting groups 381
 - starting interchanges 380
 - FILENAME, special considerations 414
 - files 177
 - application 175
 - CMCB fields returned, send request 419
 - command 173, 174
 - commands JCL sample 705
 - data area 192
 - destination JCL sample 697
 - EDISYSIN 173
 - EDITSIN 174, 294
 - envelope 176

- data JCL sample 699
- envelope options 182
- export 187
- export/import
 - data format sample 217
 - EDI standard sample 212
 - JCL sample 703
 - XML definition sample 241
- export/import JCL sample 704
- FFSTRAK JCL sample 698
- FFSWORK JCL sample 698
- formats 173
- functional acknowledgments 182
- import 187
- input 173
 - C and D 175
 - raw data 175
- maintenance techniques 717
- MVS and IEBASE JCL sample 699
- MVS and IEBASE/Comm-Press JCL sample 700
- network command (NETOP) 175
- output 173
 - EDIQUERY 180
- output JCL sample 703
- pageable translation 182
 - JCL sample 706
- print 179
- PRTFILE JCL sample 697
- query 180
- receiving 73, 74
- report 180
 - JCL sample 702
- sending 74
- SYSIN 173
- tracking 178
- translation to application 350
- work 181
- filtering errors 21
- filtering exit parameters 475
- first call for transaction 321, 344, 354
 - TRCB returned fields 345
- first call of session 318, 342, 351
- first calls, translation 322
 - overrides 323
- forcing interchange termination 335
- format
 - command language 18
 - header data format 219
- formulas, required space for pageable translation 338
- FSUPPORT member 558

- function control block (FCB)
 - field descriptions 584
- functional acknowledgments
 - commands 182
 - file 182
 - JCL samples 704
 - override JCL 704
 - retrieve functional acknowledgment image
 - API 406
 - TRCB fields returned 358, 391

G

- G records
 - layout 275
- generalized networks 546
- get data exit 479
 - parameters 479
- get envelope
 - API parameters 441
 - call 457
 - services overview 441
- get/put envelope
 - exit parameters 458
 - exit routine 457
 - program 483
- get/put envelope service
 - parameters 458
- GLB commands 14
- GLB DUMP command 37
 - examples 37
- GLB TRACE command 38
 - examples 38
- groups
 - condition codes, translation 669
 - data extract record layouts 283
 - fields that start 381
 - headers, retrieve group header API 400
 - layer 367
 - return codes, API 686
 - status, header and footer 329
 - TRCB fields returned 348, 361, 376
 - status 394

H

- HANDLE keyword 19
- headers
 - details record data format 221
 - map record 226
 - XML DTD 242

- HOLD command 39
 - examples 40
- HOT-DI
 - inbound processing diagram 503
 - initializing 306, 500
 - diagram 501
 - multiple tasks 501
 - outbound processing 505
 - diagram 506
 - processing considerations 502
 - running MVS-CICS 500
 - terminating 505
- I records
 - layout 273
- ICS profile layout 251
- import
 - associated objects 186
- IMPORT command 41
 - examples 41
- importing
 - administrative data 12
 - commands 12
 - control file 187
 - field descriptions 187
 - data categories 186
 - files 190
 - data area 192
 - record sequence 190
 - maps 12
 - profiles 193
 - trading partner information 12
 - utility overview 186
- inbound processing 5
 - HOT-DI diagram 503
 - SAP status 568
- incremental translation, internal 337
- independent programs 482
- Information Exchange interface 531
- Information Exchange sessions 531
 - cleanup 533
- initialize SYNC 438
 - API parameters 438
- initialize SYNC request
 - CCB fields returned 438
- initializing
 - CMCB
 - cancel 424
 - process network acknowledgments 428
 - receive 421
 - return filename 426
 - send files 418
 - send transactions 413
 - DataInterchange 486
 - DataInterchange in CICS 501
 - enveloping 370
 - environmental services 304
 - parameters 305
 - return codes 305
 - HOT-DI 500
 - diagram 501
 - multiple tasks 501
 - HOT-DI utility service 306
 - return codes 678
 - SYNC 693
 - SYNC functions 438
- input
 - files 173
 - records 175
- interchange ID keys
 - alternate 434
 - full 432
- interchanges
 - alternate key creation 430, 433
 - condition codes, translation 670
 - data extract record layout 282
 - enveloping
 - TRCB status 326
 - fields that start 380
 - headers
 - retrieve interchange header API 399
 - key creation 429
 - layer 367
 - return codes, API 687
 - status, header and footer 328
 - termination, forced 335
 - TRCB fields returned 359, 375, 378, 383
 - status 393
- interfaces
 - Expedite/CICS 531
 - Information Exchange 531
 - MQSeries 570
 - network 546
 - other applications and networks 545
 - SAP 568
- internal
 - calls 427
 - translation, outbound 337
- invoking DataInterchange Utility using EDIW 544
- ISPF functions 190

