Digital DEC/EDI

Application Development

Revised for Software Version: V4.0

Compaq Computer Corporation
Houston, Texas

November 2001

Printed in the U.S.A.

Contents

Preface

Part I Getting Started with Digital DEC/EDI: A Tutorial

Chapter 1 Creating an Outgoing Mapping Table

Chapter 2 Creating an Incoming Mapping Table

Chapter 3 Command Line Interface

Chapter 4 C Language Application Programming Interface (API)

Chapter 12 Importing Digital DEC/EDI Document Data

Chapter 13 Mapping in More Detail

Chapter 14 Mapping Expressions

Chapter 15 Lookup Tables

Chapter 16 Supported Mapping Constructs

Chapter 17 Using Hooks to Customize the Mapper

Preface

Product Name: Digital DEC/EDI Version 4.0

Purpose of This Book

This book describes how to solve problems when supporting Digital DEC/EDI in an OpenVMS environment.

Readership

The book is intended for use by system managers, programmers and other personnel who need to support the Digital DEC/EDI Server in an OpenVMS environment. The book is divided into the following parts:

1. Problem Solving

   This part is intended for anyone who is responsible for solving problems reported on a Digital DEC/EDI system.

2. Error Messages

   This part contains a list of each of the errors which Digital DEC/EDI can log. With each message listed, there is a description of why the message was logged and what recovery action you need to take. The book is intended to be used as reference material in conjunction with the section on Problem Solving.

Related Books

This is one of a set of Digital DEC/EDI books. The complete list is as follows:

- *Digital DEC/EDI: Introduction*

  This book introduces general EDI concepts, and Digital's EDI system, Digital DEC/EDI. It describes the main components of the Digital DEC/EDI system, and how business documents are processed and communicated to trading partners. The book seeks to establish the concepts and terms used by Digital DEC/EDI. These are also summarized in a glossary.

You are strongly recommended to become familiar with the material in thisbook before proceeding to install or use Digital DEC/EDI.

- *Digital DEC/EDI: Installation*

  This book describes how to install the Digital DEC/EDI software, how to perform basic system configuration and how to verify such an installation. It describes how to install the Application Client, Server, Cockpit and CommandCenter components.

- *Digital DEC/EDI: Application Development*

  This book describes the Application Client interfaces and the means of connecting business applications to the Application Client. It also details the creation and deployment of mapping tables as part of the process of integrating applications with Digital DEC/EDI.

- *Digital DEC/EDI: User's Guides (Digital UNIX and OpenVMS)*

  These guides contain information on setting up and operating Digital DEC/EDI systems. They also contain information covering configuration, maintainance and problem solving.

- *Digital DEC/EDI: Release Notes*

  Further to the above, each software kit contains a set of release notes applicable to that software. These release notes contain information about known product problems (with workarounds where appropriate) and any operational tips or hints not provided as part of the above documentation set. You are strongly recommended to review these release notes before installing the software. Refer to the appropriate installation guide for information on how to locate the release notes.

Comprehensive on-line documentation is supplied as part of the Digital DEC/EDI software: for example, on-line help libraries and manpage help information. In addition the Digital DEC/EDI Cockpit kit contains the Digital DEC/EDI: Error Messages Help Library. This contains all error messages the product may log along with a description of why the message occurred and what to do about it. It is provided in MS-Windows help library format.

Digital DEC/EDI InfoCenter

For further information on Digital's EDI and Electronic Commerce Solutions and Services, please visit the EDI InfoCenter on the World Wide Web. The location is:

```
http://www.decedi.com
```

Related Third Party Documentation

Refer to the documentation provided with third-party products for installation and configuration details.

Documentation on Tools Supplied with Digital DEC/EDI

There are a number of tools provided with the Digital DEC/EDI Server, in the directory DECEDI$TOOLS. Some of the tools have their documentation with them in DECEDI$TOOLS (placed there by the Digital DEC/EDI installation procedure); other tools are documented in the Digital DEC/EDI books. The tools and their documentation are listed below.

- Data Label Generator

  This tool generates data label definitions for Digital DEC/EDI. It can create data labels for an entire version of a standard, or data labels that are restricted to a trading partner-specific definition of a document within the standard and version. This tool reduces the amount of setup you need to do when installing versions of a standard into Digital DEC/EDI.

  For documentation on how to use this tool, see either of the following files:

  ```
  DECEDI$TOOLS:DECEDI$DLG_DOC.TXT (ASCII format)
  ```

- Table Extractor and Loader

  The Digital DEC/EDI Table Extractor and Loader is a tool that enables you to extract or load the table definitions used by the Translation Service. Using this tool, you can extract an existing definition and load it back into the same system, for example, to use with a different version of

the document standard, or with a different trading partner. Alternatively you can load the definition into an entirely different Digital DEC/EDI system, for example when upgrading to a later release of the Digital DEC/EDI software.

You can also use the Table Extractor and Loader to extract table definitions from a Digital DEC/EDI system installed on OpenVMS VAX and load them into one installed on OpenVMS Alpha, and vice versa.

The Table Extractor and Loader can extract and load any of the following definitions:

– User defined or modified document definitions.

– User defined or modified segment, element, sub-element, and code definitions, including data labels.

– Trading partner specific document, segment, and element definitions, including data labels.

– Value validation definitions.

– Code translation definitions.

For documentation on using this tool, see either of the following files:

```
DECEDI$TOOLS:DECEDI$TEL_DOC.TXT (ASCII format)
```

- Digital DEC/EDI Tailoring Tool

  This tool enables you to do any of the following:

  – Move on-line or archive storage directories to a different disk or directory.

  – Add new on-line or archive storage directories.

  – Move all or part of the Digital DEC/EDI Archive or Audit Database to a different disk or directory.

  – Change the size of the Digital DEC/EDI Archive or Audit Database.

  See *Digital DEC/EDI: OpenVMS User Support Manual* for more information about this tool.

- Digital DEC/EDI Database Tuning Tool

  This tool automatically tunes your Rdb database. The Database Tuning Utility (TEDI) permits users to modify and tune key Rdb database parameters relevant to the performance of Digital DEC/EDI.

TEDI is the main tuning tool for the Digital DEC/EDI Rdb database. It is provided as an image called DECEDI$TUNE_DB.EXE in the directory DECEDI$TOOLS.

TEDI allows you to modify the following Rdb audit database parameters:

– the number of Rdb users for the database

– whether or not to use snapshots.

– the number of local & global buffers

– whether or not to use global buffers, if so :

1. the number of global buffers
2. the user limit on global buffers

TEDI has two modes of operation: SET and TUNE.

The SET option modifies the Rdb database with the parameters you specify. No automatic calculations are performed with this option.

The TUNE option calculates the parameters and optionally modifies the Rdb database to the new calculated parameters. The current machine environment is examined to determine the optimum parameter settings.

For documentation on how to use this tool, see either of the following files:

```
DECEDI$TOOLS:DECEDI$TUNE_DB.TXT (ASCII format)
```

- Mapping Table Export Tool

  This tool enables you to export Mapping Tables that have been developed using the MAPPING TABLE user interface to the source format required by the Digital DEC/EDI CommandCenter Mapping Table Editor.

  This enables you to preserve much of your investment in developing Mapping Tables, when migrating to using the Mapping Table Editor.

  For documentation on how to use this tool, see either of the following files:

```
DECEDI$TOOLS:DECEDI$FBEXPORT_USER_GUIDE.TXT (ASCII format)
```

- DEC/EDI Avail_table Recovery tool

  This Tool enables you to insert records available for fetching in tlf_table and not present in avail_table to avail_table.

Typographical conventions

Digital DEC/EDI / Compaq DEC/ EDI

The ownership of DEC/EDI was transferred to Digital GlobalSoft Ltd, a subsidiary of Compaq Computer Corporation based in India with effect from May 1, 2001. Consequent to this transfer, the name of the product was changed to Digital DEC/EDI. There may be references made to the existing name of the product, Compaq DEC/EDI in various sections of the documentation and screen display. We are in the process of implementing the name change across the product code and documentation. This is expected to be completed within the next couple of months. Pending the completion of this, all references to Compaq DEC/EDI in the documentation pertain to the Digital DEC/EDI product. Please refer to the product website at `www.decedi.com` for further information on the transfer of ownership.

Part I Getting Started with Digital DEC/EDI: A Tutorial

This part of the book presents a detailed step-by-step guide to developing a Mapping Table: one for an outgoing table, and the other for in incoming table.

The Mapping Tables are based on the simple Trading Partner scenario described in the system verification example in *Digital DEC/EDI: Introduction.* You must complete the system verification before proceeding with this tutorial.

It is advisable to be familiar with the information in *Digital DEC/EDI: Introduction* and *Digital DEC/EDI: Installation.*

Chapter 1 Creating an Outgoing Mapping Table

This chapter guides you through the steps involved in developing the Mapping Table for an example outgoing invoice message.

Getting Started

You may use either the CommandCenter Mapping Table Editor application, or the Mapper development user interface (OpenVMS only) to develop a Mapping Table. The examples referenced in this manual are based on the use of the Mapping Table Editor. Equivalent functionality is provided on OpenVMS through the Mapper user interface, except where otherwise stated.

CommandCenter

To start the Mapping Table Editor, click on the Mapping Table Editor icon in the Digital DEC/EDI CommandCenter program group, or from the start menu.

On starting up the Mapping Table Editor application, a copyright screen is displayed for a short period. After reading this you may either wait until it disappears, or press any key to continue immediately.

The Getting Started dialog is then displayed.

Once you are familiar with using the Mapping Table Editor you may disable the Getting Started dialog, by clearing the tick box.

Mapping Table

To start the Mapper Development User Interface on OpenVMS, use the `Interchange edit mapping_table` command, followed by the name of the Mapping Table you wish to edit or create:

```
$ INTERCHANGE EDIT MAPPING_TABLE MINVOX_OUT.FBO
```

The Map Navigator

CommandCenter The Map Navigator is a dialog screen designed to assist with the steps involved in creating or editing a Mapping Table. It displays a list of the main steps, and allows you to select the next step to be executed. It also displays an overview of purpose of each step

Using the Map Navigator During Editing

If you wish to use the Map Navigator at any stage from within the Mapping Table Editor, you may select the Map navigator icon on the toolbar. The Map Navigator Panel then appears.

Select the Map Navigator by clicking on the button containing the Navigator icon.

A dialog appears which gives you a choice of creating a new Mapping Table or editing an existing Mapping Table.

Select the option, Defining the Mapping Table Attributes

On creating a new Mapping Table, the Mapping Table Attributes dialog appears.

This allows you to define global attributes for the Mapping Table, such as the name, direction and the names of any Applications that may use this Mapping Table. The dialog contains a series of tabbed sections that you may define if required.

Mapping Table Attributes — Usage Tab

The Usage tab allows you to define the Table Name, Direction and Table Notes. Once the Mapping Table Direction has been defined, you may not subsequently change it, as any Mapping Assignments depend on the Direction.

The default Mapping Table name is `MAP1`, change this to `TUTOR_O` by typing the new name in the Table Name field. The trailing `_O` is convention that is often used to distinguish outgoing and incoming Mapping Tables, though it is not necessary for you to follow this convention.

Ensure that the Direction indicator is set to Outgoing. This specifies that mapping assignments will be made from the Application File to the Internal Format File.

You may enter any notes on the usage of the Mapping Table in the Table Notes section. As a tip, it is often useful to record the history of any major changes made to the Mapping Table in this section.

Mapping Table Attributes — Security Tab

The Security tab allows you to enter the identifiers of any Applications that may use this Mapping Table. Any other Applications that specify the use of this Mapping Table will fail at run time.

Add the following Application ID to the list of Application IDs for this Mapping Table:

```
DEC-DIRECT-UK-LTD
```

You may define up to twenty Application IDs for a Mapping Table. If more than one Application ID is defined, then a specific one must be given as the Named Application on the Application Client call.

The Application ID must be defined in a Trading Partner Agreement, using the Trading Partner Profile editor application.

Mapping Table Attributes — Defaults Tab

The Defaults tab allows you to specify default options to be used by the Mapper at run time. This means that when this Mapping Table is used, you do not need to specify all options on the Application Client call.

In this example, the Partner ID is always `SHINY-NEW-SYSTEMS` so this can be specified as a default option. Enter this value in the Partner ID field:

```
SHINY-NEW-SYSTEMS
```

The remaining options are specified either on the Application Client call, or selected as part of the mapping process.

Mapping Table Attributes — Auditing Tab

The Auditing tab allows you to specify additional auditing to be applied by the Mapper when this Mapping Table is used.

No additional auditing is required for the purposes of this tutorial.

On selection OK, both the Mapping Table document window and the Map Navigator panel appear.

You may edit the Mapping Table Attributes at any time, either by double clicking on the entry in the Mapping Table document window, or by selecting the Details option from the Table menu while the Mapping Table document window is active.

Extracting the EDI Document

Each Mapping Table must contain definitions of the structure of each EDI Document Type that can be mapped to or from using this Table. This includes definitions of the EDI document structure, as well as the structure of Segments and Data Labels used within the document.

Prior to extracting the EDI Document from the Server system, the EDI standard, version and documents must be defined. In addition, Data Labels must be generated for the standard and version. You must perform these steps as part of the Digital DEC/EDI Server installation and configuration.

For the purposes of this example, a private definition based on the EDIFACT 90.1 standard is used. Private definitions are created using the EDI Table Editor application.

The Digital DEC/EDI Internal Document Type used for the `MINVOX` document is `MINVOICE`. Select the `MINVOICE` definition and wait for the document extract to complete.

The EDI Document and EDI Document Sequence windows appear, showing the content and structure of the EDI definition.

Defining the Record Layouts

In this example, the Application File is an ASCII text file in stream format, where each line represents a single Record.

Each Record is composed of a number of Fields and Structures of known size and data type. You define the layout of each Record in terms of its component Fields.

Each Record Layout has a unique name which you assign to it. Each Field has a Level Number, Field Name, Data Type and Size.

From analyzing your Application File, you have determined that the records have the structure shown in Table 1-1 *Record Layouts*.

Table 1-1 Record Layouts

Level	Field Name	Data Type	Size
Record: INVOIC_HDR			
1	RC	ALPHABETIC	1
1	DOC_NAME_CODE	TEXT	3
1	DOC_REF_NUM	TEXT	18
1	DOC_DATE	TEXT	6
1	DOC_TIME	TEXT	4
1	DOC_FUNC_CODE	TEXT	2
Record: NAME_AND_ADDRESS			
1	RC	ALPHABETIC	1
1	PARTY_QUAL	TEXT	2
1	PARTY_NAME	TEXT	35
1	STREET_1	TEXT	35
1	STREET_2	TEXT	35
1	STREET_3	TEXT	35
1	CITY	TEXT	35
1	COUNTY	TEXT	9
1	POST_CODE	TEXT	9
1	COUNTRY	TEXT	2
Record: DELIVERY_LOC			
1	RC	ALPHABETIC	1
1	LOC_QUAL	TEXT	3
1	LOC_NAME	TEXT	17
Record: PAY_TERMS			

Table 1-1 Record Layouts (continued)

Level	Field Name	Data Type	Size
1	RC	ALPHABETIC	1
1	TERMS_TYPE	TEXT	2
1	TERMS_DATE	TEXT	6
1	DATE_QUAL	TEXT	3
1	MIN_DUE	UNSIGNED NUMERIC	15 SCALE -2
1	PERCENT_PAYABLE	UNSIGNED NUMERIC	7
1	PAYMENT_TERMS	TEXT	35
Record: LINE_ITEM			
1	RC	ALPHABETIC	1
1	ITEM_NO	UNSIGNED NUMERIC	6
1	ARTICLE_NO	TEXT	35
1	ARTICLE_ID	TEXT	3
1	UNIT_PRICE	UNSIGNED NUMERIC	15 SCALE -2
1	PRICE_TYPE	TEXT	2
1	NUM_UNITS	UNSIGNED NUMERIC	9
Record: PACKAGE			
1	RC	ALPHABETIC	1
1	NUM_PACKAGES	UNSIGNED NUMERIC	6
1	PACKAGE_CODE	TEXT	7

To enter the Record Layouts, you may use either the Layouts menu or the right hand mouse button. The Add option is used to add a new Record Layout, Field or structured Field.

Defining the Record Sequence

You define the Application File as a sequence of Records which may be simple Records, Structures or Arrays.

For an outgoing Application File, when you specify the Record Sequence, you must also specify a Recognition Expression, which enables the Mapper to uniquely identify each Record as it is read from the Application File.

In this example, the first Field of each Record contains a unique character which identifies the Record Type. However, in general a recognition expression can be any valid Mapping language expression.

You define the Record Sequence using the Records whose layout you have already defined. Within the Record Sequence, you can specify how many times the Record can occur, whether the Record marks the beginning of a Document or Batch of Documents, or whether it marks the end of a Document.

You can also specify values for the special Global Variables which determine the Object Name ($DOCTYPE) and Partner ID ($PARTNER) to be used once this Record has been encountered.

The Level Number indicates the nesting level of the Record. In this example, the data for each new document starts with an `INVOICE_HDR` Record, and all other Records are subordinate to it. An Application File may contain data for one or more EDI Documents.

Table 1-2 Record Layouts

Level	Record Name	Min	Max	Recog-nition	Object Name	Notes
1	INVOICE_HDR	1	1	RC=’B’	MIN-VOICE	Begins Doc
2	NAME_AND_ADDRESS	1	MANY	RC=’N’		
2	DELIVERY_LOC	1	MANY	RC=’D’		

Table 1-2 Record Layouts (continued)

Level	Record Name	Min	Max	Recog-nition	Object Name	Notes
2	PAY_TERMS	1	MANY	RC='P'		
2	LINE_ITEM	1	MANY	RC='L'		
3	PACKAGE	1	MANY	RC='C'		

You enter the Record Sequence using the Add option in the Sequences menu. This presents the Record Sequence dialog.

Creating a Mapping Set

For an outgoing Mapping Table, a Mapping Set contains a set of assignments to Data Labels in the destination EDI Document. Data Labels correspond to individual Elements or Sub-elements within a Data Segment.

When you create a Mapping Set, you give an Object Name and optionally a Partner ID. These are used as selection criteria for the Mapping Set.

You create a Mapping Set using the **Create...Set** option of the Table menu. Mapping Set

The Browse button shows the EDI Document definitions that have already been extracted into this Mapping Table. You may select one of these.

Select the previously extracted EDIFACT 90.1 MINVOX document and press OK.

This process automatically generates the Mapping Set, Index of Maps and initial Map Parts containing empty assignment statements.

Creating the Mapping Assignments

The next step is to assign data values to Segments in the destination EDI Document. Each Segment is comprised of a number of elements or sub-elements which are identified by Data Labels.

Index of Maps

The Index of Maps shows the structure of the EDI Document to be generated.

Each node in the Index of Maps is visited in turn, to determine whether data can be generated for that node.

The Heading, Detail and Summary nodes do not actually generate any Segment in the destination EDI Document, but may be used to specify initial values or global variables, which apply to that section of the Document.

Initializations

You define any global variables to be used during the Mapping Assignments using the Initializations screen.

Use the Map Navigator or Mapping menu to show the Initializations screen. To add an Initialization, select the Create option from the Initializations menu, and assign a value to the global variable.

You do not need to specify a data type for the value, as this is implied by the context or value assigned.

You have determined that the global variables shown in Table 1-3 *Global Variables* will be used within the Mapping Table.

Table 1-3 Global Variables

Variable	Initial Value
amount_due	0
vat_rate	17.5
message_amount	0
line_total	0
taxable_amount	0
tax_amount	0

Lookups

The Mapping Table may contain Lookup Tables, as a convenient means of mapping one data value to another. For example, this is useful if your Application File contains a brief identifier or part number which is to appear in some other form in the EDI Document.

The example Application File contains some brief identifiers for the package type, which need to be mapped as shown in Table 1-4 *Lookup Values for Package Type*.

Table 1-4 Lookup Values for Package Type

From	To
WB	wickerbottle
CF	coffer
BE	bundle

Mapping Assignments

The Mapping Assignments to be made are shown in Table 1-5 *Mapping Assignments*.