issue commit API 399
issue commit parameters 399

J

JCL
 samples 695
JCL DISP option 175
JCL parameter example 16
JCL samples 695, 696, 697, 706, 707

K

keys
 account number and user ID
 alternate keys 434
 full key 433
 full envelope 432
 interchange 429
 alternate 430, 433
 interchange ID and qualifier
 alternate key 434
 full key 432
 trading partner nickname 432
 transaction ID 433
 transaction store, common data extract 281
keywords
 APPLID, MVS environment 15
 case sensitive 173
 DATE 19
 DLM 15
 HANDLE 19
 LANGID 15
 MQEXCP 16
 MQPRT 15
 MQQUERY 16
 MQRPT 15
 MQSYSIN 15
 MQTRAK 16
 overview 17
 PLAN 16
 RECOVERY 495
 SYSID 15
 SYSTEM 16
 TIME 19
 USERPGM 10

L

LANGID keyword 15
language profile layout 249

languages
 exit 444
last call for transaction 327, 350, 363
last call of session 331, 350, 363, 379, 395
layers
 group 367
 interchange 367
 transaction 367
libraries 577
link edit requirements 292
linkage editor 445
literals
 &BOUNDARY 336
load libraries 577
 members 577
LOAD LOG ENTRIES command 42
 example 42
load modules diagram 293
locating trading partner profiles 396, 397
logs, activity layout 247
loops
 data format 220
 details record data format 222

M

mailbox (requestor) profile block (REQDB) 649
 field descriptions 650 to 654
mailbox profiles 244
maintenance
 file 717
 profiles 12
management reports
 categories 7
 creating 9, 10, 11
 data extract layouts 276
managing data 6
maps
 associated objects 224
 export/import 224
 exporting 12
 importing 12
 receive 5
 records
 application control 229
 commands 231
 data transformation map rule 234
 element 228
 global variables 231
 header 226
 local variables 230

- nodes 230
- receive usage 233
- reference 230
- segment 227
- send usage 232
- syntax 229
- members
 - FSUPPORT 558
 - load libraries 577
- message handler 559
 - control information 564
 - error occurred response fields 565
 - successful condition response fields 565
- messages
 - default name 417
 - default user class 416
- modes
 - production 317
 - switching 317
 - test 317
- MQEXCP keyword 16
- MQPRT keyword 15
- MQQUERY keyword 16
- MQRPT keyword 15
- MQSeries
 - continuous receive 516
 - interface overview 570
 - queues, profile layout 256
- MQSYSIN keyword 15
- MQTRAK keyword 16
- multiple unit of work 316
- MVS
 - communications
 - IEBASE and Comm-Press JCL sample 700
 - IEBASE JCL sample 699
 - condition codes 674
 - DataInterchange Utility 15
 - DB2 attachment 294
 - special considerations 714

N

NETOP 175

network

- acknowledgments API parameters 427
- activity data extract record 278
- activity report 44
- application to network program 408
- CICS program control 561
- commands profile 246, 555
 - example 557

- field descriptions 555
- error occurred response fields 563
- no data received response fields 563
- profile 246
- security profile 245
- send network function 415
- successful condition response fields 563
- NETWORK ACTIVITY DATA EXTRACT
 - command 43
 - examples 43
- network commands file 175
 - building. 554
- network interfaces
 - generalized 546
 - other applications and networks 545
 - point-to-point 546, 549
 - activating connections 549
 - supported 546
- network profile block (NPDB) 645
 - field descriptions 646 to 648
- network profiles
 - CICS definition 560
- nodes record 230
- non-EDI data, sending 94

O

- optimizing performance 713
- optional records 177, 271
- options, JCL DISP 175
- outbound processing
 - HOT-DI diagram 506
 - SAP status 568
- outbound processing, HOT-DI 505
- output
 - CMCB fields 411
 - files 173
 - envelope 176
- output file, JCL samples 703
- overrides, envelope file 177
- overriding condition codes 22

P

- pageable translation 182, 337
 - file 182
 - file size formula 182
 - JCL samples 706
 - required space formula 338
- parameters
 - communication 554

- sample DB2 JCL 706
- sample JCL 696
- sample XML JCL 707
- partial structures 334, 363
- perform commands file
 - JCL samples 705
- performance monitor user exit 541
 - commarea format 542
- performance, special considerations 713
- persistent environment 14, 526
 - commands 14
 - enabling/disabling 527
 - multiple MVS subtasks 526
 - multiple regions 527
 - sizing MVS space 526
- PL/I calls 296
 - samples 297
- PLAN keyword 16
- point-to-point networks 546, 549
 - activating connections 549
- post-translation exit, logical name 444
- pre-translation exit, logical name 444
- PRINT ACKNOWLEDGMENT IMAGE
 - command 45
 - examples 46
- PRINT ACTIVITY SUMMARY command 47
 - examples 48
- print activity summary report 48
- print commands 50, 52, 54, 56
- PRINT EVENT LOG command 49
 - examples 50
- print events logging report 50
- print file 179
 - sample 179
- PRINT STATUS SUMMARY command 51
 - examples 52
- PRINT STATUS SUMMARY2 command 53
 - examples 54
- PRINT TRANSACTION DETAILS command 55
 - examples 56
- PRINT TRANSACTION IMAGE command 57
 - examples 58
- printing
 - commands 45, 47, 49, 51, 53, 55, 57, 59
- printing commands 8
- process network acknowledgments API
 - CMCB initialization 428
 - parameters 427
- PROCESS NETWORK ACKS command 59
 - examples 59
- processing
 - continuous receive 518
 - flow, application to network 551
 - HOT-DI considerations 502
 - inbound 5
 - multiple TS queues 489
 - serial 490
 - TS queues, additional processing 527
- production, switching modes 317
- profile maintenance
 - commands 12
- profiles
 - activity log
 - layout 247
 - application defaults
 - layout 247
 - CICS performance
 - layout 256
 - continuous receive
 - layout 255
 - deleting 12
 - EDIFACT envelope
 - layout 249
 - ICS envelope
 - layout 251
 - language
 - layout 249
 - layouts
 - export/import 244
 - mailbox (requestor)
 - layout 244
 - MQSeries queue
 - layout 256
 - network
 - layout 246
 - network commands 555
 - example 557
 - layout 246
 - network security
 - layout 245
 - querying 12, 64
 - requestor
 - See mailbox profiles
 - trading partner data block 409
 - trading partners
 - data extract 276
 - export/import profiles 193
 - field descriptions 197 to 209
 - file example 197
 - format 195
 - UCS envelope
 - layout 253

- UN/TDI envelope
 - layout 252
- user exit
 - layout 248
- X12 envelope
 - layout 254
- programming languages
 - API-supported 292
 - Assembler 299
 - C 298
 - COBOL 295
 - PL/I calls 296
- programs
 - condition codes 672
 - data exit 482
 - data extract samples 579
 - get/put envelope 483
 - independent 482
 - list table considerations 538
 - processing table considerations 539
 - samples included 186
- PRTFILE 179
 - JCL samples 697
 - sample 179
- PURGE command 60
 - examples 61
- purging transactions 60
- put data exit
 - parameters 480
- put envelope
 - API parameters 442
 - call 457
 - services overview 441

Q

- Q record layout 275
- queries
 - profiles 12
 - transaction store 717
- QUERY command 62
 - examples 63
- query file 180
- QUERY PROFILE command 64
 - examples 64
- querying
 - profiles 64
 - transactions 62
- queue standard data API 427
 - parameters 427