Table 1-5 Mapping Assignments

Mapping Assignment	Comment
Heading Area	
amount_due = 0	
message_amount = 0	
line_total = 0	
taxable_amount = 0	
tax_amount = 0	
BGM Context: INVOICE_HDR Beginning of Message	

Table 1-5 Mapping Assignments (continued)

Mapping Assignment	Comment
BGM_C002_1001 = DOC_NAME_CODE	Document name, coded
BGM_1004 = DOC_REF_NUM	Document number
BGM_C031_2001 = DOC_DATE	Date, coded
BGM_C031_2002 = DOC_TIME	Time
BGM_1225 = DOC_FUNC_CODE	Message function code
NAD Context: NAME_AND_ADDRESS Name And Address	
NAD_3035 = PARTY_QUAL	Party Qual
NAD_C080_3036_1 = PARTY_NAME	Party Name
NAD_C059_3042_1 = STREET_1	Street and Number
NAD_C059_3042_2 = STREET_2	Street and Number
NAD_C059_3042_3 = STREET_3	Street and Number
NAD_3164 = CITY	City Name
NAD_3229 = COUNTY	Country Sub-entity
NAD_3251 = POST_CODE	Post code
NAD_3207 = COUNTRY	Country, coded
LOC Context: DELIVERY_LOC Location Identifier	
LOC_3227 = LOC_QUAL	Place/location qualifier
LOC_C087_3224 = LOC_NAME	Place/location name
PAT Context: PAY_TERMS Payment Terms Basis	
PAT_4279 = TERMS_TYPE	Payment terms type, coded
PAT_C012_2001 = TERMS_DATE	Date, coded
PAT_C012_2005 = DATE_QUAL	Date/time qualifier

Table 1-5 Mapping Assignments (continued)

Mapping Assignment	Comment
PAT_5306 = MIN_DUE	Minimum amount due
PAT_5484 = PERCENT_PAYABLE	Percent of invoice payable
PAT_C104_4276_1 = PAYMENT_TERMS	Terms of payment

Table 1-5 Mapping Assignments (continued)

Mapping Assignment	Comment
Detail Area	
LIN Context: LINE_ITEM Line Item	
LIN_1082 = ITEM_NO	Line item number
LIN_C198_7020_1 = ARTICLE_NO	Article number
LIN_C198_7023_1 = ARTICLE_ID	Article number identifier
LIN_C118_5110 = UNIT_PRICE	Unit price
LIN_C118_5375 = PRICE_TYPE	Price type code
LIN_6170 = NUM_UNITS	Number of pricing units
amount_due = UNIT_PRICE * NUM_UNITS	
message_amount = message_amount + amount_due	
line_total = message_amount	
LIN_5116 = $ROUND(amount_due,2)	
PAC Context: PACKAGE Package	
PAC_7224 = NUM_PACKAGES	Number of packages
PAC_C202_7065 = PACKAGE_CODE	Type of packages, coded
PAC_C202_7064 = $LOOKUP(package_types, PACKAGE_CODE)	
Summary Area	
TMA Total Message Amount	
tax_amount = (message_amount*vat_rate) * 0.01	
message_amount = message_amount + tax_amount	
TMA_5356 = $ROUND(message_amount,2)	

Table 1-5 Mapping Assignments (continued)

Mapping Assignment	Comment
TMA_5360 = $ROUND(line_total,2)	
TMA_5338 = $ROUND(message_amount,2)	
TMA_5492 = $ROUND(tax_amount,2)	

To edit an assignment, click on an assignment line within the Mapping Table document view. This causes the assignment expression to be displayed in the assignment editor at the top of the screen.

Type in the data required for the assignment, or select it using the Expression dialog box. Once the assignment is correct, use the tick button to make the changes.

Compiling the Mapping Table

Once all the changes have been made to the Mapping Table, you may compile the Mapping Table. This validates the Mapping Table, and compiles it into binary form for processing by the Mapper on the Server system.

CommandCenter

In File menu, choose the options tab and set the Compiler Target Version, same as the version of the DEC/EDI server. Select the Compile option from the Mapping menu, and click on OK to confirm the compilation.

This displays a Compiler Results panel, which gives details of any compilation errors encountered, as well as information about the compilation process.

Once the Mapping Table has successfully compiled, a dialog box appears to ask whether the compiled Mapping Table is to be copied to the Server system. You may specify the Server to copy the compiled Mapping Table to.

Once the Mapping Table has been copied to the Server System, you may use the outgoing Mapping Table for any subsequent `post` requests from the Application Client.

Mapping Table

Either select the Compile option from the Exit menu, or use the `INTERCHANGE EDIT MAPPING_TABLE/COMPILE` command:

```
$ INTERCHANGE EDIT MAPPING_TABLE/COMPILE
```

This creates a new version of the Mapping Table file, which contains the compiled Mapping information. The compiled Mapping Table file must be copied to the the directory DECEDI$MAPS, in order for the Mapping Service to use it:

```
$ COPY MINVOX_OUT.FBO DECEDI$MAPS:
```

The Mapping Service automatically recaches any Mapping Tables that have been replaced in this directory, the next time the Mapping Table is used in a client POST or FETCH request.

Summary

You should now have completed the definition of the outgoing Mapping Table for the EDIFACT 90.1 MINVOX document.

Chapter 2 Creating an Incoming Mapping Table

This chapter describes the steps involved in creating the incoming Mapping Table. Use the Map Navigator to help you step through the sequence of tasks.

Defining the Mapping Table Attributes

On creating a new Mapping Table, the Mapping Table Attributes dialog appears.

This allows you to define global attributes for the Mapping Table, such as the name, direction and the names of any Applications that may use this Mapping Table. The dialog contains a series of tabbed sections that you may define if required.

Mapping Table Attributes — Usage Tab

The Usage tab allows you to define the Table Name, Direction and Table Notes. Once the Mapping Table Direction has been defined, you may not subsequently change it, as any Mapping Assignments depend on the Direction.

Mapping Table Attributes — Security Tab

The Security tab allows you to enter the identifiers of any Applications that may use this Mapping Table. Any other Applications that specify the use of this Mapping Table will fail at run time.

Add the Application ID `SHINY-NEW-SYSTEMS` to the list of Application IDs for this Mapping Table.

You may define up to twenty Application IDs for a Mapping Table. If more than one Application ID is defined, then a specific one must be given as the Named Application on the Application Client call.

The Application ID must be defined in a Trading Partner Agreement, using the Trading Partner Profile editor application.

Mapping Table Attributes — Defaults Tab

The Defaults tab allows you to specify default options to be used by the Mapper at run time. This means that when this Mapping Table is used, you do not need to specify all options on the Application Client call.

In this example, the Partner ID is always `DEC-DIRECT-UK-LTD` so this can be specified as a default option. Enter this value in the Partner ID field.

The remaining options are specified either on the Application Client call, or selected as part of the mapping process.

Mapping Table Attributes — Auditing Tab

The Auditing tab allows you to specify additional auditing to be applied by the Mapper when this Mapping Table is used.

No additional auditing is required for the purposes of this tutorial.

On selection OK, both the Mapping Table document window and the Map Navigator panel appear.

You may edit the Mapping Table Attributes at any time, either by double clicking on the entry in the Mapping Table document window, or by selecting the Details option from the Table menu while the Mapping Table document window is active.

Using the Map Navigator During Editing

If you wish to use the Map Navigator at any stage from within the Mapping Table Editor, you may select the Map navigator icon on the toolbar. Extracting the EDI Document

Each Mapping Table must contain definitions of the structure of each EDI Document Type that can be mapped to or from using this Table. This

includes definitions of the EDI document structure, as well as the structure of Segments and Data Labels used within the document.

Prior to extracting the EDI Document from the Server system, the EDI standard, version and documents must be defined. In addition, Data Labels must be generated for the standard and version. You must perform these steps as part of the Digital DEC/EDI Server installation and configuration.

For the purposes of this example, a private definition based on the EDIFACT 90.1 standard is used. Private definitions are created using the EDI Table Editor application.

The Digital DEC/EDI Internal Document Type used for the `MINVOX` document is `MINVOICE`. Select the `MINVOICE` definition and wait for the document extract to complete.

The EDI Document and EDI Document Sequence windows appear, showing the content and structure of the EDI definition.

Defining the Record Layouts

In this example, the Application File is an ASCII text file in stream format, where each line represents a single Record.

Each Record is composed of a number of Fields and Structures of known size and data type. You define the layout of each Record in terms of its component Fields.

Each Record Layout has a unique name which you assign to it. Each Field has a Level Number, Field Name, Data Type and Size.

From analyzing your Application File, you have determined that the records have the structure shown in Table 2-1 *Record Layouts*.

Table 2-1 Record Layouts

Level	Field Name	Data Type	Size
Record: INVOIC_HDR			
1	RC	ALPHABETIC	1
1	DOC_NAME_CODE	TEXT	3

Table 2-1 Record Layouts (continued)

Level	Field Name	Data Type	Size
1	DOC_REF_NUM	TEXT	18
1	DOC_DATE	TEXT	6
1	DOC_TIME	TEXT	4
1	DOC_FUNC_CODE	TEXT	2
Record: NAME_AND_ADDRESS			
1	RC	ALPHABETIC	1
1	PARTY_QUAL	TEXT	2
1	PARTY_NAME	TEXT	35
1	STREET_1	TEXT	35
1	STREET_2	TEXT	35
1	STREET_3	TEXT	35
1	CITY	TEXT	35
1	COUNTY	TEXT	9
1	POST_CODE	TEXT	9
1	COUNTRY	TEXT	2
Record: DELIVERY_LOC			
1	RC	ALPHABETIC	1
1	LOC_QUAL	TEXT	3
1	LOC_NAME	TEXT	17
Record: PAY_TERMS			
1	RC	ALPHABETIC	1
1	TERMS_TYPE	TEXT	2
1	TERMS_DATE	TEXT	6
1	DATE_QUAL	TEXT	3

Table 2-1 Record Layouts (continued)

Level	Field Name	Data Type	Size
1	MIN_DUE	UNSIGNED NUMERIC	15 SCALE -2
1	PERCENT_PAYABLE	UNSIGNED NUMERIC	7
1	PAYMENT_TERMS	TEXT	35
Record: LINE_ITEM			
1	RC	ALPHABETIC	1
1	ITEM_NO	UNSIGNED NUMERIC	6
1	ARTICLE_NO	TEXT	35
1	ARTICLE_ID	TEXT	3
1	UNIT_PRICE	UNSIGNED NUMERIC	15 SCALE -2
1	PRICE_TYPE	TEXT	2
1	NUM_UNITS	UNSIGNED NUMERIC	9
Record: PACKAGE			
1	RC	ALPHABETIC	1
1	NUM_PACKAGES	UNSIGNED NUMERIC	6
1	PACKAGE_CODE	TEXT	7
Record: INVOICE_TLR			
1	RC	ALPHABETIC	1
1	MESSAGE_AMT	UNSIGNED NUMERIC	15 SCALE -2

Table 2-1 Record Layouts (continued)

Level	Field Name	Data Type	Size
1	LINE_ITEM_TOTAL	UNSIGNED NUMERIC	15 SCALE -2
1	TAXABLE_AMT	UNSIGNED NUMERIC	15 SCALE -2
1	TAX_AMT	UNSIGNED NUMERIC	15 SCALE -2

To enter the Record Layouts, you may use either the Layouts menu or the right hand mouse button. The Add option is used to add a new Record Layout, Field or structured Field.

Defining the Record Sequence

You define the Application File as a sequence of Records which may be simple Records, Structures or Arrays.

You define the Record Sequence using the Records whose layout you have already defined. Within the Record Sequence, you can specify how many times the Record can occur.

For an incoming Application File, when you specify the Record Sequence, you specify the minimum and maximum number of times that the record can occur at that portion in the Application File. There is no need to specify a Recognition Expression for an Incoming Mapping Table.

The Level Number indicates the nesting level of the Record. In this example, the data for each new document starts with an `INVOICE_HDR` Record, and all other Records are subordinate to it. An Application File may contain data produced by one or more EDI Documents. The Record

Sequence for the Incoming Mapping Table is shown in Table 2-2 *Record Sequence*.

Table 2-2 Record Sequence

Level	Record Name	Min	Max
1	INVOICE_HDR	1	1
2	NAME_AND_ADDRESS	1	MANY
2	DELIVERY_LOC	1	MANY
2	PAY_TERMS	1	MANY
2	LINE_ITEM	1	MANY
3	PACKAGE	1	MANY
2	INVOICE_TLR	1	MANY

You enter the Record Sequence using the Add option in the Sequences menu.

Creating a Mapping Set

For an incoming Mapping Table, a Mapping Set contains a set of assignments to fields within the records that comprise the destination Application File.

When you create a Mapping Set, you give an Object Name and optionally a Partner ID. These are used as selection criteria for the Mapping Set.

You create a Mapping Set using the **Create...Set** option of the Table menu. The Browse button shows the EDI Document definitions that have already been extracted into this Mapping Table. You may select one of these.

Select the previously extracted EDIFACT 90.1 MINVOX document and press OK.

This process automatically generates the Mapping Set, Index of Maps and initial Map Parts containing empty assignment statements.

Creating the Mapping Assignments

The next step is to assign data values to fields within the destination Application File.

Index of Maps

The Index of Maps shows the structure of the Application File to be generated.

Each node in the Index of Maps is visited in turn, to determine whether data can be generated for that node.

Initializations

You define any global variables to be used during the Mapping Assignments using the Initializations screen.

Use the Map Navigator or Mapping menu to show the Initializations screen. To add an Initialization, select the Create option from the Initializations menu, and assign a value to the global variable.

You do not need to specify a data type for the value, as this is implied by the context or value assigned.

You have determined that the global variable shown in Table 2-3 *Global Variables* will be used within the Mapping Table.

Table 2-3 Global Variables

Variable	Initial Value
total	0

Lookups

The Mapping Table *may* contain Lookup Tables, as a convenient means of mapping one data value to another. Although this tutorial does *not* require a Lookup Table, it would be useful if your Application File contains a brief identifier or part number which is to appear in some other form in the Application File.

Mapping Assignments

The Mapping Assignments to be made are shown in Table 2-4 *Mapping Assignments*.

Table 2-4 Mapping Assignments

Mapping Assignment	Comment
Heading Area	
INVOICE_HDR Context: BGM Beginning of Message	
DOC_NAME_CODE = BGM_C002_1001	Document name, coded
DOC_REF_NUM = BGM_1004	Document number
DOC_DATE = BGM_C031_2001	Date, coded
DOC_TIME = BGM_C031_2002	Time
DOC_FUNC_CODE = BGM_1225	Message function code
NAME_AND_ADDRESS Context: NAD Name And Address	
PARTY_QUAL = NAD_3035	Party Qual
PARTY_NAME = NAD_C080_3036_1	Party Name
STREET_1 = NAD_C059_3042_1	Street and Number
STREET_2 = NAD_C059_3042_2	Street and Number
STREET_3 = NAD_C059_3042_3	Street and Number
CITY = NAD_3164	City Name
COUNTY = NAD_3229	County Sub-entity
POST_CODE = NAD_3251	Post Code
COUNTRY = NAD_3207	Country, coded
DELIVERY_LOC Context: LOC Location Identification	
LOC_QUAL = LOC_3227	Place/location qualifier
LOC_NAME = LOC_C087_3224	Place/location name

Table 2-4 Mapping Assignments (continued)

Mapping Assignment	Comment
PAY_TERMS Context: PAT Payment Terms Basis	
TERMS_TYPE = PAT_4279	Payment terms type, coded
TERMS_DATE = PAT_C012_2001	Date, coded
DATE_QUAL = PAT_C012_2005	Date/time qualifier
MIN_DUE = PAT_5306	Minimum amount due
PERCENT_PAYABLE = PAT_5484	Percent of invoice payable
PAYMENT_TERMS = PAT_C104_4276_1	Terms of payment
Detail Area	
LINE_ITEM Context: LIN Line Item	
ITEM_NO = LIN_1082	Line item number
ARTICLE_NO = LIN_C198_7020_1	Article number
ARTICLE_ID = LIN_C198_7023_1	Article number identifier
UNIT_PRICE = LIN_C118_5110	Unit Price
PRICE_TYPE = LIN_C118_5375	Price type code
NUM_UNITS = LIN_6170	Number of pricing units
amount_due = LIN_5116	Amount due
PACKAGE Context: PAC Package	
NUM_PACKAGES = PAC_7224	Number of packages
PACKAGE_CODE = PAC_C202_7065	Type of packages, coded
PACKAGE_TYPE = PAC_C202_7064_1	Type of package
Summary Area	

Table 2-4 Mapping Assignments (continued)

Mapping Assignment	Comment
INVOICE_TLR Context:TMA Total Message Amounts	
MESSAGE_AMT = TMA_5356	Message amount
LINE_ITEM_TOTAL = TMA_5360	Line item total
TAXABLE_AMT = TMA_5338	Taxable amount
TAX_AMT = TMA_5492	Amount of tax

To edit an assignment, click on an assignment line within the Mapping Table document view. This causes the assignment expression to be displayed in the assignment editor at the top of the screen

Type in the data required for the assignment, or select it using the Expression dialog box. Once the assignment is correct, use the tick button to make the changes.

Compiling the Mapping Table

Once all the changes have been made to the Mapping Table, you may compile the Mapping Table. This validates the Mapping Table, and compiles it into binary form for processing by the Mapper on the Server system.

In File menu, choose the options tab and set the Compiler Target Version, same as the version of the DEC/EDI server. Select the Compile option from the Mapping menu, and click on OK to confirm the compilation.

This displays a Compiler Results panel, which gives details of any compilation errors encountered, as well as information about the compilation process.

Once the Mapping Table has successfully compiled, a dialog box appears to ask whether the compiled Mapping Table is to be copied to the Server system. You may specify the Server to copy the compiled Mapping Table to.

Once the Mapping Table has been copied to the Server System, you may use the incoming Mapping Table for any subsequent POST requests from the Application Client.

Summary

You have now completed the definition of the incoming Mapping Table for the EDIFACT 90.1 MINVOX document.

Chapter 3 Command Line Interface

This chapter describes the commands provided by the Digital DEC/EDI Application Client running on any supported UNIX or OpenVMS platform.

Accessing the Digital DEC/EDI Client Environment

You use the `trade` command to access the Application Client environment.

There are three main environment sub-commands that you use in exchanging, and monitoring the exchange of documents with a trading partner:

- `fetch`

  Use this command to fetch files from the server to which your trading partner has previously posted them.

- `post`

  Use this command to post files to the server from where they can be processed and sent to your trading partner.

- `track`

  Use this command to track the status of files as they progress through the server.

Using Commands

There are three ways in which you can use Application Client commands:

- You can enter the `trade` command to access the Application Client environment, and then enter multiple sub-commands; for example:

UNIX

```
# trade
Compaq DEC/EDI V4.0
© Compaq Computer Corporation 1990, 2001.
All rights reserved.
CLIENTEDI> fetch application [options] file [site_options]
```

OpenVMS

```
$ TRADE
DEC/EDI V4.0
© Compaq Computer Corporation 1990, 2000.
All rights reserved.
CLIENTEDI> FETCH APPLICATION [options] file [site_options]
```

- You can enter a single `trade` command with the relevant option; for example:

UNIX

```
# trade fetch application [options] file [site_options]
```

OpenVMS

```
$ TRADE FETCH APPLICATION [options] file [site_options]
```

You can write a series of commands into a **script file** or **command file** and then execute the file.

UNIX

In the following example, the file, when executed, posts *n* documents (in-house files) into Digital DEC/EDI where *n* is a parameter to the script:

```
#!/bin/csh
#  P1 = Number of documents
#  ++++++++++++++++++++++++
set start_time=‘date‘
set count=1
send_one:
trade post TEST-APPL-A test.document -type=document \
  -partner_name=TEST-TP-B -object_name=PURCHASE-ORDER \
  -tracking_reference="0000000149"
echo "Sent document" $count
@ count++
if ($count > $1) goto all_sent
goto send_one
all_sent:
set end_time=‘date‘
echo "Finished..."
echo ""
echo "Send Start time   = " $start_time
echo "Send End time     = " $end_time
```

```
exit 0
```

OpenVMS

In the following example, the file, when executed, posts *n* documents (in-house files) into Digital DEC/EDI where *n* is a parameter to the command procedure:

```
$ ! P1 = number of documents
$ ! ++++++++++++++++++++++++
$ !
$ start_time = F$TIME()
$ count = 1
$ send_one:
$ TRADE POST TEST-APPL-A test.document /TYPE=DOCUMENT -
  /PARTNER_NAME=TEST-TP-B /OBJECT_NAME=PURCHASE-ORDER -
  /TRACKING_REFERENCE="0000000149"
$ WRITE SYS$OUTPUT “Sent document “, count
$ count = count + 1
$ IF (count > p1) THEN GOTO all_sent
$ GOTO send_one
$ all_sent:
$ end_time = F$TIME()
$ WRITE SYS$OUTPUT "Finished..."
$ WRITE SYS$OUTPUT ""
$ WRITE SYS$OUTPUT "Send Start time   = ", start_time
$ WRITE SYS$OUTPUT "Send End time     = ", end_time
$ EXIT
```

Note that there is a limit to the number of files that you can send in one go from the command line, as determined by the size of the command line buffer used to hold commands as they are being entered. The limit will depend on the operating system envrionment, the length of the file names and the number of options used.

UNIX

You can find an online example of a script file containing **trade** commands in:

```
/usr/examples/decedi/client/cli_exam.sh
```

Commands and Return Status Values

All CLI commands return a status value, and one or more lines of text identifying the final status or any error messages.

On UNIX systems, a success state returns a zero (0). On OpenVMS systems, a success status returns 1 (SS$_NORMAL).

An error state returns an integer greater than 5. The error status values are defined in the include file, `decedi_api_msgs.h`.

On OpenVMS systems, this file is located in `SYS$LIBRARY`, on UNIX systems this file is located in `/usr/include`.

The following table lists each value's meaning.

Table 3-1 Commands and Return Status Values

DECEDI message ID	UNIX status	OpenVMS severity	Message Meaning
Success	1	Success	Success
INSUFVM	6	Error	Insufficient Virtual Memory
INTERROR	7	Error	Internal error
NOTFOUND	8	Warning	No objects found
PARTIALMAP	9	Error	Map partially completed
MAPFAIL	10	Error	Map Failed
NOMAPOUTPUT	11	Error	Map produced no output
BADPARAM	12	Error	Bad parameter
BADITMLST	13	Error	Bad item List
ORBSRVNOTFND	16	Error	No Server Found
ORBSRVDIED	20	Error	Server process died
ORBTIMEOUT	21	Error	Server timeout
ORBBADNODE	22	Error	Bad server selection node
CLIERROR	23	Error	Command Line Interface Error
CLIEXIT	24	Error	Command Line Interface Exiting
SRVERROR	25	Error	Server Error
NOEDISYS	26	Error	EDI Server System not running
NOTAUTH	27	Error	Not authorized to access the EDI Server System
POSTFAIL	28	Error	Failed to post file %s

Table 3-1 Commands and Return Status Values

DECEDI message ID	UNIX status	OpenVMS severity	Message Meaning
FETCHFAIL	29	Error	Failed to fetch file %s
OPENINPERR	30	Error	Could not open one of the input files
OPENOUTERR	31	Error	Could not open one of the output files
ZEROMATCH	32	Error	No objects were found that matched the selection criteria
CLIBADCMD	33	Error	Command syntax incorrect
NOCLIENTLIC	34	Error	No active client license was found
ORBCOMMFAIL	35	Error	Communications to server failed

An Example Script File

As an example of how this information can be used in a script file, the following is an extract from the sample script files provided in each UNIX client kit:

UNIX

```
trade post DEC-DIRECT-UK-LTD minvox_o.dat -table_name=minvox_o
STATUS_VALUE=$?
if [ "$STATUS_VALUE" -ne 0 ]
 then
    case "$STATUS_VALUE" in
        10)
            echo "Use -error_log option on commandline.(edit
this script file)"
            echo ""
            break
            ;;
        12)
            echo "Check your command line."
            echo ""
            break
            ;;
        14)
            echo "See decedi.err for more information."
            echo ""
            break
```

```
            ;;
        18)
            echo "Start it using/usr/etc/ObjectBroker/obbstrt"
            echo ""
            break
            ;;
        25)
            echo "Use Cockpit or see the Error Log
(decedi_look) to see what error occurred."
            echo ""
            break
            ;;
        26)
            echo "Issue command 'decedi_start' on the server"
            echo ""
            break
            ;;
        *)
            echo "Error $STATUS_VALUE occurred."
            echo ""
            break
            ;;
   esac
fi
```

You can find this, and other example files in:

/usr/examples/decedi/client

Commands and Case-sensitivity

UNIX

It is important to note that pre-defined UNIX commands and their parameters are *case-sensitive* (though this may not be so for “alias” commands that you may define).

If you enter a command in the wrong case, it will return a “Bad Parameter” error.

OpenVMS

On OpenVMS systems, commands and their parameters are not case-sensitive, unless the parameter value is enclosed within a quoted string.

Commands and Quoted Strings

If you need to pass a value to the CLI that contains embedded spaces, then the value must appear within a quoted string.

UNIX

On UNIX systems, when passing quoted strings as parameters to the CLI commands, you must use the backslash character "\" to escape the quote character from its special shell meaning. For example:

```
trade post AP file.dat -table_name=map \
-tracking_reference=\”My Reference 001\”
```

For example, to specify -business_references=("A","B") from the Bourne shell you need to type the following

```
trade post AP file.dat -table_name=map \
-business_references=\(\"A\",\"B\"\)
```

trade fetch

The **trade fetch** command enables the user to fetch one or more files from the Digital DEC/EDI Server. These files may have come through the Translation Service (possibly via the Mapping Service) or directly from the Communications Service by using the Translation Bypass facility. The type of file being fetched must be specified and can be one of the following:

- Application_file — files are fetched from the Mapper on the Server. These files are also known as structured files.
- Transmission_file — files are fetched from the Communications Service on the Server, bypassing any translation or mapping. These files are also known as unstructured files.
- Document — files are fetched from the Translation Service on the Server, bypassing any mapping. These files are also known as *ihf* (in-house file) files.

Format

```
trade fetch application_name[command_options][file[file_options]]
```

Parameters

Note: Remember that all parameters are case-sensitive. If they are not specified correctly, the command will return a "Bad Parameter" error.

application_name

The application_name is the name of the client application submitting the request to the Server. This parameter is used by Digital DEC/EDI as a means of authenticating the request. The client application must be registered as an authorized application on the Server.

This parameter is mandatory.

Tru64 UNIX Applications are registered by using the CommandCenter Management Services Editor.

OpenVMS Applications are registered by using the INTERCHANGE command EDIT CONFIGURATION.

For more information, refer to the *User's Guides.*

file

The file specification of one or more files into which files from the Server are to be placed. If more than one file name is entered, the file options immmediately following each file refer only to that file. See *File Options* on page 3-10 for further information on which File options can be (or must be) supplied with this parameter.

If the Server is remote from the client, the client application must make sure that any file specified resides in a directory that allows write access to the Digital DEC/EDI client information server process. This process uses the `decedi` account (DECEDI account on OpenVMS) by default.

This parameter is mandatory.

Command Options

link_id

The identifier of the connection from which files are directly received. These files are data files that do not require translation. This means that they bypass the Translation Services. The files can be fetched when their status (on the server) reaches TRANS_AVAILABLE.

Tru64 UNIX The connection must have been defined by using the CommandCenter Communications Editor.

When the file option is `-type=transmission_file`, this option is mandatory and must be specified in *UPPER CASE*.

OpenVMS The connection must have been defined by using the INTERCHANGE command EDIT CONFIGURATION.

When the file option is `/TYPE=TRANSMISSION_FILE`, this option is mandatory.

Format

UNIX `-link_id=<connection_id>`

OpenVMS `/LINK_ID=<connection_id>`

test_indicator

Indicates whether the files in this request are to be treated as test files or live files.

Format

UNIX `-test_indicator[=option]`

OpenVMS `/TEST_INDICATOR[=option]`

Where option is one of:

live	Live files received from the trading partner or from a remote application. By default, both live and partner_test documents can be fetched.
mapper_test	Files tested through the Mapping Service only. This option is used in conjunction with the **local_test** option to allow test data to be received through the Mapping Service.
partner_test	Files received from the trading partner or from a remote application, that are intended to be used for test purposes only. By default, both live and partner_test documents can be fetched.

File Options

For each file to be fetched, a number of options can be specified. Whether a particular option is mandatory or not depends upon the file type that is specified using the `-type` option or the /TYPE qualifier.:

type=	**Mandatory Options**
application_file	table_name
document	partner_name, object_name, type
transmission_file	link_id†, type

† Note: This is a Command option, not a File option.

Most optional file options are associated with the Mapper only and are not applicable when **`type=transmission_file`** or **`type=document`**:

type=	**Applicable Non-Mandatory Items**
application_file	All except for `table_name` (which is mandatory). These file options override any setup information in the Mapper itself and, if not specified, the defaults come from the Mapper.
document	`timeout`
transmission_file	`timeout`

The File options are associated with each file to be fetched and must be entered *after* the file specification on the command line. For example:

UNIX

```
# trade fetch my-app file.dat -type=application_file \
       -table_name=mapping_tbl -output_file=mapper_out.out
```

OpenVMS

```
# TRADE FETCH my-app file.dat /TYPE=application_file -
       /TABLE_NAME=mapping_tbl /OUTPUT_FILE=mapper_out.out
```

business_references

This associates one or more user supplied business references with the document audit trail. If more than one is supplied then each must be separated by a comma.

Business references may subsequently be used to track the document by using the Digital DEC/EDI client **track** command, the Digital DEC/EDI client API routine **DECEDI_TRACK**, or the Digital DEC/EDI Cockpit.

Leading references may be omitted if a particular business reference slot is intended for the users reference which may have been generated either during the mapping process, or by a remote application.

Up to 5 business references may be specified.

This is applicable only when **`type=application_file`** or **`type=document`**.

Format

UNIX

```
-business_references=<business_reference>
```

```
-business_references=(<business_reference>[,
      <business_reference>... ])
```

OpenVMS

```
/BUSINESS_REFERENCES=<business_reference>
/BUSINESS_REFERENCES=(<business_reference>[,
     <business_reference>... ])
```

comment

A comment to add to the mapping audit trail.

This is applicable only when **`type=application_file`**.

Format

UNIX

```
-comment=<mapping_comment>
```

OpenVMS

```
/COMMENT=<mapping_comment>
```

debug

The file specification of the file on the client to which the Mapper debug output is to be written.

This is applicable only when **`type=application_file`**.

Format

UNIX

```
-debug=<file_specification>
-nodebug
```

OpenVMS

```
/DEBUG=<file_specification>
/NODEBUG
```

error_log

The file specification of the file on the client to which the Mapper error log output is to be written.

This is applicable only when **`type=application_file`**.

Format

UNIX

```
-error_log=<file_specification>
-noerror_log
```

OpenVMS

```
/ERROR_LOG=<file_specification>
```

/NOERROR_LOG

io_debug

The file specification of the file on the client to which the Mapper I/O debug output is to be written.

This is applicable only when **type=application_file**.

Format

UNIX

-io_debug=<file_specification>

-noio_debug

OpenVMS

/IO_DEBUG=<file_specification>

/NOIO_DEBUG

local_test

The file specification of the input file on the Server to use when the Mapper is in local test mode. This is used in conjunction with the command option **test_indicator=mapper_test**.

This is applicable only when **type=application_file**.

Format

UNIX

-local_test=<file_specification>

-nolocal_test

OpenVMS

/LOCAL_TEST=<file_specification>

/NOLOCAL_TEST

match_flag

Specifies which documents the Mapper is to match.

This is applicable only when **type=application_file**.

Format

UNIX

-match_flag[=option]

OpenVMS

/MATCH_FLAG[=option]

Where option is one of:

match_first — match the first available document

`match_all` — match all documents.

`number=n` — match all documents until `n` documents fetched or no more documents available for fetching.

named_application

Indicates that the application name specified as a parameter is to be passed to the Mapper, and is to be used as the Application ID for the duration of the map fetch request. The application name must match the Application ID defined within the trading partner profile.

This option is mandatory if you have specified more than one Application ID, or the special value of ANY, within the security settings of the mapping table.

If this option is not specified, the Mapper uses the Application ID specified within the security settings of the mapping table.

This is applicable only when **`type=application_file`**.

Format

UNIX `-named_application`

OpenVMS `/named_application`

object_name

A name distinguishing this object from other objects handled by the application.

In the case of application files (**`type=application_file`**), it may be used to specify the name of the mapping set to be used, when the mapping table contains more than one mapping set.

In the case of in-house-files (**`type=document`**), it is used to describe what the object is, for example, a purchase order or an invoice. This is the same as the Digital DEC/EDI internal document name.

This is NOT applicable for **`type=transmission_file`** and is mandatory when **`type=document`**.

Format

UNIX `-object_name=<object_name>`

OpenVMS `/OBJECT_NAME=<object_name>`

output_file

The file specification of the file on the client to which screen output from the Mapper is to be written.

This is applicable only when **`type=application_file`**.

Format

UNIX `-output_file=<file_specification>`

OpenVMS `/OUTPUT_FILE=<file_specification>`

partner_name

The name of the trading partner from whom this file was received.

This is a mandatory option for **`type=document`**.

This is NOT applicable when **`type=transmission_file`**.

Format

UNIX `-partner_name=<partner_name>`

OpenVMS `/PARTNER_NAME=<partner_name>`

restart_from

The number of the document from which mapping is to restart.

This is applicable only when **`type=application_file`**.

Format

UNIX `-restart_from=<mapping_restart_position>`

OpenVMS `/RESTART_FROM=<mapping_restart_position>`

Where `<mapping_restart_position>` is an integer.

Where `<mapping_restart_position>` is an integer.

table_name

The name of the mapping table to use if the services of the Mapper are required. Mapping tables reside on the Server in the mapping table directory.

Note: *The directory specification and file extension (.fbo) must NOT be included with the table name.*

This is applicable only when **`type=application_file`** and is a mandatory option.

Format

UNIX `-table_name=<mapping_table_name>`

OpenVMS `/TABLE_NAME=<mapping_table_name>`

timeout

The maximum number of seconds that the Application Client should wait for the file to become available. If not specified, a timeout of zero seconds is assumed.

Format

UNIX `-timeout=<number_of_seconds>`

OpenVMS `/TIMEOUT=<number_of_seconds>`

type

This specifies the type of file being fetched and whether it is to be fetched using the Mapper or not.

Format

UNIX `-type[=option]`

OpenVMS `/TYPE[=option]`

Where option is one of:

type=	Applicable Non-Mandatory Items
application_file	Fetch structured application files from the Mapper. This is the default for type.
transmission_file	Fetch unstructured transmission files directly from the Communications Service on the Server.
document	Fetch in-house files (Digital DEC/EDI Version 1 internal format documents) from the Translation Service on the Server.

Examples

1.

UNIX

```
# trade fetch my-application test_file.dat \
              -partner_name=their-application \
              -business_references="invoice_1234"\
              -table_name=map_test_tbl
```

OpenVMS

```
$ TRADE FETCH my-application test_file.dat -
              /PARTNER_NAME=their-application -
              /BUSINESS_REFERENCES="invoice_1234"-
              /TABLE_NAME=map_test_tbl
```

In this example, an application called `my-application` submits a request to fetch structured files that have been received from a trading partner called `their-application`, convert them into application file format using the mapping table `map_test_tbl`, and write them to the file `test_file.dat` in the user's current working directory.

Note that because the default file type is *application_file*, the **`type`** option is not necessary.

2.

UNIX

```
TRADE
CLIENTEDI> fetch my-app test_file.dat \
                   -partner_name=acme-orders \
                   -table_name=map_test_tbl
```

OpenVMS

```
$ TRADE
CLIENTEDI> FETCH my-app test_file.dat -
                   /PARTNER_NAME=acme-orders -
                   /TABLE_NAME=map_test_tbl
```

IIn this example, an application called `my-app` submits a request to fetch structured application files that have been received from a trading partner called `acme-orders`, convert them into application file format using the mapping `table map_test_tbl`, and write them to the file `test_file.dat` in the user's current working directory.

Note that because the default file type is application_file, the **`type`** *option is not necessary.*

3.

UNIX

```
TRADE
```

```
CLIENTEDI> fetch my-app -link_id=OFTP_1  new_car_design.dat \
                  -type=transmission_file
```

OpenVMS

```
$ TRADE
CLIENTEDI> FETCH my-app /LINK_ID=OFTP_1  new_car_design.dat -
                  /TYPE=transmission_file
```

In this example, an application called `my-app` submits a request to fetch transmission files coming into the Server via the connection identifier, `OFTP_1`. The transmission file is written into the file `new_car_design.dat` in the user's current directory.

4.

UNIX

```
CLIENTEDI> fetch my-app /usr/users/me/test_document.dat \
                -type=document -partner_name=acme-invoic \
                -object_name=edifact-invoic
```

OpenVMS

```
CLIENTEDI> FETCH my-app SYS$LOGIN:test_document.dat -
                /TYPE=document /PARTNER_NAME=acme-invoic -
                /OBJECT_NAME=edifact-invoic
```

In this example, an application called `my-app` submits a request to fetch document files that have been received from a trading partner called `acme-invoic` and have an internal document identifier of `edifact-invoic`.

Note that all of these options are mandatory when **`type=document`**.

trade post

The **trade post** command enables the user to submit one or more files into the Digital DEC/EDI Server. These files may go through the Translation Service (possibly via the Mapping Service) or directly to the Communications Service. The type of file posted must be specified, and can be one of the following:

- Application_file — files are posted to the Mapper on the Server. These files are also known as structured files.
- Transmission_file — files are posted to the Communications Service on the Server, bypassing any translation or mapping. These files are also known as unstructured files.
- Document — files are posted to the Translation Service on the Server bypassing any mapping. These files are also known as *ihf* (in-house file) files.

Several command and file options are associated with the `post` command: some are mandatory.

Format

```
trade post application_name [command_options]
[file [file_options]]...
```

Parameters

Note: Remember that all parameters are case-sensitive. If they are not specified correctly, the command will return a “Bad Parameter” error.

application_name

The application_name is the name of the client application submitting the request to the Server. This parameter is used by Digital DEC/EDI as a means of authenticating the request. The client application must be registered as as authorized application on the Server.

This parameter is mandatory.

Tru64 UNIX

Applications are registered by using the CommandCenter Management Services Editor.

OpenVMS Applications are registered by using the INTERCHANGE command EDIT CONFIGURATION.

For more information, refer to the *User's Guides.*

file

The file specification of one or more files to be posted to the Server. If more than one file name is entered, the *file options* immediately following each file refer only to that file. See *File Options* on page 3-22 for further information on which File options can be (or must be) supplied with this parameter.

If the Server is remote from the client, the client application must make sure that any files specified allow read access to the Digital DEC/EDI client information server process. This process uses the `decedi` account by default.

This parameter is mandatory.

Command Options

connection_data

This provides information to the communications connection which overrides the defaults for that connection. This field is specific to the type of gateway referenced by the **`link_id`**.

Tru64 UNIX Please refer to the CommandCenter Trading Partner Editor's on-line help for types that support this feature, and what format the data should be in.

OpenVMS Please refer to the INTERCHANGE HELP command for types that support this feature, and what format the data should be in.

This is applicable only when **`type=transmission_file`**.

Format

UNIX
```
-connection_data=<connection_specific_string>
```

OpenVMS
```
/CONNECTION_DATA=<connection_specific_string>
```

Where `<connection_specific_string>` is of the format:

```
<initiator_oftp_id>\<virtual_file_name>\
<fixed_record_length>\<user_data>\
```

```
<originator_oftp_id>
```

For example, the following would be valid entries:

```
id999\my_file_name\80\access\
id999\\\access\
\my_file_name\80\\
```

Refer to the *User's Guides* for more information.

link_id

The connection identifier of the connection to which files are posted directly.

Tru64 UNIX The connection must have been defined by using the CommandCenter Communications Editor.

OpenVMS The connection must have been defined by using the INTERCHANGE command EDIT CONFIGURATION.

Refer to the *User's Guide*s for more information.

When the file option is **`type=transmission_file`**, this option is mandatory.

Format

UNIX
```
-link_id=<connection_id>
```

OpenVMS
```
/LINK_ID=<connection_id>
```

priority

Specifies how the Server is to process the files. The default is **`priority=normal`**. The Translation Service builds high priority files into a single document interchange immediately. Normal priority files are built into document interchanges at scheduled intervals. Similarly, the communications gateway triggers the sending of a high priority file immediately and normal priority files are sent at scheduled intervals.

Format

UNIX
```
-priority[=option]
```

OpenVMS
```
/PRIORITY[=option]
```

Where option is one of:

normal	Use scheduled intervals. This is the default value.
high	Documents sent immediately

test_indicator

Indicates whether the files in this request are to be treated as test submissions or live submissions.

Format

UNIX `-test_indicator[=option]`

OpenVMS `/TEST_INDICATOR[=option]`

Where option is one of:

live	Live files sent to the trading partner or to a remote application. This is the default option.
mapper_test	Files tested through to the Mapping Service only. This option is used in conjunction with the **local_test** option to allow test data to be sent through the mapping service.
translation_test	Files tested through to the Translation Service only. The documents are not transmitted to the trading partner.
partner_test	Files tested through all services to the trading partner or to a remote application.

File Options

For each file to be posted, there are a number of options that can be specified. Whether a particular option is mandatory or not depends upon the file type that is specified using the **type** option.:

type=	**Mandatory Options**
application_file	table_name
document	partner_name, object_name, tracking_reference, type
transmission_file	link_id†, type

† Note: This is a Command option, not a File option.