R

- raw data 316
 - input file 175
 - record layout 271
 - sending 331
- receive
 - CMCB initialization 421
 - maps 5
 - parameters 421
 - returned information 422
 - transactions API 420
 - usage map record 233
- RECEIVE AND DEENVELOPE command 66
 - examples 67
- RECEIVE AND SEND command 68
 - examples 68
- RECEIVE AND TRANSLATE command 69
 - examples 70
- RECEIVE command 65
 - combination commands in CICS 5
 - examples 65
- receive, deenvelope and translate diagram 340
- receiving
 - and deenveloping, translate-to-application 338
 - files 73, 74
 - restart command 87
 - special considerations 339
 - trading partner profiles, locating 396, 397
 - transactions 65, 66, 68, 69
 - translate-to-application 338
 - special considerations 339
- RECONSTRUCT AND SEND command 72
 - examples 72
- RECONSTRUCT command 71
 - examples 71
- reconstructing transactions 71, 72
- record format ID
 - data format 219
- records
 - 000 end of group 192
 - 0C1/0C2 191
 - 1Y1 212
 - 1Y2 213
 - 1Y3 213
 - 1Y4 214
 - 1Y5 214
 - 1Y6 214
 - 1Y7 215
 - 1Y8 215
 - 1Y9 216
 - 1YA 216

- 2W1 218
- 2W2 219
- 2W3 219
- 2W4 220
- 2W5 220
- 2W6 221
- 2W7 221
- 2W8 221
- 2W9 222
- 2WA 222
- 2WB 223
- 3T5 232
- 3T6 233
- 3T7 234
- 3V1 226
- 3V2 227
- 3V3 228
- 3V9 229
- 3VA 229
- 3VB 230
- 3VC 230
- 3VD 230
- 3VE 231
- 7A1 210
- 7P1 194
- 7P2 194
- 7P3 209
- 7P4 209
- 7Z1 210
- acknowledgment image data extract layout 288
- AJ1 242
- AJ2 242
- AJ3 243
- application data extract layout 287
- BK2 231
- C (control) 257
 - field descriptions 259 to 268
 - layout 257
- D (data) 257
- D (data) record layout 269
- data
 - layout 268
 - multiple structures 269
 - single structure 269
 - size constraints 268
- data format 220
- details record
 - data format 222
- E (interchange header) layout 274
- end transaction and interchange (Z) layout 270
- export/import common control 191
- export/import common end of group 192
- G (group header) layout 275
- group data extract layout 283
- header, trading partner profile 194
- I (information) layout 273
- interchange data extract layout 282
- layouts 173
- mapping 218, 219, 220, 221, 222, 223, 226, 227, 228, 229, 230, 231, 232, 233, 234, 242, 243
- optional 177, 257, 271
 - commands 272
- Q (queuing totals) layout 275
- raw data 257
 - layout 271
- SAP status, extracting 569
- SAP status, removing 569
- T (transaction set header) layout 275
- trading partner
 - comments definition 210
 - contact 209
 - contact details 210
 - control numbers 209
- trading partner profile 194
- transaction data extract layout 284
- transaction image data extract layout 288
- transaction store
 - data extract 282
- utility 257
- Z (end transaction and interchange) 257, 270
- Z layout 270
- recovery
 - continuous receives 537
 - DB2 conditions 437
 - RECOVERY keyword considerations 495
 - scope 334
 - unit of work 491, 495
 - application 496
 - DataInterchange utility 493
- RECVFILE AND SEND command 74
 - examples 74
- RECVFILE command 73
 - examples 73
- REENVELOPE AND SEND command 77
 - example 78
- REENVELOPE command 75
 - examples 76
- reenveloping transactions 77
- RELEASE command 79
 - examples 80
- releasing transactions 79
- REMOVE LOG ENTRIES command 81