Most optional file options are associated with the Mapper only and are not applicable when **`type=transmission_file`** or **`type=document`**:

type=	**Applicable Non-Mandatory Options**
application_file	All except for **table_name** (which is mandatory). These file options override any setup information in the Mapper itself and, if not specified, the defaults come from the Mapper.
document	None
transmission_file	tracking_references

The File options are associated with each file to be posted and must be entered *after* the file specification on the command line. For example:

UNIX

```
# trade post my-app file.dat -type=application_file\
                                -table_name=mapping_tbl \
                                -output_file=mapper_out.dat
```

OpenVMS

```
$ TRADE POST my-app file.dat /TYPE=application_file -
                               /TABLE_NAME=mapping_tbl -
                               /OUTPUT_FILE=mapper_out.dat
```

business_references

This associates one or more user supplied business references with the document audit trail. If more than one is supplied then each must be separated by a comma.

Business references may subsequently be used to track the document by using the Digital DEC/EDI client **track** command, the Digital DEC/EDI client API routine **DECEDI_TRACK**, or the Digital DEC/EDI Cockpit.

Leading references may be omitted if a particular business reference slot is intended for the users reference which may have been generated either during the mapping process, or by a remote application.

Up to 5 business references may be specified.

This is applicable only when **`type=application_file`** or **`type=document`**.

Format

UNIX

```
-business_references=<business_reference>
-business_references=(<business_reference>[,
      <business_reference>... ])
```

OpenVMS

```
/BUSINESS_REFERENCES=<business_reference>
/BUSINESS_REFERENCES=(<business_reference>[,
     <business_reference>... ])
```

comment

A comment to add to the mapping audit trail.

This is applicable only when **type=application_file**.

Format

UNIX

```
-comment=<mapping_comment>
```

OpenVMS

```
/COMMENT=<mapping_comment>
```

debug

The file specification of the file on the client to which the Mapper debug output is to be written.

This is applicable only when **type=application_file**.

Format

UNIX

```
-debug=<file_specification>
-nodebug
```

OpenVMS

```
/DEBUG=<file_specification>
/NODEBUG
```

error_log

The file specification of the file on the client to which the Mapper error log output is to be written.

This is applicable only when **type=application_file**.

Format

UNIX

```
-error_log=<file_specification>
```

```
-noerror_log
```

OpenVMS

```
/ERROR_LOG=<file_specification>
/NOERROR_LOG
```

io_debug

The file specification of the file on the client to which the Mapper I/O debug output is to be written.

This is applicable only when **`type=application_file`**.

Format

UNIX

```
-io_debug=<file_specification>
-noio_debug
```

OpenVMS

```
/IO_DEBUG=<file_specification>
/NOIO_DEBUG
```

local_test

The file specification of the input file on the Server to use when the Mapper is in local test mode. This is used in conjunction with the command option **`test_indicator=mapper_test`**.

This is applicable only when **`type=application_file`**.

Format

UNIX

```
-local_test=<file_specification>
-nolocal_test
```

OpenVMS

```
/LOCAL_TEST=<file_specification>
/NOLOCAL_TEST
```

named_application

Indicates that the application name specified as a parameter is to be passed to the Mapper, and is to be used as the Application ID for the duration of the map post request. The application name must match the Application ID defined within the trading partner profile.

This option is mandatory if you have specified more than one Application ID, or the special value of ANY, within the security settings of the mapping table.

If this option is not specified, the Mapper uses the Application ID specified within the security settings of the mapping table.

This is applicable only when **type=application_file**.

Format

UNIX `-named_application`

OpenVMS `/NAMED_APPLICATION`

object_name

A name distinguishing this object from other objects handled by the application.

In the case of application files (**type=application_file**), it may be used to specify the name of the mapping set to be used, when the mapping table contains more than one mapping set.

In the case of in-house-files (**type=document**), it is used to describe what the object is, for example, a purchase order or an invoice. This is the same as the Digital DEC/EDI internal document name.

This is NOT applicable for **type=transmission_file** and is mandatory when **type=document**.

Format

UNIX `-object_name=<object_name>`

OpenVMS `/OBJECT_NAME=<object_name>`

output_file

The file specification of the file on the client to which screen output from the Mapper is to be written.

This is applicable only when **type=application_file**.

Format

UNIX `-output_file=<file_specification>`

OpenVMS `/OUTPUT_FILE=<file_specification>`

partner_name

The name of the trading partner for whom this file is destined.

This is a mandatory option for **type=document**.

This is NOT applicable when **type=transmission_file**.

Format

UNIX

```
-partner_name=<partner_name>
```

OpenVMS

```
/PARTNER_NAME=<partner_name>
```

reprocess

A list of documents within an application file to be reprocessed.This is normally used after you have corrected the error that has caused a SOFT ERROR to occur during the processing of a document within an application file.

To determine which documents have failed during processing, use the Cockpit to examine the Mapper audit trail, or the **output** option to capture the status of each document processed.

If the application file contains documents for trading partners for which no agreement has yet been defined within Digital DEC/EDI, you may use the following logical name or environment variable to cause the mapper to report a SOFT ERROR and continue processing the application file:

OpenVMS

```
$ DEFINE/TABLE=DECEDI$LOGICAL_NAMES -
  DECEDI$$NO_TP_AGREE_CONTINUE “1”
```

Tru64 UNIX

```
set DECEDI__NO_TP_AGREE_CONTINUE=1
export DECEDI__NO_TP_AGREE_CONTINUE
```

This is applicable only when **type=application_file**.

Format

UNIX

```
-reprocess=<document_list_string>
```

OpenVMS

```
/REPROCESS=<document_list_string>
```

Where:

document_list_string is one of

description	examples (OpenVMS and UNIX)
digit	/REPROCESS=1 -reprocess=1,2,8
range	/REPROCESS=”1-3” -reprocess=2-5 -reprocess=1,3,7-10
keyword	/REPROCESS=all -reprocess=last /REPROCESS=”1,3,5-last”

restart_from

The number of the document from which mapping is to restart.

This is applicable only when **`type=application_file`**.

Format

UNIX `-restart_from=<mapping_restart_position>`

OpenVMS `/RESTART_FROM=<mapping_restart_position>`

Where `<mapping_restart_position>` is an integer.

table_name

The name of the mapping table to use if the services of the Mapper are required. Mapping tables reside on the Server in the mapping table directory.

Note: *The directory specification and file extension (.fbo) must NOT be included with the table name.*

This is applicable only when **`type=application_file`** and is a mandatory option.

Format

UNIX `-table_name=<mapping_table_name>`

OpenVMS `/TABLE_NAME=<mapping_table_name>`

tracking_reference

This is a tracking reference that the user can apply to the file being posted. The reference can be used with the `track` command to track objects in the system.

This is a mandatory option when **`type=document`**.

Format

UNIX `-tracking_reference=<reference>`

OpenVMS `/TRACKING_REFERENCE=<reference>`

type

This specifies the type of file being posted.

Format

UNIX `-type[=option]`

OpenVMS `/TYPE[=option]`

Where option is one of:

application_file	Structured application files that will go to the Mapper. (This is the default for **`type`**).
transmission_file	Unstructured transmission files that will be sent directly to the Communications Service on the Server.
document	In-house files (Digital DEC/EDI Version 1 internal format documents) that will be processed by the Translation Service on the Server.

Examples

1.

UNIX

```
# trade post my-app -test_indicator=partner_test test_file.dat
\
   -partner_name=acme-order  -tracking_reference="ORDER_1234"\
   -table_name=map_test_tbl
```

OpenVMS

```
$ TRADE POST my-app /TEST_INDICATOR=partner_test file.dat -
   /PARTNER_NAME=acme-order  /TRACKING_REFERENCE="ORDER_1234"-
   /TABLE_NAME=map_test_tbl
```

In this example, an application called `my-app` submits a request to post the application file, `file.dat` in the user's current working directory to a trading partner called `acme-orders` as a test. The file is to be converted into internal (in-house) format using the mapping table `map_test_tbl`. The file is given the reference `ORDER_1234` so that it can be tracked using the `track` command.

Because the default file type is `application_file`*, the* `-type` *option is not necessary.*

2.

UNIX

```
# trade post my-app -link_id=OFTP_1 \
    -connection_data="\new-car-design\80\\" \
    new_car_cad_design.dat -type=transmission_file
```

OpenVMS

```
$ TRADE POST my-app /LINK_ID=OFTP_1 -
    /CONNECTION_DATA="\new-car-design\80\\" -
    new_car_cad_design.dat /TYPE=transmission_file
```

In this example, an application called `my-app` submits a request to post the transmission file, `new_car_cad_design.dat` in the user's current working directory using the connection `OFTP_1` (defined in the Server). The OFTP virtual file name is to be `new-car-design` and the transmission file is to be transmitted using a record size of 80 characters.

Note that this trade post request bypasses any mapping or conversion because the file type is `transmission_file`.

3.

UNIX

```
# trade post my-app -priority=high \
/usr/users/me/invoic_document.dat -partner_name=acme-orders \
   -object_name=edifact-order -tracking_reference=acme_ref \
   -type=document
```

OpenVMS

```
$ TRADE POST my-app /PRIORITY=HIGH -
   SYS$LOGIN:invoic_document.dat /PARTNER_NAME=acme-orders -
   /OBJECT_NAME=edifact-order /TRACKING_REFERENCE=acme_ref -
   /TYPE=DOCUMENT
```

In this POST command, an application called `my-app` submits a request to post the in-house file formatted document in a file called, `invoic_document.dat` to a trading partner called `acme-orders` at high

priority. The name of the internal document is specified as `edifact-order` and a tracking reference is given as `acme_ref`.

4.

UNIX

```
# trade post DEC-DIRECT-UK-LTD minvox_o.dat \
    -table_name=minvox_o minvox_5_times_o.dat \
    -table_name=minvox_o
```

OpenVMS

```
$ TRADE POST DEC-DIRECT-UK-LTD minvox_o.dat -
    /TABLE_NAME=minvox_o minvox_5_times_o.dat -
    /TABLE_NAME=minvox_o
```

In this example, the POST command is used to send several files at once. Note how the `table_name` is treated as a file option. File options must be specified for each application file in the list.

Important: Wildcarding of application names is disallowed.

trade track

The `trade track` command enables the user to track objects within the Digital DEC/EDI Server. It can provide lists of objects that match certain selection criteria and it can return different types of information for the objects that meet this selection criteria (for example routing information or status).

The output from the command shows each object's primary tracking reference at the start of a new line, with its attributes and values on subsequent lines (indented).

The `track` command can track only one kind of object at a time. The type of object being tracked must be specified and may be one of the following:

- Application_file — files that have been posted or fetched via the Mapper on the Server. These files are also known as structured files.
- Transmission_file — files that have been sent or received via the Communications Service on the Server, or are waiting to be sent via the Communications Service. These files are also known as unstructured files.
- Document — files that have gone through or are waiting to go through the Translation Service on the Server. This also includes documents that are involved in Application-to-application agreements. These files are also known as (in-house file) *ihf* files.

Format

```
trade track application_name [options]
selection_list...
```

Parameters

Note: Remember that all parameters are case-sensitive. If they are not specified correctly, the command will return a "Bad Parameter" error.

application_name

The application_name is the name of the client application submitting the request to the Server. This parameter is used by Digital DEC/EDI as a means of authenticating the request. The client application must be registered as as authorized application on the

Server.

This parameter is mandatory.

Tru64 UNIX Applications are registered by using the CommandCenter Management Services Editor.

OpenVMS Applications are registered by using the INTERCHANGE command EDIT CONFIGURATION.

For more information, refer to the *User's Guides.*

selection_list

The selection_list is a space-separated list indicating the values to obtain for each object found. Note that not all selection values are applicable to all file types. For example, if **`type=document`** is specified, then the `selection_list` value, *communications_data*, returns no information.

If a selection value is applicable to only one type, then it is not necessary to explicitly specify that type (**`type`**=) on the command line. For example, if the selector *acknowledgement* is selected, then **`type=document`** can be omitted, since *acknowledgement* is applicable only for that file type.

The options available for the selection_list are shown in Table 3-2 *Selection Identifiers*:

Table 3-2 Selection Identifiers

Value	Description	
tracking_references	Provides all tracking references associated with the file. Data displayed for file types is:	
	application_file	Internal user reference
	transmission_file	Transmission file name
	document	User reference, document count, transmission file name

Table 3-2 Selection Identifiers (continued)

Value	Description	
envelope_references	Provides all EDI envelope control references associated with the object, where applicable. Data displayed for file types is:	
	application_file	Not applicable.
	transmission_file	Not applicable.
	document	Interchange control number, group control number, document control number.
business_references	Provides information about business references associated with the objects. Data displayed for file types is	
	application_file	Not applicable
	transmission_file	Not applicable
	document	Business References (up to 5)
routing	Provides all data associated with the routing of this object through the Digital DEC/EDI Server. Data displayed for file types is:	
	application_file	Application name, partner name, document type, direction.
	transmission_file	Connection id, direction.
	document	Application name, partner name, document type, direction.

Table 3-2 Selection Identifiers (continued)

Value	Description	
status	Returns the current status of the object. It can be one of the following:	
	completed	The object has completed processing within the Server. Either it has successfully reached completion or it has been cancelled.
	available	The object is available for fetching by an application.
	in_progress	The object is currently being processed by the Server.
	failed	The object failed to be processed by the Server.
interchange	Provides specific information about the EDI interchange. Data displayed for file types is:	
	application_file	Not applicable.
	transmission_file	Not applicable.
	document	Application interchange id and option, partner interchange id and option, segment terminator, element separator, subelement separator, release character, decimal notation character, etc.
functional_group	Provides information about functional groups within the interchange. Data displayed for file types is:	
	application_file	Not applicable.
	transmission_file	Not applicable.
	document	Group type, group control number and more.

Table 3-2 Selection Identifiers (continued)

Value	Description	
acknowledgement	Provides information about functional acknowledgements. Data displayed for file types is:	
	application_file	Not applicable.
	transmission_file	Not applicable.
	document	acknowledgement status, direction, application id, and document count of functional acknowledgement.
document	Provides information about the document type. Data displayed for file types is:	
	application_file	Not applicable.
	transmission_file	Not applicable.
	document	Standard, version, external document type, document control number, common access reference, message version and release, association assigned code, controlling agency.
communications_data	Provides detailed information about transmission files that have passed through a communications service. Data displayed for file types is:	
	application_file	Not applicable.
	transmission_file	OFTP: EERP ack sent, start position, dataset name, date, time, record size, record format. X.400: message router id. Pedi: all Pedi header information.
	document	Not applicable.
history	Provides the detailed status history for documents and transmission files.	

Table 3-2 Selection Identifiers (continued)

Value	Description	
mapper_details	Provides information about the map run associated with a document. Data displayed for file types is:	
	application_file	Not applicable
	transmission_file	Not applicable
	document	Mapper Run ID
other_data	Provides other miscellaneous data about documents and transmission files. Data displayed for file types is:	
	application_file	Not applicable.
	transmission_file	Character count, priority, Digital DEC/EDI store file location.
	document	Character count, segment count, priority, Digital DEC/EDI store file location, X12 application file information, batching information, original document number in the case of a reset document.
all	Provides all available audit information for documents and transmission files and all the tracking and routing information for application files.	

Options

application_name

Select objects by the sending or receiving application. This can be different from the application making the request (the application name specified as a parameter).

Format

UNIX `-application_name=<application_name>`

OpenVMS `/APPLICATION_NAME=<application_name>`

before

Select objects that entered the system before the specified time.

Format

UNIX `-before=<time>`

OpenVMS `/BEFORE=<time>`

Where `<time>` is of the format: `DD-MMM-YYYY:HH:MM:SS.CC.`

business_references

Select objects by the business references associated with them.

If a single business reference is supplied, that is, if there are no brackets and comma-separated values, then each of the five possible business reference slots in turn is checked for the supplied value.

If brackets are used to introduce a list of values, then the required position is determined by where it occurs in the comma separated list of values supplied, and its value will only be tested against that of the associated business reference slot. For example,

```
-business_references="ABC"
```

will cause "ABC" to be checked against each of the 5 business reference slots, whilst

```
-business_references=("","ABC")
```

will cause "ABC" to be checked against only the second business reference slot.

Format

UNIX
```
-business_references=<business_reference>
-business_references=(<business_reference>
        [,<business_reference>... ])
```

OpenVMS
```
/BUSINESS_REFERENCES=<business_reference>
/BUSINESS_REFERENCES=(<business_reference>
         [,<business_reference>... ])
```

current_status

Select objects by status.

Format

UNIX `-current_status[=option]`

OpenVMS `/CURRENT_STATUS[=option]`

Where `option` is one of:

`completed`

`available`

`failed`

`in_progress`

database

Select objects from either the Live database, the Archive database or both databases.

Format

UNIX `-database[=option]`

OpenVMS `/DATABASE[=option]`

Where option is one of:

-live	Select objects from the Live Server audit database. This is the default.
-archive	Select objects from the Archive audit database. By definition these will be of status **PURGEABLE** or **CANCELLED**.
-both	Select objects from both the Live and Archive databases.

direction

Select objects by direction.

Format

UNIX `-direction[=option]`

OpenVMS `/DIRECTION[=option]`

Where `option` is one of:

`inbound`

`outbound`

document_name

Select object by the external document type name, for example, `INVOIC` or `810`.

Format

UNIX `-document_name=<external_document_id>`

OpenVMS `/DOCUMENT_NAME=<external_document_id>`

link_id

Select object by the sending or receiving connection identifier.

Format

UNIX `-link_id=<connection_id>`

OpenVMS `/LINK_ID=<connection_id>`

map_id

Select object by the mapper run ID associated with it.

Format

UNIX `-map_id=<mapper_run_id>`

OpenVMS `/MAP_ID=<mapper_run_id>`

partner_name

Select object by the sending or receiving partner.

Format

UNIX `-partner_name=<partner_name>`

OpenVMS `/PARTNER_NAME=<partner_name>`

object_name

Select objects by type, as specified when the objects were posted or fetched, for example PURCHASE-ORDER or INVOICE.

Format

UNIX `-object_name=<object_name>`

OpenVMS `/OBJECT_NAME=<object_name>`

output_file

The file specification of the file on the client to which output is written. If you omit this option, then all output goes to standard output.

Format

UNIX `-output_file=<file_specification>`

OpenVMS `/OUTPUT_FILE=<file_specification>`

since

Select objects that entered the system after the specified time.

Format

UNIX `-since=<time>`

OpenVMS `/SINCE=<time>`

Where `<time>` is of the format: `DD-MMM-YYYY:HH:MM:SS.CC`.

standard

Select objects by the edi standard used.

Format

UNIX `-standard[=option]`

OpenVMS `/STANDARD[=option]`

Where `option` is one of:

```
edifact
x12
tdcc
odette
tradacoms
```

test_indicator

Select objects by the test mode used.

Format

UNIX `-test_indicator[=option]`

OpenVMS `/TEST_INDICATOR[=option]`

Where `option` is one of:

```
live
mapper_test
translation_test
partner_test
```

tracking_reference

Select objects by tracking reference. The reference can be that returned from the `fetch` or `post` command, or it can be a reference returned from a previous invocation of the `track` command.

Format

UNIX `-tracking_reference=<tracking_reference>`

OpenVMS `/TRACKING_REFERENCE=<tracking_reference>`

type

Indicates which file of files to select.

Format

UNIX `-type[=option]`

OpenVMS `/TYPE[=option]`

Where `option` is one of:

application_file	Select structured application files from the mapper. This is the default for **`type`**.
transmission_file	Select unstructured transmission files directly from the Communications Service on the Server.
document	Select in-house files (Digital DEC/EDI version 1 internal format documents) from the Translation Service on the Server.

version

Select objects by the version of the standard used to process them.

Format

UNIX `-version=<version>`

OpenVMS `/VERSION=<version>`

Examples

1.

UNIX
```
# trade track my-app -direction=inbound \
        -current_status=in_progress tracking_references
```

OpenVMS
```
$ TRADE TRACK my-app /DIRECTION=INBOUND -
        /CURRENT_STATUS=IN_PROGRESS TRACKING_REFERENCES
```

This command gives tracking references for all incoming application files on the Server whose processing is still in progress. In this example, **`type`** defaults to **`application_file`**.

2.

UNIX
```
# trade track my-app -standard=x12 \
     -version=002003 status interchange
```

OpenVMS
```
$ TRADE TRACK my-app /STANDARD=X12 -
     /VERSION=002003 STATUS INTERCHANGE
```

This command reports on all X12 documents of version 002003 and prints out the current status and some of the ISA interchange fields.

Note that **`type=document`** is not necessary in this case because the file type is implied in the selector, `interchange`.

3.

UNIX
```
# trade track my-app -type=transmission_file \
-direction=outbound \
-database=both -output_file=track_output.dat other_data
```

OpenVMS
```
$ TRADE TRACK my-app /TYPE=TRANSMISSION_FILE -
/DIRECTION=OUTBOUND -
/DATABASE=BOTH /OUTPUT_FILE=track_output.dat OTHER_DATA
```

This command reports miscellaneous information (sizes, and so on) on all live and archived transmission files in the Server that are outbound and puts the result in a file called `track_output.dat` in the user's current working directory.

exit

Causes the application client interface to exit.

Format

UNIX `exit`

OpenVMS `EXIT`

Example

UNIX

```
CLIENTEDI> exit
#
```

OpenVMS

```
CLIENTEDI> EXIT
$
```

Chapter 4 C Language Application Programming Interface (API)

The C Language Application Programming Interface (API) defines the C language bindings that must be used by client applications to interface to the Digital DEC/EDI Server. There is a single common interface across all client platforms which means that applications written to run on one platform can easily be made to run on another.

Introducing Digital DEC/EDI API Routines

The API provides a number of routines that an application program can use to post, fetch and track files.

The API routines use item lists to allow the passing of any number of optional parameters. The item lists are constructed using tag-value pairs.

The three routines are introduced below, and described in detail in the following sections.

Posting Files

To post files, the application must use the `DECEDI_POST` routine to submit one or more files to the Digital DEC/EDI Server. The files go either to the mapper or directly to the Communications Service.

The application builds up the request in a series of calls to the `DECEDI_ADD_ITEM_LIST` routine, and then passes the resultant item lists into the `DECEDI_POST` routine. Having completed the call, the application program can free the item lists using the `DECEDI_FREE_ITEM_LIST` routine.

Fetching Files

To fetch files, the application must use the DECEDI_FETCH routine to fetch one or more files from the Digital DEC/EDI Server. These files may have come from the Translation Service or directly from the Communications Service.

The application builds up the request in a series of calls to the DECEDI_ADD_ITEM_LIST routine, and then passes the resultant item lists into the DECEDI_FETCH routine. Having completed the call, the application program can free the item lists using the DECEDI_FREE_ITEM_LIST routine.

Tracking Files

The application can use the DECEDI_TRACK routine to track objects in the Digital DEC/EDI Server. The Application Client returns a list of objects which meet the selection criteria specified in the call to the DECEDI_TRACK routine. The type of information returned about each object is also specified in the call. The returned data is passed back as a dynamic list structure which should be deallocated by using the DECEDI_FREE_TRACK_LIST routine.

The filter criteria and selection of the type of data to return is done by the application making a series of requests to DECEDI_ADD_ITEM_LIST, and then passing the resultant filter and selection item lists to DECEDI_TRACK. Having completed the call, the application can free the item lists by using the DECEDI_FREE_ITEM_LIST routine.

DECEDI_ADD_ITEM_LIST

Adds an optional item to the item list. This is used to build up a request, which is then sent by using DECEDI_POST, DECEDI_FETCH, or DECEDI_TRACK.

C Binding

```
#include    <decedi_api_def.h>

unsigned long int DECEDI_ADD_ITEM_LIST (item_list,
                                        item,
                                        value_type,
                                        value_length,
                                        value,
                                        flags)

decedi_t_item_list            *item_list;
unsigned long int             item;
unsigned long int             value_type;
unsigned long int             value_length;
char                          *value;
unsigned long int             flags;
```

Arguments

item_list

The address of the item list into which the item is to be added. For a new list, set the item list to NULL before this call.

item

The identifier of the item that is being added. Lists of valid identifier types are given in the routine definitions that use the constructed item lists (DECEDI_FETCH, DECEDI_POST, DECEDI_TRACK).

value_type

The type of the value being given. The possible values for this flag

shown in are shown in Table 4-1 *Values for Value Type Flag*.

Table 4-1 Values for Value Type Flag

Value	Description
DECEDI_STRING	Value is a null terminated string.
DECEDI_UINTEGER	Value is an unsigned 32 bit integer.

When using the `DECEDI_UINTEGER` type, it is important to pass the actual value using the pre-defined type `decedi_t_ulong` rather than use the `unsigned long int` type. This is so that the application can remain platform independent.

value_length

The length of the value being given.

value

The address of the value.

flags

Optional flags controlling what is done with the value. The possible values for this flag are shown in Table 4-2 *Values for Add Item Flag*.

Table 4-2 Values for Add Item Flag

Value	Description
DECEDI_READ_ONLY	Data cannot be modified by any subsequent call. This is the default.
DECEDI_WRITEABLE	Data placed on the list can be modified by a subsequent Digital DEC/EDI call.

Description

This routine adds an item to the item list and either copies or references the data associated with it. If it is a new item list then the item list will be allocated.

The item list must be disposed of when finished with by calling `DECEDI_FREE_ITEM_LIST`.

Return Values

DECEDI_BADPARAM	The request was rejected because the call had invalid, or missing, parameters, or the `decedi_t_ulong type` wasn't being used for values passed as `DECEDI_UINTEGER` types.
DECEDI_INSUFVM	There was insufficient virtual memory available to carry out the request.
DECEDI_SUCCESS	The item was successfully added to the list.

Examples

```
#include    <decedi_api_def.h>

unsigned int status;
decedi_t_item_list item_list = (decedi_t_item_list) NULL;
decedi_t_ulong priority = DECEDI_HIGH_PRIORITY;

status = DECEDI_ADD_ITEM_LIST (&item_list,
                     (unsigned long int) DECEDI_ITM_PRIORITY,
                     (unsigned long int) DECEDI_UINTEGER,
                     (unsigned long int) sizeof (priority),
                     (char *) &priority,
                     (unsigned long int) DECEDI_READ_ONLY);
if (status == DECEDI_SUCCESS)
{
    ....
```

DECEDI_FETCH

This enables the caller to fetch one or more files from the Digital DEC/EDI Server. These files may have come directly from the Communications Service, through the translator, or from the mapper.

The type of file being fetched must be specified, and can be one of the following:

- Application_file — files are fetched from the Mapper on the Server. These files are also known as structured files.
- Transmission_file — files are fetched from the Communications Service on the Server, bypassing any translation or mapping. These files are also known as unstructured files.
- Document — files are fetched from the Translation Service on the Server. These files are also known as *ihf* files.

C Binding

```
#include    <decedi_api_def.h>

unsigned long int DECEDI_FETCH (application,
                                overrides,
                                test_indicator,
                                relationship_flag,
                                files)

char                           *application;
decedi_t_item_list             overrides;
unsigned long int              *test_indicator;
unsigned long int              *relationship_flag;
decedi_t_item_list             files;
```

Arguments

application

The name of the client application submitting the request to the Server. This is a null-terminated character string. This argument is used by Digital DEC/EDI as a means of authenticating the request; the client application must be registered as an authorized application on the Server.

Applications are registered by using the CommandCenter Services

Editor. For more information, refer to the *Digital DEC/EDI: User's Guide*.

overrides

An item list specifying the overrides to apply to all files in this request. Build the item list by using one or more calls to `DECEDI_ADD_ITEM_LIST`. Valid item identifier values for the item list are shown in Table 4-3 *DECEDI_FETCH Override Item Identifiers*.

Note that there are 'File overrides' in addition to these command overrides. The File overrides apply to each specific file fetched, whereas command overrides apply to the entire command. See the 'files' argument for further information.

Table 4-3 DECEDI_FETCH Override Item Identifiers

Value	Description
DECEDI_ITM_LINK_ID	Connection ID if bypassing the Mapping and Translation Services. The value associated with this is a character string. This item is mandatory when the file type (DECEDI_ITM_FILE_TYPE) is DECEDI_FILE_TYPE_TRANSMISSION_FILE.
DECEDI_ITM_TEST_INDICATOR	Specifies whether to select test submissions or live submissions. If not specified, then any test indicator value can be matched. The value associated with this item is an integer containing one of the allowed values specified in Table 4-4 *Values for Test Indicator Flag*.

test_indicator

A returned flag indicating the test indicator of the selected files. The valid values for this flag are shown in Table 4-4 *Values for Test Indicator Flag*.

Table 4-4 Values for Test Indicator Flag

Value	Description
DECEDI_LIVE	Files in this request are live business files.
DECEDI_MAPPER_TEST	Files in this request are to be tested through to the mapper but no further.
DECEDI_PARTNER_TEST	Files in this request are test files from the sending partner.

Note that only one test indicator type can be returned per call.

relationship_flag

Not currently used.

files

An item list specifying the files to be fetched in this request. Build the item list using one or more calls to `DECEDI_ADD_ITEM_LIST`. The valid item identifier values for the item list are shown in Table 4-5 *DECEDI_FETCH File Override Identifiers*.

Note that the `DECEDI_ITM_FILE_NAME` must be the first file item for each file.

Table 4-5 DECEDI_FETCH File Override Identifiers

Value	Description
DECEDI_ITM_BUSINESS_REFERENCES	User application references to be added into the document audit trail for later tracking and reporting. This is a string value containing up to five values, each separated by a comma. Empty values are allowed so the user can save references to one or more particular slots in the five available. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies either structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE) or internal format files (DECEDI_FILE_TYPE_DOCUMENT)
DECEDI_ITM_COMMENT	Comment to be added to the mapping audit trail. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-5 DECEDI_FETCH File Override Identifiers (continued)

Value	Description
DECEDI_ITM_DEBUG	Specification of the file on the client to which the mapper debug output is to be written. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_NO_DEBUG	Specifies that there is to be no mapper debug output. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-5 DECEDI_FETCH File Override Identifiers (continued)

Value	Description
DECEDI_ITM_ENTRY_TIME	Return the time, as a text string, that the object first entered the Digital DEC/EDI Server system. For example, for an EDIFACT document, returns the time that the transmission file containing the interchange was received by the Digital DEC/EDI Communications service. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies internal format files (DECEDI_FILE_TYPE_DOCUMENT)
DECEDI_ITM_ERROR_LOG	Specification of the file on the client to which the error log output is to be written. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-5 DECEDI_FETCH File Override Identifiers (continued)

Value	Description
DECEDI_ITM_NO_ERROR_LOG	Specifies that there is to be no error log file. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_FILE_NAME	Specifies the location and name of the file that the client wishes to fetch into. The value associated with this is a string. This must be the first file item for each file. This item is mandatory and must be placed first in the item list.
DECEDI_ITM_FILE_TYPE	Indicates the type of the file to be fetched. The Server handles files differently, depending on the file type. The value associated with this identifier is an unsigned integer. The valid values for this flag are given in Table 4-6 *Values for DECEDI_FETCH File Type Identifier Value*. If not specified then DECEDI_FILE_TYPE_APPLICATION_FILE is assumed.

Table 4-5 DECEDI_FETCH File Override Identifiers (continued)

Value	Description
DECEDI_ITM_IO_DEBUG	Specification of the file on the client to which the mapper I/O debug output is to be written. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_NO_IO_DEBUG	Specifies that there is to be no mapper I/O debug output. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_INTERNAL_REF	Return the Server's internal reference for this object. The value associated with this will be a string that is long enough to hold the returned string. The flag attribute DECEDI_WRITEABLE must be specified when adding this value to the item list.

Table 4-5 DECEDI_FETCH File Override Identifiers (continued)

Value	Description
DECEDI_ITM_INT_CTRL_NUM	Returns the interchange control number associated with the document. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies internal format files (DECEDI_FILE_TYPE_DOCUMENT)
DECEDI_ITM_LOCAL_TEST	Specification of the input file on the Server to use when the mapper is in local test mode. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_NO_LOCAL_TEST	Specifies that the mapper is not to use local test mode. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-5 DECEDI_FETCH File Override Identifiers (continued)

Value	Description
DECEDI_ITM_MATCH_FLAG	Flag indicating how the mapper should match documents. The value associated with this identifier is an unsigned integer. The valid values for this flag are given in Table 4-7 *Values for Match Flag Identifier Value.* If not specified then DECEDI_MATCH_ALL is assumed. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_OBJECT_NAME	Identifies the type of object. It is used by the mapper and for internal-format files (in-house files) to distinguish different objects destined for the same application. The value associated with this identifier is a character string. This item is not applicable for transmission files (DECEDI_ITM_FILE_TYPE_TRANSMISSION_FILE) and is mandatory when the file type is internal format (DECEDI_FILE_TYPE_DOCUMENT).

Table 4-5 DECEDI_FETCH File Override Identifiers (continued)

Value	Description
DECEDI_ITM_OUTPUT_FILE	Specification of the file on the client to which the screen output from the mapper is to be written. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_PARTNER_NAME	Identifies the partner who sent the file or the objects which are derived from it. The value associated with this identifier is a character string. This item is not applicable for transmission files (DECEDI_ITM_FILE_TYPE_TRANSMISSION_FILE) and is mandatory when the file type is internal format (DECEDI_FILE_TYPE_DOCUMENT).

Table 4-5 DECEDI_FETCH File Override Identifiers (continued)

Value	Description
DECEDI_ITM_RESTART_FROM	Document number from which mapping is to restart. The value associated with this is an unsigned integer. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_RETURN_STATUS	Specifies where to place the result of the request for this particular file. The value associated with this will be an unsigned integer. The possible values returned in this field are given in Table 4-8 *Return Values for DECEDI_FETCH File Status*. The flag attribute DECEDI_WRITEABLE must be specified when adding this value to the item list.
DECEDI_ITM_TABLE_NAME	Name of the mapping table to use if the file is going via the mapper. The value associated with this is a character string. This item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE) and is mandatory.

Table 4-5 DECEDI_FETCH File Override Identifiers (continued)

Value	Description
DECEDI_ITM_TIMEOUT	Maximum number of seconds to wait for the file to become available. If not specified a timeout of zero seconds is assumed. The value associated with this will be an unsigned integer.
DECEDI_ITM_TRACKING_REF	User-defined tracking reference associated with this file. The Server will use these values when auditing information about objects, and to respond to requests made by the DECEDI_TRACK call. The value associated with this is a character string.
DECEDI_ITM_NAMED_APPLICATION	Indicates that the application name specified as an argument is to be passed to the mapper. By default, the mapper obtains application names from within the map. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

The File Type Identifier values for DECEDI_FETCH are shown in Table 4-6 *Values for DECEDI_FETCH File Type Identifier Value*.

Table 4-6 Values for DECEDI_FETCH File Type Identifier Value

Value	Description
DECEDI_FILE_TYPE_DOCUMENT	Document files will bypass the mapper and be sent directly to the application. These files are also known as In-house files (DECEDI_FILE_TYPE_IHF).
DECEDI_FILE_TYPE_APPLICATION_FILE	Application_file documents will go through the mapper and their resultant application files will be placed in the required file. These files are also known as Structured files (DECEDI_FILE_TYPE_STRUCTURED).
DECEDI_FILE_TYPE_TRANSMISSION_FILE	Transmission_file documents will bypass the mapper and be placed directly in the required file. These files are also known as Unstructured files (DECEDI_FILE_TYPE_UNSTRUCTURED).

The Match Flag Identifier values for `DECEDI_FETCH` are shown in Table 4-7 *Values for Match Flag Identifier Value*.

Table 4-7 Values for Match Flag Identifier Value

Value	Description
DECEDI_MATCH_ALL	Match all documents found.
DECEDI_MATCH_FIRST	Match first document found.

The file status Return values for `DECEDI_FETCH` are shown in

Table 4-8 *Return Values for DECEDI_FETCH File Status*.

Table 4-8 Return Values for DECEDI_FETCH File Status

Value	Description
DECEDI_RETURN_CREATE_FAILED	One of the output files specified in the request could not be created.
DECEDI_RETURN_OPEN_FAILED	One of the input files specified in the request could not be opened.
DECEDI_RETURN_MAP_FAILED	File was not fetched because the mapping failed.
DECEDI_RETURN_MAP_NOOUTPUT	The requested file map ran successfully but produced no output. Other files in the request will be processed.
DECEDI_RETURN_NOMORE	No files were found matching the specification.
DECEDI_RETURN_MAP_PARTIAL	A file in the request was only partially mapped. Details of what the mapper found can be obtained from the mapper output files, if specified. Other files in the request will be processed.
DECEDI_RETURN_SUCCESS	File was successfully fetched.

Description

This routine issues a request to the Server to fetch one or more files from the Server. It then waits for the Server to fulfill the request and return the results.

Construct the request using one or more calls to `DECEDI_ADD_ITEM_LIST`. Once this call has been completed, use `DECEDI_FREE_ITEM_LIST` to release the associated item lists.

Return Value

DECEDI_BADITMLST	The request was rejected because the call had invalid, or missing item lists.
DECEDI_BADPARAM	The request was rejected because the call had invalid, or missing parameters.
DECEDI_INSUFVM	Insufficient virtual memory available to complete request.
DECEDI_INTERROR	Internal error.
DECEDI_NOCLIENTLIC	No active client license present.
DECEDI_NOEDISYS	EDI Server not running.
DECEDI_NOTAUTH	Not authorized to access the EDI Server.
DECEDI_OPENINPERR	Could not open one of the input files.
DECEDI_OPENOUTERR	Could not open one of the output files.
DECEDI_ORBBADNODE	Bad Server selection node.
DECEDI_ORBCOMMFAIL	Communications to Server failed.
DECEDI_ORBSRVDIED	Server process died.
DECEDI_ORBSRVNOTFND	No Server found.
DECEDI_ORBTIMEOUT	Server timeout.
DECEDI_SRVERROR	Server error.
DECEDI_SUCCESS	The request was successfully processed by the Server.
DECEDI_WARNING	Not all the request was successfully processed. Check the individual file statuses.
DECEDI_ZEROMATCH	No records matched.

Examples

```
#include    <decedi_api_def.h>

unsigned long int status;
decedi_t_item_list item_list = (decedi_t_item_list) NULL;
decedi_t_ulong file_status;
unsigned long int test_indicator;
```

```
unsigned long int relationship;


status = DECEDI_ADD_ITEM_LIST (&item_list,
               (unsigned long int) DECEDI_ITM_FILE_NAME,
               (unsigned long int) DECEDI_STRING,
               (unsigned long int) strlen("test_file.dat"),
               (char *) "test_file.dat",
               (unsigned long int) DECEDI_READ_ONLY);
if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_ADD_ITEM_LIST (&item_list,
               (unsigned long int) DECEDI_ITM_PARTNER_NAME,
               (unsigned long int) DECEDI_STRING,
              (unsigned long int) strlen("THEIR-APPLICATION"),
               (char *) "THEIR-APPLICATION",
               (unsigned long int) DECEDI_READ_ONLY);

if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_ADD_ITEM_LIST (&item_list,
               (unsigned long int) DECEDI_ITM_TRACKING_REF,
               (unsigned long int) DECEDI_STRING,
               (unsigned long int) strlen("INVOICE #1234"),
               (char *) "INVOICE #1234",
               (unsigned long int) DECEDI_READ_ONLY);

if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_ADD_ITEM_LIST (&item_list,
               (unsigned int) DECEDI_ITM_TABLE_NAME,
               (unsigned int) DECEDI_STRING,
               (unsigned int) strlen("MY_MAP"),
               (char *) "MY_MAP",
               (unsigned long int) DECEDI_READ_ONLY);

if (status != DECEDI_SUCCESS)
{
....
```

```
}

status = DECEDI_ADD_ITEM_LIST (&item_list,
               (unsigned long int) DECEDI_ITM_RETURN_STATUS,
               (unsigned long int) DECEDI_UINTEGER,
               (unsigned long int) sizeof(file_status),
               (char *) &file_status,
               (unsigned long int) DECEDI_WRITEABLE);

if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_FETCH ("MY-APPLICATION", /* Application name
*/
             (decedi_t_item_list) NULL, /* Overrides        */
             &test_indicator,           /* Test Indicator   */
             &relationship,             /* Relationship     */
             item_list);                /* Files to fetch   */

status = DECEDI_FREE_ITEM_LIST(&item_list);
```

DECEDI_FREE_ITEM_LIST

DECEDI_FREE_ITEM_LIST enables the caller to free all memory associated with an item list and its associated values.

C Binding

```
#include    <decedi_api_def.h>

unsigned long int DECEDI_FREE_ITEM_LIST ( item_list )

decedi_t_item_list *item_list;
```

Arguments

item_list

The item list to remove from memory. You create the item list using DECEDI_ADD_ITEM_LIST.

Description

This routine deallocates all memory associated with an item list.

Return Values

DECEDI_SUCCESS	The list was successfully released.
DECEDI_BADPARAM	The request was rejected because the call had invalid, or missing, parameters.