- examples 81
- REMOVE STATISTICS command 82
 - examples 82
- REMOVE TRANSACTIONS command 83
 - examples 84
- removing
 - event log entries 6
 - statistics 82
 - transactions 60, 81, 83
- REPORT CONTINUOUS RECEIVE STATUS
 - command 85
 - examples 85
- report file 180
 - JCL samples 702
- reporting
 - continuous receive status 13
 - continuous receive status format 13
 - statistics 6
- reports
 - customized 35
 - envelope data 35
 - events logging 50
 - management reports 9, 10, 11
 - network activity 44
 - print activity summary 48
 - print events logging 50
 - selection criteria for status summary 54
 - status summary 52
 - status summary2 54
 - transaction data 10
 - transaction details 56
 - transaction image 58
 - transaction store 9, 11
- required space
 - pageable translation formula 338
- RESET STATISTICS command 86
 - examples 86
- response applications 521
 - continuous receive 522
 - DataInterchange Utility 521
 - transaction 523
 - specifying 525
 - types 521
- RESTART RECEIVE command 87
 - examples 87
- restart receive parameters 421
- restart receive transactions API 420
- RESTART SEND command 88
 - examples 88
- restart send transactions
 - parameters 413
- restart send transactions API 412
- restarting receives
 - RESTART RECEIVE command 87
- restarting sends 88
- RETRANSLATE TO APPLICATION command 89
 - examples 90
- retrieve detailed data 404
- retrieve functional acknowledgment image API 406
- retrieve group header
 - API 400
 - parameters 400
- retrieve interchange header
 - API 399
 - parameters 399
- retrieve transaction acknowledgment image API 406
- retrieve transaction header
 - API 400
 - parameters 401
- retrieve transaction header API 400
- retrieve transaction image API 406
- return
 - filename request
 - CMCB fields returned 427
- return codes 655
 - API 678
 - API commit 694
 - API communication 691
 - API communication nonsevere 692
 - API communication severe 693
 - API communication warnings 692
 - API data element 681
 - API group 686
 - API initializing 678
 - API interchange 687
 - API invalid data 688
 - API negative codes 679
 - API normal 680
 - API rollback 694
 - API segment 682
 - API status update 693
 - API SYNC initializing 693
 - API termination 679
 - API transaction environmental 684, 689
 - API transaction program 689
 - API transaction syntax 683
 - API translation 680
 - CICS abends 295
 - communication 411
 - DataInterchange 656
 - CICS considerations 504
 - exit routines 444

- TRANSFORM command 659
 - warning
 - API 680
- return filename API 426
 - parameters 426
- return filename request
 - CMCB initialization 426
 - returned information 427
- rollback
 - rollback work request
 - CCB fields returned 440
- rollback work API
 - parameters 439
 - return codes 694
- RPTFILE 180

S

- samples
 - command file JCL 705
 - communications with IEbase and Comm-Press JCL 700
 - communications with IEbase JCL 699
 - data extract programs 579
 - DB2 parameter JCL 706
 - ddname JCL 702
 - destination files JCL 697
 - EDI standard JCL 701
 - envelope data file JCL 699
 - export/import JCL 703, 704
 - export/import JCL statements 704
 - FFSTRACK JCL 698
 - FFSWORK JCL 698
 - functional acknowledgments JCL 704
 - JCL 695
 - JCL modifications 695
 - JCL parameters for MVS 16
 - JCL RPTFILE 697
 - output JCL 703
 - pageable translation JCL 706
 - report file JCL 702
 - STEPLIB JCL 697
 - translate-to-standard JCL 697
 - Utility parameter JCL 696
 - XML parameter JCL 707
- SAP
 - extracting status records 569
 - inbound processing status 568
 - interface 568
 - outbound processing status 568
 - removing status records 569
 - status codes 569
- SAP STATUS EXTRACT command 91
 - examples 91
- SAP STATUS REMOVE command 92
 - examples 92
- scope, recovery 334
- security exit 460
 - authentication 469
 - compression 474
 - enabling during receive 462
 - enabling during send 461
 - encryption 464
 - logical name 444
 - parameters 462
 - support 479
- security support exits 479
- segments
 - condition codes, translation 665
 - detail record 214
 - header record 214
 - notes record 215
 - records, maps 227
 - return codes, API 682
- SELECTING clauses 18
- selection criteria
 - continuous receive 517
- selection criteria for status summary report 54
- send
 - map files
 - export/import
 - send map sample 225
- SEND command 93
 - examples 93
 - fixed-to-fixed translation 4
- send files
 - API 417
 - CMCB initialization 418
 - parameters 417
 - returned information 419
- send maps
 - files
 - export/import sample 225
- send network function 415
- send transactions
 - API 412
 - CMCB initialization 413
 - parameters 413
 - returned information 415
- send usages
 - map record 232
- SENDFILE command 94