Examples

```
#include    <decedi_api_def.h>

unsigned long int status;
decedi_t_item_list *item_list;

status = DECEDI_FREE_ITEM_LIST(&item_list);
                                    /* item list to be freed*/
```

DECEDI_FREE_TRACK_LIST

DECEDI_FREE_TRACK_LIST enables the caller to free all memory associated with a dynamic list of objects returned from a call to DECEDI_TRACK.

C Binding

```
#include    <decedi_api_def.h>

unsigned long int DECEDI_FREE_TRACK_LIST ( results )

decedi_t_track_list *results;
```

Arguments

results

Results structure to be freed.

Description

This routine frees all memory associated with a results list that was previously returned by DECEDI_TRACK.

Return Values

DECEDI_SUCCESS	The list was successfully released.
DECEDI_BADPARAM	The request was rejected because the call had invalid, or missing, parameters.

Examples

```
#include    <decedi_api_def.h>

unsigned long int status;
decedi_t_track_list *results;

status = DECEDI_FREE_TRACK_LIST(&results);
                                    /* results list to be freed*/
```

DECEDI_POST

DECEDI_POST enables the caller to submit one or more files into the Digital DEC/EDI Server. These files may be posted directly to the Communications Service, through the Translator or via the Mapper.

The type of file being posted must be specified, and can be one of the following:

- Application_file — files are posted to the Mapper on the Server. These files are also known as structured files.
- Transmission_file — files are posted to the Communications Service on the Server, bypassing any translation or mapping. These files are also known as unstructured files.
- Document — files are posted to the Translation Service on the Server. These files are also known as *ihf* files.

C Binding

```
#include    <decedi_api_def.h>

unsigned long int DECEDI_POST ( application,
                                overrides,
                                files )

char                    *application;
decedi_t_item_list      overrides;
decedi_t_item_list      files;
```

Arguments

application

The name of the client application submitting the request to the Server. This is a null-terminated character string. This argument is used by Digital DEC/EDI as a means of authenticating the request; the client application must be registered as an authorized application on the Server. Applications are registered by using the CommandCenter Management Services Editor. For more information, refer to the *Digital DEC/EDI: User's Guide*.

overrides

An item list specifying the overrides to apply to all files in this request. Build the item list using one or more calls to DECEDI_ADD_ITEM_LIST. Table 4-9 *Override Identifiers* gives the item identifier values allowed in the item list.

Note that there are *file overrides* in addition to these command overrides. The file overrides apply to each specific file posted whereas command overrides apply to the entire command. See the

files argument for further information.

Table 4-9 Override Identifiers

Value	Description

DECEDI_ITM_CONNECTION_DATA	This provides information to the communications connection which overrides the defaults for that connection. This field is specific to the type of gateway referenced by the connection id. Please refer to the CommandCenter Trading Partner Editor's on-line help for types that support this feature, and what format the data should be in.
	
	`<initiator_oftp_id>\`
	`<virtual_file_name>\`
	`<fixed_record_length>\`
	`<user_data>\`
	`<originator_oftp_id>`

Table 4-9 Override Identifiers (continued)

Value	Description
DECEDI_ITM_LINK_ID	Connection ID, if bypassing the mapper and translator. The value associated with this is a character string. This item is mandatory when the file type (DECEDI_ITM_FILE_TYPE) is DECEDI_FILE_TYPE_TRANSMISSION_FILE.
DECEDI_ITM_PRIORITY	Specifies the priority the Server is to give to these files. The value associated with this is an unsigned integer containing one of the allowed values specified in Table 4-11 *Values for DECEDI_POST Priority Flag*. If not specified then DECEDI_NORMAL_PRIORITY is assumed.
DECEDI_ITM_TEST_INDICATOR	Specifies whether to treat the files in this request as test submissions or live submissions. The value associated with this is an unsigned integer containing one of the allowed values specified in Table 4-10 *Values for DECEDI_POST Test Indicator Flag*. If not specified then DECEDI_LIVE is assumed.

The possible values for the `DECEDI_POST` Test Indicator flag are

given in Table 4-10 *Values for DECEDI_POST Test Indicator Flag*.

Table 4-10 Values for DECEDI_POST Test Indicator Flag

Value	Description
DECEDI_LIVE	Files in this request are live business files.
DECEDI_MAPPER_TEST	Files in this request are to be tested through to the mapper but no further.
DECEDI_PARTNER_TEST	Files in this request are to be tested through all services to the destination partner, and where possible, marked as being a test by the relevant services.
DECEDI_TRANSLATION_TEST	Files in this request are to be tested through to the Translation Service but no further.

The possible values for the `DECEDI_POST` Priority flag are given in Table 4-11 *Values for DECEDI_POST Priority Flag*.

Table 4-11 Values for DECEDI_POST Priority Flag

Value	Description
DECEDI_HIGH_PRIORITY	Files in this request are to be treated as high priority objects. These objects are processed by the translation and Communications Services immediately, rather than at scheduled intervals.
DECEDI_NORMAL_PRIORITY	Files in this request are to be treated as for normal processing.

files

Item list specifying the files to be posted in this request. Build the item list using one or more calls to `DECEDI_ADD_ITEM_LIST`.

The item identifier values allowed in the item list for this request are given in Table 4-12 *DECEDI_POST File Override Identifiers*.

Note that the `DECEDI_ITM_FILE_NAME` must be the first file item for each file.

Table 4-12 DECEDI_POST File Override Identifiers

Value	Description
DECEDI_ITM_BUSINESS_REFERENCES	User application references to be added into the document audit trail for later tracking and reporting. This is a string value containing up to five values, each separated by a comma. Empty values are allowed so the user can save references to one or more particular slots in the five available. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies either structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE) or internal format files (DECEDI_FILE_TYPE_DOCUMENT)
DECEDI_ITM_COMMENT	Comment to be added to the mapping audit trail. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-12 DECEDI_POST File Override Identifiers (continued)

Value	Description
DECEDI_ITM_DEBUG	Specification of the file on the client to which the mapper debug output is to be written. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_NO_DEBUG	Specifies that there is to be no mapper debug output. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_ERROR_LOG	Specification of the file on the client to which the error log output is to be written. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-12 DECEDI_POST File Override Identifiers (continued)

Value	Description
DECEDI_ITM_NO_ERROR_LOG	Specifies that there is to be no error log file. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_FILE_NAME	Specifies the location and name of the file to post. The value associated with this is a string. This must be the first file item for each file. This item is mandatory and must be placed first in the item list.
DECEDI_ITM_FILE_TYPE	Indicates the type of the file that is being posted. The Server handles files differently, depending on the type of the file. The value associated with this identifier is an unsigned integer. The valid values for this flag are given in Table 4-13 *Values for File Type Flag*. If not specified then DECEDI_FILE_TYPE_APPLICATION_FILE is assumed.

Table 4-12 DECEDI_POST File Override Identifiers (continued)

Value	Description
DECEDI_ITM_INTERNAL_REF	Return the Server's internal reference for this object. The value associated with this will be a string that is long enough to hold the returned string. The flag attribute DECEDI_WRITEABLE must be specified when adding this value to the item list.
DECEDI_ITM_IO_DEBUG	Specification of the file on the client to which the mapper I/O debug output is to be written. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_NO_IO_DEBUG	Specifies that there is to be no mapper debug output. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-12 DECEDI_POST File Override Identifiers (continued)

Value	Description
DECEDI_ITM_LOCAL_TEST	Specification of the input file on the Server to use when the mapper is in local test mode. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_NO_LOCAL_TEST	Specifies that the mapper is not to use local test mode. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).
DECEDI_ITM_NAMED_APPLICATION	Indicates that the application name specified as an argument is to be passed to the mapper. By default, the mapper obtains application names from within the map. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-12 DECEDI_POST File Override Identifiers (continued)

Value	Description
DECEDI_ITM_OBJECT_NAME	Identifies the type of object. It is used by the mapper to distinguish different objects sent from the same application. The value associated with this identifier is a character string. This item is not applicable for transmission files (DECEDI_ITM_FILE_TYPE_TRANSMISSION_FILE) and is mandatory when the file type is internal format (DECEDI_FILE_TYPE_DOCUMENT).
DECEDI_ITM_OUTPUT_FILE	Specification of the output file to use on the Client. The value associated with this is a character string. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-12 DECEDI_POST File Override Identifiers (continued)

Value	Description
DECEDI_ITM_PARTNER_NAME	Identifies the partner which will receive the file. The value associated with this identifier is a character string. This item is not applicable for transmission files (DECEDI_ITM_FILE_TYPE_TRANSMISSION_FILE) and is mandatory when the file type is internal format (DECEDI_FILE_TYPE_DOCUMENT).
DECEDI_ITM_RESTART_FROM	Document number from which mapping is to be restarted. The value associated with this is an unsigned integer. This optional item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE).

Table 4-12 DECEDI_POST File Override Identifiers (continued)

Value	Description
DECEDI_ITM_RETURN_STATUS	Specifies where to put the result of the request for this particular file. The value associated with this will be an unsigned integer. The possible values returned in this field are given in Table 4-14 *Return Values for DECEDI_POST File Status*. The flag attribute DECEDI_WRITEABLE must be specified when adding this value to the item list.
DECEDI_ITM_TABLE_NAME	Name of mapping table to use if using the mapper. The value associated with this is a character string. This item is applicable only when DECEDI_ITM_FILE_TYPE specifies structured application files (DECEDI_FILE_TYPE_APPLICATION_FILE) and is mandatory.

Table 4-12 DECEDI_POST File Override Identifiers (continued)

Value	Description
DECEDI_ITM_TRACKING_REF	User-defined tracking reference to be associated with this file. The Server will use these values when auditing information about objects, and to respond to requests made by the DECEDI_TRACK call. The value associated with this is a character string. This item is mandatory when DECEDI_ITM_FILE_TYPE specifies internal format documents (DECEDI_FILE_TYPE_DOCUMENT).

The possible values for the File Type flag are given in Table 4-13 *Values for File Type Flag*.

Table 4-13 Values for File Type Flag

Value	Description
DECEDI_FILE_TYPE_DOCUMENT	Document files will bypass the mapper and go to the Translation Service. These files are also known as In-house files (DECEDI_FILE_TYPE_IHF).
DECEDI_FILE_TYPE_APPLICATION_FILE	Application_file documents go through the mapper, and their resultant documents are passed to the relevant Translation Service. These files are also known as Structured files (DECEDI_FILE_TYPE_STRUCTURED).
DECEDI_FILE_TYPE_TRANSMISSION_FILE	Unstructured documents bypass the mapper and Translation Service and go directly to the Communications Service. These files are also known as Unstructured files (DECEDI_FILE_TYPE_UNSTRUCTURED).

The possible Return values for the File Status flag are given in Table 4-14 *Return Values for DECEDI_POST File Status*.

Table 4-14 Return Values for DECEDI_POST File Status

Value	Description
DECEDI_RETURN_OPEN_FAILED	One of the files specified in the request could not be opened.
DECEDI_RETURN_MAP_FAILED	File was not posted because the mapping failed.
DECEDI_RETURN_MAP_OUTPUT	The file requests map ran successfully but produced no output for this file. Other files in the request will be processed.
DECEDI_RETURN_MAP_PARTIAL	A file request was only partially mapped. Details of what the mapper found can be obtained from the mapper output files if specified. Other files in the request will be processed.
DECEDI_RETURN_SUCCESS	File was successfully posted.

Description

This routine issues a request to the Server to post one or more files into the Server. It then waits for the Server to fulfill the request and return the results to the caller.

The request will have previously been constructed using one or more calls to `DECEDI_ADD_ITEM_LIST`. Once this call has been completed the associated item lists can be released using `DECEDI_FREE_ITEM_LIST`.

Return Values

DECEDI_BADITMLST	The request was rejected because the call had invalid, or missing item lists.
DECEDI_BADPARAM	The request was rejected because the call had invalid, or missing parameters.
DECEDI_INSUFVM	Insufficient virtual memory available to complete request.
DECEDI_INTERROR	Internal error.
DECEDI_NOCLIENTLIC	No active client license present.
DECEDI_NOEDISYS	EDI Server not running.
DECEDI_NOTAUTH	Not authorized to access the EDI Server.
DECEDI_OPENINPERR	Could not open one of the input files.
DECEDI_OPENOUTERR	Could not open one of the output files.
DECEDI_ORBBADNODE	Bad Server selection node.
DECEDI_ORBCOMMFAIL	Communications to Server failed.
DECEDI_ORBSRVDIED	Server process died.
DECEDI_ORBSRVNOTFND	No Server found.
DECEDI_ORBTIMEOUT	Server timeout.
DECEDI_SRVERROR	Server error.
DECEDI_SUCCESS	The request was successfully processed by the Server.
DECEDI_WARNING	Not all the request was successfully processed. Check the individual file statuses.
DECEDI_ZEROMATCH	No records matched.

Examples

```
#include   <decedi_api_def.h>

unsigned long int status;
```

```
decedi_t_item_list item_list = (decedi_t_item_list) NULL;
decedi_t_ulong file_status;
unsigned long int test_indicator;
unsigned long int relationship;


status = DECEDI_ADD_ITEM_LIST (&item_list,
               (unsigned long int) DECEDI_ITM_FILE_NAME,
               (unsigned long int) DECEDI_STRING,
               (unsigned long int)
                 strlen("/usr/users/happyjack/test_file.dat"),
               (char *) "/usr/users/happyjack/test_file.dat",
               (unsigned long int) DECEDI_READ_ONLY);
if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_ADD_ITEM_LIST (&item_list,
               (unsigned long int) DECEDI_ITM_PARTNER_NAME,
               (unsigned long int) DECEDI_STRING,
              (unsigned long int) strlen("THEIR-APPLICATION"),
               (char *) "THEIR-APPLICATION",
               (unsigned long int) DECEDI_READ_ONLY);

if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_ADD_ITEM_LIST (&item_list,
               (unsigned long int) DECEDI_ITM_TRACKING_REF,
               (unsigned long int) DECEDI_STRING,
               (unsigned long int) strlen("INVOICE #1234"),
               (char *) "INVOICE #1234",
               (unsigned long int) DECEDI_READ_ONLY);

if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_ADD_ITEM_LIST (&item_list,
               (unsigned long int) DECEDI_ITM_TABLE_NAME,
               (unsigned long int) DECEDI_STRING,
               (unsigned long int) strlen("my_map"),
               (char *) "my_map",
               (unsigned long int) DECEDI_READ_ONLY);
```

```
if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_ADD_ITEM_LIST (&item_list,
              (unsigned long int) DECEDI_ITM_RETURN_STATUS,
              (unsigned long int) DECEDI_UINTEGER,
              (unsigned long int) sizeof(file_status),
              (char *) &file_status,
              (unsigned long int) DECEDI_WRITEABLE);

if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_POST ("MY-APPLICATION",  /* Application name
*/
             (decedi_t_item_list) NULL, /* Overrides        */
             item_list);               /* Files to post    */

status = DECEDI_FREE_ITEM_LIST(&item_list);
```

DECEDI_TRACK

This enables the caller to track objects within the Digital DEC/EDI Server. It can provide lists of objects that match certain selection criteria, and it can return different types of information for the objects that meet the selection criteria, for example routing information or object status. Refer to Chapter 5 *Tracking Facilities* for information on tracking.

The Track command can track only one kind of object at a time. The type of object being tracked must be specified and may be one of the following:

- Application_file — files that have been posted or fetched via the Mapper on the Server. These files are also known as Structured files.
- Transmission_file — files that have been sent or received via the Communications Service on the Server, or are waiting to be sent via the Communications Service. These files are also known as Unstructured files.
- Document — files that have gone through or are waiting to go through the Translation Service on the Server. This also includes documents that are involved in Application-to-application agreements. These files are also known as *ihf* files.

C Binding

```
#include    <decedi_api_def.h>

unsigned long int DECEDI_TRACK (application,
                                filters,
                                match_count,
                                selections,
                                results)

char                    *application;
decedi_t_item_list      filters;
unsigned long int       *match_count;
decedi_t_item_list      selections;
decedi_t_track_list     *results;
```

Arguments

application

The name of the client application submitting the request to the

Server. This is a null-terminated character string. This argument is used by Digital DEC/EDI as a means of authenticating the request; the client application must be registered as an authorized application on the Server. Applications are registered by using the CommandCenter Management Services editor. For more information, refer to the *Digital DEC/EDI: User's Guide*.

filters

An item list specifying the selection criteria for the objects that are to be tracked. Build up the item list using one or more calls to `DECEDI_ADD_ITEM_LIST`. The valid item identifier values allowed in the item list for this request are given in Table 4-15 *DECEDI_TRACK Object Selection Criteria*. If no filters are specified then all objects are selected.

Table 4-15 DECEDI_TRACK Object Selection Criteria

Value	Description
DECEDI_ITM_APPLICATION_NAME	Select objects by the sending or receiving application. The value associated with this is a string.
DECEDI_ITM_BEFORE	Select objects that entered the Digital DEC/EDI system before the specified time. The value associated with this is a string of the format `DD-MMM-YYYY HH:MM:SS.CC`. If you do not supply a value for day, month or year, the current value will be used as the default.
DECEDI_ITM_BUSINESS_REFERENCES	Select document objects by the business references that have been applied to them. This is a string value containing up to five values, each separated by a comma. Empty values are allowed so that each of the five available business reference slots can be tested for a match against specific values. To search irrespective of slot position, use DECEDI_ITM_SINGLE_BUSINESS_REFERENCE instead.
DECEDI_ITM_CURRENT_STATUS	Select objects by object status. The value associated with this will be an unsigned integer that can have any of the values specified in Table 4-16 *DECEDI_TRACK Object Status Values.*

Table 4-15 DECEDI_TRACK Object Selection Criteria (continued)

Value	Description
DECEDI_ITM_DATABASE	Select objects from either the Live database, Archive database or both. The value associated with this will be an unsigned integer that can have one of the values specified in Table 4-18 *DECEDI_TRACK Object Database Values*.
DECEDI_ITM_DIRECTION	Select objects by direction; that is, whether they are being sent or received. The value associated with this will be an unsigned integer which can have one of the valid values specified in Table 4-17 *DECEDI_TRACK Object Direction Values*.
DECEDI_ITM_DOCUMENT_NAME	Select objects by external document type. The value associated with this is a string (for example, “INVOIC” or “810”).
DECEDI_ITM_FILE_TYPE	Select objects by file type. The valid values for this flag are given in Table 4-13 *Values for File Type Flag*. If not specified then DECEDI_FILE_TYPE_APPLICATION_FILE is assumed.
DECEDI_ITM_LINK_ID	Select objects by connection ID, if bypassing the mapper and Translation Service. The value associated with this is a character string.
DECEDI_ITM_MPR_RUN_ID	Select document objects by mapper run ID. The value associated with this is a character string.

Table 4-15 DECEDI_TRACK Object Selection Criteria (continued)

Value	Description
DECEDI_ITM_OBJECT_NAME	Select objects by object type. The value associated with this is a string.
DECEDI_ITM_PARTNER_NAME	Select objects by the sending or receiving partner. The value associated with this is a string.
DECEDI_ITM_SINCE	Select objects that entered the Digital DEC/EDI system after the specified time. The value associated with this is a string of the format `DD-MMM-YYYY HH:MM:SS.CC`. If you do not supply a value for day, month or year, the current value will be used as the default.
DECEDI_ITM_SINGLE_BUSINESS_REFERENCE	Select objects by the business references that have been applied to them. This searches each of the five business reference slots for a match. To search specific slot positions, use DECEDI_ITM_BUSINESS_REFERENCES.
DECEDI_ITM_STANDARD	Select object by the standard used to process it. The value associated with this will be an unsigned integer that can have one of the values specified in Table 4-19 *DECEDI_TRACK Object Standard Values*.

Table 4-15 DECEDI_TRACK Object Selection Criteria (continued)

Value	Description
DECEDI_ITM_TEST_INDICATOR	Select object by its test indicator. The value associated with this will be an unsigned integer that can have one of the values specified in Table 4-20 *DECEDI_TRACK Object Test Indicator Values*.
DECEDI_ITM_TRACKING_REFERENCE	Select objects by tracking reference. The value associated with this is a string.
DECEDI_ITM_VERSION	Select object by the version of the standard used to process it. The value associated with this is a string (for example, “003002” or “901”).

The possible values for Object Status are given in Table 4-16 *DECEDI_TRACK Object Status Values*.

Table 4-16 DECEDI_TRACK Object Status Values

Value	Description
DECEDI_AVAILABLE	The object is available for fetching by an application.
DECEDI_COMPLETED	The object has completed processing within the Server. Either it has successfully reached completion or has been cancelled.
DECEDI_FAILED	The object failed to be processed through the Server.
DECEDI_IN_PROGRESS	The object is currently being processed through the Server.