- examples 94
- sending
 - files 74
 - non-EDI data 94
 - partial structures 334
 - raw data, required values 331
 - recovery scope for raw data 334
 - restart command 88
 - trading partner profiles
 - locating 396, 397
 - transactions 68, 72, 77, 380
 - deferred 380
 - immediately 380
 - translation functions 311
- sent to network status 519
- serial processing 490
- service name block (SNB)
 - field descriptions 579
- service segments, TRCB fields returned 392
- services
 - CCB fields returned
 - call exit 481
 - get data 479
 - get envelope 459
 - put data 480
 - put envelope 459
 - communication
 - functions 409
 - overview 407
 - return codes 411
 - data extraction overview 402
 - enveloping overview 365
 - environmental overview 304
 - get envelope overview 441
 - get/put envelope parameters 458
 - negative return codes 679
 - put envelope overview 441
 - SYNCPOINT overview 436
 - translation overview 308
 - update status overview 429
 - utility in CICS 504
- sessions
 - continuous receive 534
 - cleanup 536
 - identifying unrecoverable sessions 536
 - starting and stopping 534
 - first call 318, 342, 351
 - Information Exchange 531
 - cleanup 533
 - last call 331, 350, 363, 379, 395
- setup
 - DB2 in CICS 497
 - thread pools 498
- SNB
 - initialization
 - data extraction 402
 - deenveloping 385
 - enveloping 370
 - translate-file-to-application 351
 - translate-to-application 342
 - translate-to-standard 318
 - overview 579
- special considerations
 - CICS 715
 - COBOL 720
 - continuous receive 516
 - DATATYPE 414
 - deenveloping 339
 - trading partner profiles 397
 - EDICRIN 567
 - enveloping 379
 - trading partner profiles 396
 - FILENAME 414
 - MVS 714
 - performance 713
 - processing program table 539
 - program list table 538
 - receiving 339
 - translate-to-application, receiving and
 - deenveloping 339
 - translate-to-standard 331
 - translation 316
 - inbound 662
 - outbound 661
 - translate-to-application 363
 - XML 573
- START CONTINUOUS RECEIVE command 95
 - examples 95
- starting continuous receives 95, 534
- startup, CICS 499
- statements, export/import JCL sample 704
- statistics
 - removing 82
 - reporting 6
 - resetting cumulative records 86
- status
 - continuous receive 13, 85
 - sent to network 519
- status codes
 - SAP 569
- status summary report 52
- status update

- return codes 693
- STEPLIB requirements 697
- STOP CONTINUOUS RECEIVE command 96
 - examples 96
- stopping continuous receives 96, 534
- storage
 - CICS mechanisms 487
- structures
 - data format 221
 - details record
 - data format 223
 - partial 334, 363
- stub programs 292
 - get data exit 479
 - get/put envelope exit 458
 - get/put envelope service 458
 - put data exit 480
- SYNC, initialization return codes 693
- SYNCPOINT services
 - overview 436
- syntax
 - command language 17
 - conventions xv
 - errors, API transaction 683
 - map records 229
- SYSID keyword 15
- SYSIN 173
- SYSTEM keyword 16