The possible values for Object Direction are given in Table 4-17 *DECEDI_TRACK Object Direction Values*.

Table 4-17 DECEDI_TRACK Object Direction Values

Value	Description
DECEDI_INBOUND	The object is being received by the Server.
DECEDI_OUTBOUND	The object is being sent by the Server.

The possible values for Object Database are given in Table 4-18 *DECEDI_TRACK Object Database Values*.

Table 4-18 DECEDI_TRACK Object Database Values

Value	Description
LIVE (D)	Select objects from the Live audit database.
ARCHIVE (D)	Select objects from the Archive audit database. By definition these will be of status **PURGEABLE** or **CANCELLED**.
BOTH (D)	Select objects from both the Live and Archive databases.

The possible values for Object Standard are given in Table 4-19 *DECEDI_TRACK Object Standard Values*

Table 4-19 DECEDI_TRACK Object Standard Values

Value
EDIFACT
X12
TDCC
ODETTE
TRADACOMS

The possible values for Object Test Indicator are given in Table 4-20

DECEDI_TRACK Object Test Indicator Values.

Table 4-20 DECEDI_TRACK Object Test Indicator Values

Value	Description
LIVE	Live files. This is the default.
MAPPER_TEST	Files tested through to the Mapping Service.
TRANSLATION_TEST	Files tested through to the Translation Service.
PARTNER_TEST	Files tested through all services to the trading partner.

match count

This parameter returns the number of matches found.

selections

An item list specifying the type of data to return to the caller. The item identifier values allowed in the item list for this request are given in Table 4-21 *Selection Identifiers*. If not specified then only the number of objects in the system is returned.

No value is associated with any of the identifiers in Table 4-21 *Selection Identifiers*.

Table 4-21 Selection Identifiers

Value	Description
DECEDI_SEL_EDI_CONTROL_REFS	Provide all EDI envelope control references associated with the object.
DECEDI_SEL_MAPPER	Provide all data associated with the mapper when DECEDI_ITM_FILE_TYPE is DECEDI_FILE_TYPE_DOCUMENT.
DECEDI_SEL_BUSINESS_REFERENCES	Provide all business references associated with the object when DECEDI_ITM_FILE_TYPE is DECEDI_FILE_TYPE_DOCUMENT.
DECEDI_SEL_ROUTING	Provide all data associated with the routing of this object through the Digital DEC/EDI system such as application name, partner name, document type.
DECEDI_SEL_STATUS	Provide status information associated with the object.
DECEDI_SEL_TRACKING_REFS	Provide all tracking references associated with the object.
DECEDI_SEL_INTERCHANGE	Provide interchange level information when DECEDI_ITM_FILE_TYPE is DECEDI_FILE_TYPE_DOCUMENT.
DECEDI_SEL_FUNCTIONAL_GROUP	Provide functional group information when DECEDI_ITM_FILE_TYPE is DECEDI_FILE_TYPE_DOCUMENT.

Table 4-21 Selection Identifiers (continued)

Value	Description
DECEDI_SEL_ACKNOWLEDGEMENT	Provide functional acknowledgement information when DECEDI_ITM_FILE_TYPE is DECEDI_FILE_TYPE_DOCUMENT.
DECEDI_SEL_DOCUMENT	Provide document level information when DECEDI_ITM_FILE_TYPE is DECEDI_FILE_TYPE_DOCUMENT.
DECEDI_SEL_COMMS_DATA	Provide full communications audit record information when DECEDI_ITM_FILE_TYPE is DECEDI_FILE_TYPE_TRANSMISSION_FILE.

Table 4-21 Selection Identifiers (continued)

Value	Description
DECEDI_SEL_HISTORY	Provide full history information (status transitions) when DECEDI_ITM_FILE_TYPE is DECEDI FILE_TYPE_TRANSMISSION_FILE, or DECEDI_FILE_TYPE_DOCUMENT.
DECEDI_SEL_OTHER_DATA	Provide X12 application file information, size information, file location, priority, test indication, batch file information and original document count if document is resent, when DECEDI_ITM_FILE_TYPE is DECEDI_FILE_TYPE_DOCUMENT. Also when DECEDI_ITM_FILE_TYPE is DECEDI_FILE_TYPE_TRANSMISSION_FILE, provide size, priority and file location information.
DECEDI_SEL_ALL	Select all of the above information.

results

A dynamic list structure that is returned on completion with the object data requested. Deallocate the list's memory using `DECEDI_FREE_TRACK_LIST`.

Description

This routine issues a request to the Server to fetch details of one or more objects in the Digital DEC/EDI system. It returns data for only those objects that match the selection criteria.

Build this request using one or more calls to DECEDI_ADD_ITEM_LIST. Once this call has been completed, you can release the associated item lists using DECEDI_FREE_ITEM_LIST.

The results, if any have been requested, are placed in a dynamically allocated list. Once all the results have been processed, free this list using DECEDI_FREE_TRACK_LIST.

Return Values

DECEDI_BADITMLST	The request was rejected as the call had invalid, or missing item lists.
DECEDI_BADPARAM	The request was rejected as the call had invalid, or missing parameters.
DECEDI_INSUFVM	Insufficient virtual memory available to complete request.
DECEDI_INTERROR	Internal error.
DECEDI_NOCLIENTLIC	No active client license was found.
DECEDI_ORBBADNODE	Bad Server selection node.
DECEDI_ORBCOMMFAIL	Communications to Server failed.
DECEDI_ORBDOWN	ACA Services not running.
DECEDI_ORBERROR	ACA Services failure.
DECEDI_ORBSRVDIED	Server process died.
DECEDI_ORBSRVNOTFND	No Server found.
DECEDI_ORBTIMEOUT	Server timeout.
DECEDI_ORBVERMISM	ACA Services version mismatch.
DECEDI_SUCCESS	The request was successfully processed by the Server.

Examples

```
#include    <decedi_api_def.h>

unsigned long int status;
decedi_t_item_list filters = (decedi_t_item_list) NULL;
decedi_t_item_list selections = (decedi_t_item_list) NULL;
```

```
decedi_t_track_list *results;
decedi_t_ulong status_value = DECEDI_AVAILABLE;
decedi_t_ulong *match_count

status = DECEDI_ADD_ITEM_LIST (&filters,
              (unsigned long int) DECEDI_ITM_APPLICATION_NAME,
               (unsigned long int) DECEDI_STRING,
               (unsigned long int) strlen("MY-APPLICATION"),
               (char *) "MY-APPLICATION",
               (unsigned long int) DECEDI_READ_ONLY);
if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_ADD_ITEM_LIST (&filters,
               (unsigned long int) DECEDI_ITM_CURRENT_STATUS,
               (unsigned long int) DECEDI_UINTEGER,
               (unsigned long int) sizeof(status_value),
               (char *) &status_value,
               (unsigned long int) DECEDI_READ_ONLY);

if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_ADD_ITEM_LIST (&selections,
            (unsigned long int) DECEDI_SEL_TRACKING_REFERENCE,
             (unsigned long int) DECEDI_UINTEGER,
             (unsigned long int) DECEDI_READ_ONLY,
             (char *) NULL,
             (unsigned long int) DECEDI_READ_ONLY);

if (status != DECEDI_SUCCESS)
{
....
}

status = DECEDI_TRACK ("MY-APPLICATION",/* Application name
*/
                       filters,        /* Filters           */
                       &match_count,   /* Number of matches */
                       selections,     /* Data required     */
                       &results        /* Returned results  */
                        );

status = DECEDI_FREE_ITEM_LIST (&selections);
```

```
status = DECEDI_FREE_ITEM_LIST (&filters);

status = DECEDI_FREE_TRACK_LIST (&results);
```

Header Files

The Application Client contains the following header files:

- `decedi_api_def.h`

  This file provides the entry points for the API.
- `decedi_api_common.h`

  This file provides API common values and structures.
- `decedi_api_msgs.h`

  This file provides API return values.
- `decedi_platforms.h`

  This file provides API platform specific declarations.

You only need to include the header file `decedi_api_def.h`, which automatically includes the other header files.

Compiling and Linking on UNIX Platforms

Compiling

If you are using the Application Client's API on a UNIX platform (Tru64 UNIX AXP, Sun Solaris, or HP-UX) use an ANSI C compiler and compile your application as you would any other application, for example:

```
# cc -c test-appl-a.c
```

Linking

The Application Client is supplied with the following shareable API image on Tru64 UNIX. It is installed into `/usr/shlib`:

- `llibdecediapi.so`

You must link against this shareable image, for example:

```
# cc -o test-appl-a test-appl-a.o -ldecediapi
```

Compiling and Linking on OpenVMS

Compiling

If you are using the Application Client's API on OpenVMS, we recommend you use the DEC C compiler and compile your application as you would any other application, for example:

```
$ CC test-appl-a.c
```

Linking

The Application Client is supplied with the following shareable API image on OpenVMS. It is installed into SYS$SHARE:

- SYS$SHARE:DECEDI$APPSHR_V2.EXE

You must link against this shareable image, for example:

```
$ LINK /EXE=test-appl-a test-appl -
  SYS$SHARE:DECEDI$APPSHR_V2.EXE/SHARE
```

Refer to the *Digital DEC/EDI: Installation* for corresponding information on the other supported client platforms.

Part II Digital DEC/EDI Reference

This part of the book contains reference information on:

- Digital DEC/EDI Command Line Interface (CLI)
- Digital DEC/EDI Application Programming Interface (API)
- Error messages and how to react to them
- Tracking files through the system
- Debugging Mapping Tables
- How to address any problems you may encounter with application development and integration.

Chapter 5 Tracking Facilities

This chapter describes how to run the Digital DEC/EDI tracking facilities, and how the Mapper collects audit and history data.

There are two ways in which you can track document transmissions in Digital DEC/EDI:

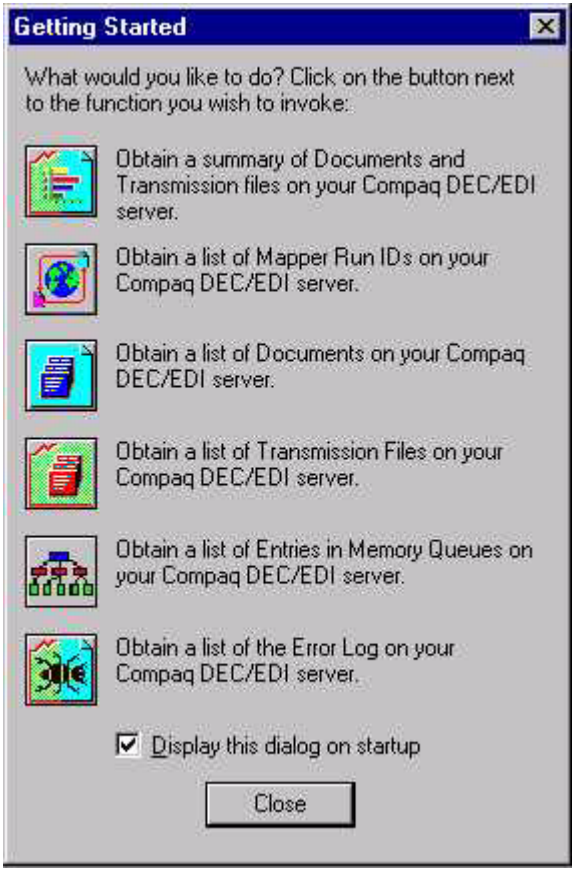
- Use the Cockpit.
- Use the `trade track` command on the Digital DEC/EDI Client.

Tracking with the Cockpit

The Cockpit allows you to track the progress of documents and transmission files through the Mapper and a Digital DEC/EDI Server. You can view this information in either summary or detailed format. The Cockpit also enables you to view Archived files, and to view and print Error Log files.

When you start the Cockpit for the first time, the Getting Started screen is displayed. **The Cockpit Getting Started** screen is shown in Figure 5-1 *The Cockpit Getting Started Screen*

Figure 5-1 The Cockpit Getting Started Screen



The screen presents a selection of the most commonly used functions that you can access through the Tool Bar and various menus. You can prevent its display on startup by unchecking the **Display this dialog on startup** option or by selecting the **Options** choice from the **File** menu.

Using the Cockpit Tracking Options

The Getting Started screen presents the following options:

- Obtain a summary of Documents and Transmission files on your Digital DEC/EDI server.

  Use this option to obtain a summary of either current or archived Documents and Transmission files logged within a date range that you can specify, and held on a server that you can select.

- Obtain a list of Mapper Run IDs on your Digital DEC/EDI server.

  Use this option to obtain a listing of Mapper Run IDs, either current or archived, within a date range that you can specify, and on a server that you can select.

- Obtain a list of Documents on your Digital DEC/EDI server.

  Use this option to obtain a listing of either current or archived documents logged within a date range that you can specify, and held on a server that you can select. You can further refine your search by specifying the Document Type, its originating application, and the amount of information on the documents that you wish to view.

- Obtain a list of Transmission files on your Digital DEC/EDI server.

  Use this option to obtain a listing of either current or archived Transmission files lodged within a date range that you can specify, and held on a server that you can select.

- View and replace the Error Log on your Digital DEC/EDI server.

  Use this option to obtain a view of the Error Log(s) held on a server that you can select.

Using the "trade track" Command

On the Digital DEC/EDI Client you use the call `trade track` to monitor the progress of documents that are being processed through the Digital DEC/EDI system. This gives you a display of the audit trail information, passed back from the Digital DEC/EDI Server. The command `trade track`, and the types of information you can request, are described in this book in *Chapter 3 Command Line Interface*.

Business References

Digital DEC/EDI stores information on objects flowing through the system in its audit trail. Each object has a unique internal identifier, by which the user can obtain information about the object.

For example, each **trade post** or **trade fetch** request that uses the mapper (**type=application_file**) generates a unique mapper run ID for that request. A single mapper run may create or fetch multiple EDI documents, each of which has its own internal Digital DEC/EDI document ID. There is no way of guessing what IDs will be generated, as the Digital DEC/EDI Server may be handling multiple applications at the same time.

Also, since these identifiers are generated internally within Digital DEC/EDI, they are not meaningful to users or business applications interrogating the audit trail.

Business References are the means by which meaningful identifiers, which the user or business application understands, can be placed in the audit trail to allow users or business applications to interrogate the audit trail in terms of information they know about - for example, a purchase order number or a trading partner name.

The business references are stored in the document audit trail, and the audit trail can accommodate up to 5 business references associated with each document. Each of these 5 business slots can be referred to individually, so the user could use each of the 5 to hold different types of keys, or they can be referred to *en-masse* if no particular scheme is in place of where particular business keys go.

Business References can be assigned by the DECEDI_POST API call or TRADE POST CLI call as the file is posted into the system or by the mapper as it converts the application file into documents. In the inbound direction, business references can be added to the document audit trail as part of the mapper mapping the document into an application file, by the DECEDI_FETCH API call, or by the TRADE FETCH CLI call.

In addition, documents that are sent using application to application routing inherit any business references that are assigned in the outgoing direction.

The contents of the business references is opaque to Digital DEC/EDI, so the user can hold whatever string values they want or compose them to fit their particular business keys. For instance, the user can define in their INVOICE maps to populate the first business reference slot with 'IN # nnnn' where nnnn is the invoice number obtained from the data being mapped. Similarly, for ORDERS, they could use 'PO # nnnn'. So, irrespective of business reference slot you could track purchase orders and invoices separately using the 'PO #' or 'IN #' prefix. Alternatively you could explicitly

assign all purchase order numbers to business reference slot 1 and all invoice numbers to business reference slot 2.

Interrogation of the audit trail by business references is supported by the DECEDI_TRACK API call, the TRADE TRACK CLI call, and by the Cockpit Graphical User Interface.

Accessing the Audit Database

Audit Database entries may be viewed and accessed using the following methods:

- The Digital DEC/EDI Application Client using the `trade track` command.
- The Digital DEC/EDI Cockpit GUI.
- SQL Interactive commands on the Server.
- An SQL program on the Server.

Audit Log

The Mapper generates events at specific points in the processing. The following lists these events and the event name assigned to each.

- **START PROCESS** — Marks when the Mapper process starts.
- **START FILE** — Marks when a new application file is created, for the incoming directory, or when a new application file is opened, for the outgoing direction.
- **START DOCUMENT** — Marks when the processing on a new document begins.
- **END DOCUMENT** — Marks when the processing for a document has completed. This happens just before the Mapper lets the Translation Service know that it has finished building/receiving the internal document file, for the outgoing/incoming direction. If a run terminates with an END DOCUMENT event, whether the transaction completed or not is unknown. Check the Digital DEC/EDI tracking to determine its disposition.
- **COMMIT DOCUMENT** — Marks when the processing for a document has been completed. This happens just after the Mapper lets the Translation Service know that it has finished building/receiving the

internal document file, for the outgoing/incoming direction, and has had a response. The document transaction has been completed.

- **END FILE** — Marks when the application file is closed.
- **END PROCESS** — Marks a normal termination of the program.
- **HOOK** — Marks when a customization routine was called that recorded an audit event for each call.
- **SOFT ERROR** — Marks when an error occurred while the Mapper was mapping a document. The Mapper has determined that it can terminate (ABORT) the current document and continue with the next.
- **HARD ERROR** — Marks when an error occurred from which the Mapper has determined it cannot continue. The current run is terminated.
- **HISTORY FILE** — Marks when a history file has been created. This event can occur at any of the history points within the runtime utility. It can also occur while a table file is being compiled in the Mapper UI if the record history option was enabled in the table at the time of the compile. The purpose of this event is to record the time of the history snapshot and the name of the file containing the history.
- **USER EVENT** — Marks when there has been a call to the routine FBR_user_audit by a customization routine to record a user declared event, and to record the associated data provided by the user.
- **MAPPING TABLE COMPILE** — Marks when there has been a successful compile of a Mapping Table file using the Mapper UI. This event is not generated by the Mapper runtime utility.
- **START RECOVERY** — Marks when the recovery process starts up. This event is not generated by the Mapper runtime utility. The events generated by the runs restarted will follow this event.
- **END RECOVERY** — Marks when the recovery process terminates. This event is not generated by the Mapper runtime utility. All events generated by the runs restarted will precede this event.

Runtime Audit Levels

The level of auditing performed by the Mapper at Runtime may be configured by using the environmental variable or logical name:

Tru64 UNIX `FBR_RUNTIME_AUDIT_LEVEL`.

OpenVMS `FBR$RUNTIME_AUDIT_LEVEL`.

You need to define this variable in the login file of the account under which you run the Mapping Service. This is normally the Digital DEC/EDI account.

The following auditing levels can be set:

Audit Level	Runtime Auditing
0	None
1 (default)	COMMIT DOCUMENT, SOFT ERROR, HARD ERROR, HOOK, USER EVENT, HISTORY FILE, START RECOVERY, END RECOVERY
2	Maximum. Auditing of any event is enabled.

Level 0 will prevent the mapper from writing any audit information at runtime to the audit database. This provides a considerable performance enhancement at runtime.

Level 1, the default, will provide better performance over maximum auditing, but not as much as gained at level 0. For standard Mapping Tables the only runtime audit point written to the audit database at level 1 will be Commit Document. The information audited for the Commit Document event provides sufficient information for tracking the document.

Level 1 is the minimum level required by Mapping Tables which utilise hook routines to access information provided by runtime auditing.

Level 2 enables maximum auditing.

The Runtime Audit Level does not affect auditing done whilst developing and compiling maps.

Audit Database Fields

The following are the data elements stored in the audit database. They are all stored in the EVENT relation.

- **RUN_ID_NBR** [6 bytes] — ALL EVENTS

  A system wide unique 6-digit decimal number assigned to a Mapper run. This tracking run number is printed in the run log for each run. The names of all history files created during this run will include this number. The number is a right justified, zero filled, decimal number.

- **ORDINAL_NBR** [10 bytes] — ALL EVENTS

  The ordinal number of the event. The first event reported will be event 1. It is used to insure that the events can be retrieved in chronological order. The number is a left justified, decimal, zero filled number.

- **EVENT** [20 bytes] — ALL EVENTS

  The event name as listed above.

- **MPR_TIMESTAMP** [DATE] — ALL EVENTS

  The date and time corresponding to the event represented by this entry. It is in binary date format.

- **USER_ID** [30 bytes] — ALL EVENTS

  The user ID of the person running the program that generated the event.

- **DCL_USER_COMMENT** [80 bytes] — START PROCESS

  The value of the DCL runtime qualifier, \COMMENT=, or the corresponding argument in the callable interface. If the run is restarted by the recovery process, it will indicate the run ID for the run being recovered.