T

- T records
 - layout 275
- tables
 - B1 definition 237
 - B1 field descriptions 238
 - DB2 translation 308
 - definitions
 - export/import file sample 237
 - definitions, export/import 237
 - entries
 - B2 definition 240
 - B2 field descriptions 240
- TD queues
 - defining EDI1 519
 - defining EDI2 and EDI3 529
 - export/import 528
 - reserved 527
- terminal-attached applications 496
- terminating
 - DataInterchange 504
 - environmental services 306
 - parameters 306
 - return codes 307
 - HOT-DI 505
 - interchanges 335
- termination return codes 679
- testing
 - switching modes 317
 - translate-to-application API 341
 - translate-to-standard API 315
- thread pools 498
- TIME keyword 19
- timeout/deadlock processing, DB2 437
- TPPDB for communications 409
- tracking file 178
- trading partner 209
 - exporting information 12
 - importing information 12
- TRADING PARTNER CAPABILITY DATA
 - EXTRACT command 97
 - examples 98
- trading partner profile block (TPPDB) 632
 - field descriptions 634 to 643
- TRADING PARTNER PROFILE DATA EXTRACT
 - command 99
 - examples 100
- trading partners
 - capability data extract record 277
 - comments definition 210
 - contact definition 209
 - contact details definition 210
 - control numbers 209
 - nickname key 432
 - profile block (TPPDB) and communication 409
 - profiles
 - data extract record 276
 - export/import 193
 - file sample 197
 - field descriptions 197 to 209
 - format 195
 - locating members for receiving 396, 397
 - locating members for sending 396, 397
 - reporting 7
 - retrieving usages 398
- transaction
 - headers
 - retrieve transaction header API 400
- TRANSACTION ACTIVITY DATA EXTRACT
 - command 101
 - examples 101
- TRANSACTION DATA EXTRACT command 103

- examples 104
- transaction details report 56
- transaction exit languages
 - Assembler 456
 - C 456
 - COBOL 455
- transaction ID
 - key 433
 - retrieving usages 398
- transaction image
 - data extract record layouts 288
 - report 58
- transaction store
 - common key 281
 - data extracts
 - information categories 280
 - record layouts 282
 - query techniques 717
- transactions
 - activity data extract record 279
 - bundles 335, 364, 379
 - clusters 335, 364, 379
 - CMCB fields returned
 - receive request 423
 - send request 415
 - condition codes
 - translation 666
 - translation environmental 667
 - data extract record layouts 284
 - deenveloping 66
 - detail
 - record 213
 - report 56
 - enveloping 373
 - sorting 368
 - exit routines 453
 - post-translation 453
 - pre-translation 453
 - first call 321, 344, 354
 - returned TRCB fields 345
 - TRCB status 325
 - header record 213
 - headers
 - TRCB fields returned 349, 362
 - headers, retrieve transaction header API 400
 - image report 58
 - last calls 327, 350, 363
 - layer 367
 - managing 6
 - notes record 216
 - purging 60
 - querying 62
 - receiving 65, 66, 68, 69
 - reconstructing 71, 72
 - reenveloping 77
 - releasing 79
 - removing 81, 83
 - reporting categories 8
 - retrieve detailed data API 404
 - retrieve transaction acknowledgment image
 - API 406
 - retrieve transaction image API 406
 - return codes
 - API environmental 684, 689
 - API program 689
 - API syntax 683
 - sending 68, 72, 77
 - sorting for enveloping 368
 - status
 - header and footer 330
 - raw data 333
 - subsequent calls 327, 349, 363
 - supplied by DataInterchange 539
 - translating 69
 - TRCB fields returned 325, 345, 347, 356, 377, 390
 - envelopes 373
 - header/trailer 395
 - status 394
- TRANSFORM command 105
 - examples 105
 - return codes 659
- TRANSLATE AND ENVELOPE command 106
 - examples 107
- TRANSLATE AND SEND command 108
 - API business tasks 302
 - examples 109
- translate specific
 - API 341
 - parameters 342
- TRANSLATE TO APPLICATION command 110
 - examples 111
- TRANSLATE TO STANDARD command 112
 - examples 112
- translate-file-to-application
 - API 350
 - FCB initialization 351
 - parameters 350
 - SNB initialization 351
 - TRCB initialization 351
 - TRIDB initialization 354
 - TRODB initialization 354, 355

- translate-to-application
 - API functions 341
 - API test 341
 - FCB initialization 342
 - receiving and deenveloping 338
 - SNB initialization 342
 - special considerations 363
 - TRCB initialization 343
 - TRIDB initialization 343
 - TRODB initialization 344
 - translate-to-application API 338
 - translate-to-standard
 - API 311
 - function codes 317
 - test 315
 - data modes 315
 - FCB initialization 318
 - JCL samples 697
 - parameters 317
 - SNB initialization 318
 - special considerations 331
 - TRCB initialization 318
 - TRIDB initialization 324
 - TRODB initialization 321
 - translation
 - automatic enveloping 312
 - error codes 614
 - field-level 614
 - group-level 615
 - interchange-level 615
 - segment-level 614
 - transaction-level 614
 - warnings 614
 - files
 - output 177
 - internal incremental 337
 - pageable 182, 337
 - JCL sample 706
 - partial structures 363
 - return codes 680
 - special considerations 316, 661, 662
 - transactions 69
 - translation status fields 328
- translation exit
- parameters 453
- translation services
- DB2 tables 308
 - first call 322
 - overrides 323
 - functions 310
 - overview 308
 - translate-to-application API 338
 - translation specific parameters 342
 - translation-file-to-application parameters 350
 - translation-to-standard parameters 317
- translator control block (TRCB) 586
- field descriptions 590 to 613
- translator input data block (TRIDB) 616
- field descriptions 617
- translator output data block (TRODB) 618
- field descriptions 619
- TRCB fields
- group status, header and footer 329
 - interchange status, enveloping 326
 - interchange status, header and footer 328
 - transaction header and footer 330
 - transaction status
 - raw data 333
 - transaction status, first call 325
 - translation status 328
- TRCB initialization
- data extraction 403
 - deenveloping 386
 - enveloping 370
 - translate-file-to-application 351
 - translate-to-application 343
 - translate-to-standard 318
- TRCB returned fields
- application data 346, 357, 391
 - data extraction detail data 405
 - data extraction records 404
 - envelope application data 374
 - functional acknowledgment data 358, 391
 - group data 348, 361
 - group header/trailer data 376
 - group status data 394
 - interchange data 359, 378, 383
 - interchange header/trailer data 375
 - interchange status data 393
 - service segment data 392
 - transaction data 325, 345, 347, 356, 390
 - envelopes 373
 - transaction header data 349, 362
 - transaction header/trailer data 377, 395
 - transaction status data 394
- TRIDB initialization
- data extraction 403
 - deenveloping 389
 - enveloping 372
 - translate-file-to-application 354
 - translate-to-application 343
 - translate-to-standard 324

TRODB initialization
 data extraction 403
 deenvolving 389
 envolving 372
 translate-file-to-application 354, 355
 translate-to-application 344
 translate-to-standard 321
 TRODB returned fields
 application data 347, 358
 data extraction detail data 406
 TRXABORT 356
 TRXACCEPT 356
 TS queues
 additional processing 527
 export/import 528
 multiple queues in CICS 489
 reserved 527

U

UCS profile layout 253
 UN/TDI profile layout 252
 unit of work 491, 493, 495, 496
 multiple 316
 raw data 316
 UNLOAD LOG ENTRIES command 113
 UNPURGE command 114
 examples 115
 UPDATE STATISTICS command 116
 examples 116
 update status
 services overview 429
 update status API 430
 parameters 430
 UPDATE STATUS command 117
 examples 117
 update status data block (USDB) 435
 update status request
 CCB fields returned 431
 returned information 431
 usages
 formula for retrieving 398
 retrieved for trading partner and transaction ID 398
 USDB 435
 user exit profile layout 248
 user ID key
 alternate 434
 user ID keys
 full 433
 USERPGM keyword 10

utility condition codes 22
 utility service API
 CICS 504
 initializing 306
 parameters 306

V

validating command language 19
 VANCICS 560
 variables
 DIERRFILTER 21
 global details record 231
 local record 230

W

warnings
 API communication return codes 692
 translation codes 663
 translator 614
 WHERE clauses 18
 work file 181

X

X12 profile layout 254
 XML
 definitions
 export/import 241
 export/import sample 241
 DTD resolution 573
 encoding considerations 574
 records
 dictionary 242
 DTD details 243
 DTD header 242
 sample JCL 707
 special considerations 573

Z

Z record layout 270
 Z records 270
 ZFCBFUNC field in CICS 562