- **FILE_ORD_NBR** [10 bytes] — START FILE, START DOCUMENT, END DOCUMENT, COMMIT DOCUMENT, END FILE, HOOK, SOFT ERROR, HARD ERROR, and USER EVENT

  The ordinal number of the application file open at the time this event occurred. The first file opened is file number 1. The number is a left justified, decimal number.

- **DOC_ORD_NBR** [10 bytes] — START DOCUMENT, END DOCUMENT, COMMIT DOCUMENT, HOOK, SOFT ERROR, HARD ERROR, and USER EVENT

  The ordinal number of the current document. The first document is number 1. The number is a left justified, decimal number.

- **RUNTIME_VER** [8 bytes] — START PROCESS

  The program version number for the Mapper runtime utility.

- **UI_VERSION** [8 bytes] — START PROCESS and COMMIT DOCUMENT

  The program version number of the Mapper UI that created the Mapping Table file.

- **COMPILE_DATA** [80 bytes] — START PROCESS and MAPPING TABLE COMPILE and COMMIT DOCUMENT

  The compile data for the Mapping Table being used. This contains the date of compile, the name of the user that compiled it, and the terminal number.

- **PID** [8 bytes] — ALL EVENTS

  The PID of the process that is performing this event. This is a null terminated, right justified zero filled, hex format number. This is used by the recovery process to insure that the runtime process that created the incomplete run is not still running.

- **DIRECTION_INDICATOR** [1 byte] — START PROCESS and COMMIT DOCUMENT

  Direction of processing, either I (for incoming) or O (for outgoing).

- **RESTART** [10 bytes] — START PROCESS and COMMIT DOCUMENT

  The restart document number taken from the DCL. This requires the use of the /RESTART=nnnn qualifier. This is a null terminated, left justified number. The first character is:

  ' ' — /MATCH not given

  '+' — /MATCH_ALL

  '-' — /MATCH_FIRST

- **APPLICATION_ID** [20 bytes] — ALL EVENTS

  In the START PROCESS event, this is the name given in the /NAMED_APPLICATION qualifier in the Application Client call. In the START DOCUMENT event, this is the name actually used for processing documents as taken from the $APPLICATION global variable.

- **PARTNER** [20 bytes] — ALL EVENTS

  In the START PROCESS event, this is the name given in the /PARTNER_ID= qualifier taken from the DCL command line. In the START DOCUMENT event, this is the name actually used for processing documents as taken from the $PARTNER global variable.

- **DOCTYPE** [20 bytes] — START PROCESS and START DOCUMENT and COMMIT DOCUMENT

  In the START PROCESS event, this is the name given in the /OBJECT_NAME qualifier in the Application Client call. In the START DOCUMENT event, this is the name actually used for processing documents as taken from the $DOCTYPE global variable. This is the Mapper document type, not the Digital DEC/EDI internal document type nor the EDI document type.

- **INTDOCTYPE** [20 bytes] START DOCUMENT and COMMIT DOCUMENT

  The name associated with the Digital DEC/EDI Internal Doctype for this document. This is the value sent to or received from Digital DEC/EDI Translation Service along with this document.

- **USER_REF** [20 bytes] — START PROCESS, START DOCUMENT (outgoing) and COMMIT DOCUMENT

  In the START PROCESS event, this is the name given in the /USER_REFERENCE= qualifier taken from the DCL command line. In the START DOCUMENT event, this is the name actually used for processing documents as taken from the $USERREF global variable.

- **TEST_INDICATOR** [20 bytes] — START PROCESS, START DOCUMENT and COMMIT DOCUMENT events.

  In the START PROCESS event, this is the name given in the /TEST_INDICATOR= qualifier taken from the DCL command line. In the START DOCUMENT event, this is the name actually used for processing documents as taken from the $TESTIND global variable. Values are LIVE, PARTNER_TEST, TRANSLATOR_TEST, and LOCAL_TEST.

- **PRIORITY_ARG** [20 bytes] — START PROCESS and START DOCUMENT and COMMIT DOCUMENT

  In the START PROCESS event, this is the name given in the /PRIORITY= qualifier taken from the DCL command line. In the START DOCUMENT event (outgoing), this is the name actually used for processing documents as taken from the $PRIORITY global variable. Values are IMMEDIATE or NORMAL.

- **FILENAME** [256 bytes] — START PROCESS, START FILE, END FILE, HISTORY FILE and COMMIT DOCUMENT

  For the START PROCESS event, this value is the argument string provided in the DCL command line to identify the application file(s). For the START FILE and END FILE events, this is the full file name with the directory path of the actual application file that was created (incoming) or opened (outgoing). For the HISTORY FILE event, this is the full file name of the history file that was created.

- **MAPPING_TABLE_ARG** [256 bytes] — START PROCESS, MAPPING TABLE COMPILE and COMMIT DOCUMENT

  For the START PROCESS event this is the argument string provided in the DCL command line to identify the Mapping Table file. This is the full file name with the directory path. For the MAPPING TABLE COMPILE event, this is the full file name of the table that was compiled.

- **NBR_OF_RECORDS** [10 bytes] — START DOCUMENT and COMMIT DOCUMENT

  The number of input/output records read or written for the current document. For incoming documents, this is on the COMMIT DOCUMENT event. For outgoing documents, this is on the START DOCUMENT event.

- **DOCUMENT_ID** [33 bytes] — COMMIT DOCUMENT

  This is the Digital DEC/EDI Document ID for the document just processed. It is the combination of the application ID, "_O_" for outgoing or "_I_" for incoming, and the doccount assigned by Digital DEC/EDI . An example of a document ID for an incoming document is AP_I_00000234.

- **STATUS** [10 bytes] — END PROCESS and HOOK

  The status of the run or the return status of the customization routine. Values are "Success", "Fail", or "Undefined". The HOOK event is only generated if the customization routine was marked for recording an audit event and hook events were enabled.

- **API_END_FLAGS_ARG** [10 bytes] — COMMIT DOCUMENT

  Value of the indicator flag assigned by the Mapper when it finnishes building or receiving the document internal file. The value indicates the treament that the document is to receive, as follows:

  - NO_ACTION — The document has been successfully received for processing (incoming). The document is to passed on for normal processing (outgoing).
  - ABORT — The document is to be put on a FAILED status queue for manual intervention (incoming or outgoing).
  - QUIT — The document is to be back on the AVAILABLE status queue (applies to incoming only).

- **API_RETURN_VALUE** [79 bytes] — COMMIT DOCUMENT

  Text of the return status message returned from the decedi$end_send (outgoing) or decedi$end_fetch (incoming) call.

- **CUSTOMIZATION_ROUTINE** [30 bytes] — HOOK

  The name of the customization routine that was called. This event is only generated if the customization routine was marked for recording an audit event and hook events were enabled.

- **ERROR_MESSAGE** [80 bytes] — SOFT ERROR or HARD ERROR

  The text of the error message.

- **USER_DATA** [256 bytes] — START PROCESS, USER EVENT and COMMIT DOCUMENT

  For the START PROCESS event, this data element holds the current default directory. This is used by the recovery process to restart a run. For the USER EVENT, this is the data provided with the call to FBR_USER_AUDIT() function to declare the event.

- **HISTORY_POINT** [20 bytes] — HISTORY FILE

  Indicates the history point being opened for recording. Values are:

  - At PROCESS START (Outgoing)
  - After PREPROCESS (Outgoing)
  - After RECORD HOOK (Outgoing)
  - At Digital DEC/EDI API (Outgoing and incoming)
  - Before RECORD HOOK (Incoming)
  - Before POSTPROCESSING (Incoming)
  - At END PROCESS (Incoming)

- **HOOK_POINT** [20 bytes] — HOOK

  The name of the hook point where the call was made. Values are:

- PREPROCESS
- POSTPROCESS
- START DOCUMENT
- END DOCUMENT
- RECORD
- SWITCH
- SOFT ERROR
- HARD ERROR
- MAPPING

History Entries

History, in the Mapper, is a snapshot of the input or output data used during processing. The actual history data is stored in binary sequential files. The names of the history files are stored in RDB audit database along with the events for the run that created them.

You can view the contents of the history files by using the TRADE TRACKING command at the DCL command line. Then select the menu option to display the history files.

History Points

The Mapping Table allows you to enable history to be taken at several points. This is specified in the Audit Controls screen of the Table Attributes tabbed dialog in the Mapping Table Editor. The Mapper can collect data a number of different points in the processing. The following list describes what sort of information can be collected at the various points:

- **Start/End of Processing**
  - At the Start of Processing:

    This is for outgoing documents only. This is a complete copy of the input application file(s) as specified in the application file parameter of the DCL run command. The snapshot is taken before the preprocess hook is called. The Mapper opens the file, reads it, then closes it before calling the preprocess hook.

  - At the End of Processing:

    This is for incoming documents only. This is a complete copy of the data written to the application file during this run of the Mapper after the post-processing hook. The Mapper reopens the application file, reads and copies the records to the history file, and then closes the file. This is all extra overhead in order to process this audit. If there is no post-processing hook or if the post-processing does not modify the data written to the application file, it is more efficient to use the "Before POSTPROCESSING" history collection point.

- **Before/After Post/Preprocessing**
  - Before Post-processing:

    This is for incoming documents only. This is a complete copy of the data written to the application file during this run of the Mapper prior to the post-processing hook. If there are no RECORD or SWITCH hooks, this is the same as the previous point.

  - After Pre-processing:

    This is for outgoing documents only. This is a complete copy of the input application file(s) as modified by the preprocess hook. If there is no preprocess hook, this point is the same as the PROCESS START history collection point. The snapshot is taken after the preprocess hook is called. The Mapper performs its normal file open and read on the application file, and then copies each record to the history file as it is being processed (before calling the RECORD hook) so there is no additional overhead for reading the application file for the sake of taking the history.

- **Before/After Record Hook**
  - Before Record Hook:

    This is for incoming documents only. This is a complete copy of the data generated from the mappings prior to being processed by the RECORD/SWITCH hooks.

  - After Record Hook:

    This is for outgoing documents only. This is a complete copy of the input application file(s) as modified by the PREPROCESS hook and the RECORD/SWITCH hook. This history collection point collects the records as seen in the internal source tree prior to mapping. If there are no record or switch hooks, this point is the same as the previous history collection point.

- **Digital DEC/EDI Internal File Format**
  - For Outgoing documents:

    This is a copy of the datalabel/datavalue pairs that the Mapper writes to the temporary internal file that it is building. This file has the same format as the internal file passed to the Translation Service when the building process is complete. If there are several documents generated by the data from the application file, they are all appended, and separated with an END-OF-FILE line.
  - For Incoming documents:

    This is a copy of the datalabel/datavalue pairs that the Mapper receives from the temporary internal file that it is reading. This file has the same format as the internal file that has been passed from the Translation Service. If there are several documents in the file, they are separated with an END-OF-FILE line.
- **Custom Routine Arguments**
- **Mapping Table**

Names of History Files

History data will be kept in straight sequential files, in binary format.

Tru64 UNIX Each of these history files will be written to the Digital DEC/EDI store directory under `/var/adm/decedi.`

OpenVMS Each of these history files will be written to the Mapper History directory, under `FBR$HISTORY`.

History file names are constructed by appending the following data:

```
FBR_ direction '_' run_id type seqnbr .HIS
```

- **FBR_** — A literal indicating that this is a Mapper file.
- **direction** — Incoming = I and Outgoing = O
- **run_id** — The tracking run identifier for the originating Mapper process. This is a 6 digit number with leading zeros.
- **type** — The type of data being stored:
  - **S** — PROCESS START history point. A copy of the application file. One history file for each application file.
  - **P** — PREPROCESS history point (Outgoing) or POSTPROCESS history point (Incoming). A copy of the record stream from, or to, or both, all of the application files. One file for each run.
  - **R** — RECORD HOOK history point. A copy of the record stream of all data placed in the internal tree (Outgoing) or the data generated from mappings (Incoming). One file for each run. One record is in the history file for each record processed.
  - **D** — The Mapper history point. This history event is used to record documents that have either been posted to or received from the Digital DEC/EDI Translation Services only.

    A history file is not generated in MAPPER_TEST mode as a local internal format file is used. In this case, the file specified by the LOCAL_TEST option contains the concatenation of all the documents processed in this map run.
  - **H** — Customization call arguments. There is one file for each run, and one record for each call. The format is shown later.
  - **C** — Table Compile. This is a copy of the compiled Mapping Table file.
- **seqnbr** — A sequence number if there are more than one of these. This is a numeric value that corresponds to the file ordinal number.
- **.HIS** — All history files will have the extension, .HIS.

The following is an example:

```
FBR_O_222222D1.HIS
```

This is a Mapper (`FBR_`) history file (`.HIS`) for an outgoing (`O`) process with the run ID of `222222`. It contains the data labels (`D`) generated and is the first file (and only file in this case).

You will be able to use wildcard characters to work with the history files. The following example causes listing of all outgoing PREPROCESS history files:

```
ls /var/adm/decedi/store_1/FBR_*.HIS
```

Format of History Files for Hook Calls

When a customization routine is marked for recording history of all calls and the customization history recording has been enabled, the Mapper records all of the arguments of each call in a history file of type H. In this file, each call is recorded in a single variable length record in binary format.

The customization routine call history record has the following format:

- **Timestamp** — QUADWORD, Binary Date format, the time the call was completed. Using this to relate it back to the other events in the audit log, the relative sequence can be determined.
- **Hook Point** — CHARACTER(32), an identifier for the hook point in which it was called. In this context the maps are considered a hook point. It is null terminated.
- **Function Name** — CHARACTER(32), the name of the function that was called. It is null terminated.
- **Return status** — UNSIGNED LONGWORD, A 32 bit integer value returned by the routine.
- **Quantity arguments** — UNSIGNED LONGWORD, the number of arguments in the call including the return value argument.
- **Arguments** — Array of STRUCTUREs, in argument order, a descriptor for each argument. The size of the array is determined by the quantity arguments field. Each descriptor contains the following:
  - Argument length — UNSIGNED WORD, 16 bit size of the argument.
  - Argument datatype — UNSIGNED BYTE, 8 bit type code.
  - Argument class — UNSIGNED BYTE, 8 bit class code.
  - Argument pointer — UNSIGNED LONGWORD, an offset from the beginning of the 8 bit class code.
  - Argument pointer — an offset from the beginning of the record to the beginning of the data.
- **Argument Data** — Based on the descriptors the data will be appended to the record following the descriptor array. They might not be in the same order as the argument descriptors, and they need not be the same size or position on the record from one record to another. The data will be the actual binary value of the data that was passed to the customization routine in the call or was returned by the customization routine in its return value argument. Since the values are captured after the call, those items that are passed by reference or by descriptor might have been modified during the call.

Chapter 6 **Debugging Facilities**

The Mapper provides these debug facilities:

1. Mapper Test
2. Mapping Debug
3. File I/O Debug

Mapper Test

When trying out a new set of maps, you will normally want to run test data through the Mapper to see what the maps generate.

By default the Mapper submits each document for processing by the Digital DEC/EDI Translation Services. However, for debugging processes, it is useful to produce just a local copy of the Digital DEC/EDI internal document so that you can examine the values assigned to individual data labels within it. In this case, the document is not processed further by the Translation Services.

To do this, you specify **test_indicator=mapper_test** in the Application Client call. For example:

UNIX

```
# trade post my-app -test_indicator=mapper_test test.dat \
    -partner_name=acme-orders -tracking_reference="ORDER_1234"
    -table_name=map_test_tbl
```

You may also wish to debug a table, after an error occurs. Again, the Application Client call needs to specify **test_indicator=mapper_test**

It is useful to specify the following qualifiers when running a Mapper test:

- **debug** to specify a debug log file.
- **output_file** to specify an screen output file.
- **local_test** to specify a local internal file, instead of passing the document to or from the Translation Service.

  For outgoing documents, this is the file to which the Mapper will write the internal document.

  For incoming documents, this is the file from which the Mapper will read the internal document.

Tru64 UNIX This defaults to the file identified by the environment variable FBR_LOCAL_TEST_IN if defined; otherwise the file is written to FBR_LOCAL_TEST_IN.IHF.

OpenVMS This defaults to the file idetified by the logical name FBR$LOCAL_TEST_IN if defined; otherwise the file is written to FBR$LOCAL_TEST_IN.IHF.

The format of the Mapper test file is the same as the in-house files in Digital DEC/EDI. Note that on incoming, the $RECEIVER-ID is assumed to be the application and is ignored. The $SENDER-ID is the partner. The $DOCUMENT-TYPE is the Digital DEC/EDI internal document type. More than one document can be placed in the file if the documents are terminated with the $END-OF-FILE flag. The document selection criteria is ignored and you get whatever is in the file.

The following shows a sample in-house file for an incoming table.

```
$SENDER-ID : DEC-DIRECT-UK-LTD
$RECEIVER-ID : SHINY-NEW-SYSTEMS
$DOCUMENT-TYPE : MINVOICE
$USER-REFERENCE : UNSPECIFIED
$TEST-INDICATOR : 3
$GROUP
BGM_C002_1001 : 380
BGM_1004 : DOC-MINVOX-206523
BGM_C031_2001 : 930401
BGM_C031_2002 : 1501
BGM_1225 : 00
$GROUP
NAD_3035 : II
NAD_C080_3036_1 : Digital Equipment Company Ltd
NAD_C059_3042_1 : Digital Park
NAD_C059_3042_2 : Worton Grange
```

```
NAD_C059_3042_3 : Imperial Way
NAD_3164 : Reading
NAD_3229 : Berks
NAD_3251 : RG2 0TE
NAD_3207 : GB
$GROUP
LOC_3227 : 20
LOC_C087_3224 : Goods Inwards
$END-GROUP
$END-GROUP
$GROUP
NAD_3035 : BY
NAD_C080_3036_1 : Shiny New Systems Ltd
NAD_C059_3042_1 : Unit 1
NAD_C059_3042_2 : Thames Valley Park
NAD_C059_3042_3 :
NAD_3164 : Henley-on-Thames
NAD_3229 : Oxon
NAD_3251 : RG9 1JB
NAD_3207 : GB
$GROUP
LOC_3227 : 20
LOC_C087_3224 : Goods Inwards
$END-GROUP
$END-GROUP
$GROUP
PAT_4279 : 02
PAT_C012_2001 : 930401
PAT_C012_2005 : 003
PAT_5306 : 1000.00
PAT_5484 : 100
PAT_C104_4276_1 : MC/VISA payment
$END-GROUP
$END-GROUP
$GROUP
$GROUP
LIN_1082 : 1
LIN_C198_7020_1 : DSR-01
LIN_C198_7023_1 : MG
LIN_C118_5110 : 50.00
LIN_C118_5375 : RT
LIN_6170 : 2
LIN_5116 : 100.00
$GROUP
PAC_7224 : 2
PAC_C202_7065 : BE
PAC_C202_7064_1 : bundle
$END-GROUP
```

```
$END-GROUP
$GROUP
LIN_1082 : 2
LIN_C198_7020_1 : DSR-02
LIN_C198_7023_1 : MG
LIN_C118_5110 : 250.00
LIN_C118_5375 : RT
LIN_6170 : 2
LIN_5116 : 500.00
$GROUP
PAC_7224 : 2
PAC_C202_7065 : CF
PAC_C202_7064_1 : coffer
$END-GROUP
$END-GROUP
$GROUP
LIN_1082 : 3
LIN_C198_7020_1 : DSR-03
LIN_C198_7023_1 : MG
LIN_C118_5110 : 100.00
LIN_C118_5375 : RT
LIN_6170 : 3
LIN_5116 : 300.00
$GROUP
PAC_7224 : 3
PAC_C202_7065 : WB
PAC_C202_7064_1 : wickerbottle
$END-GROUP
$END-GROUP
$GROUP
LIN_1082 : 4
LIN_C198_7020_1 : DSR-04
LIN_C198_7023_1 : MG
LIN_C118_5110 : 50.00
LIN_C118_5375 : RT
LIN_6170 : 2
LIN_5116 : 100.00
$GROUP
PAC_7224 : 2
PAC_C202_7065 : BE
PAC_C202_7064_1 : bundle
$END-GROUP
$END-GROUP
$END-GROUP
$GROUP
TMA_5356 : 1175.00
TMA_5360 : 1000.00
TMA_5338 : 1175.00
```

```
TMA_5492 : 175.00
$END-GROUP
$END-OF-FILE
```

Mapping Debug

Debugging a mapping table can be a tedious operation without some idea about what is happening during the mapping process. The Mapper provides assistance by capturing information about the mappings as they occur during a run of test or live data.

You activate the mapping debug facilities by specifying a **debug** qualifier when you send or fetch a document using the Client call `trade post` or `trade fetch`.

When activated, the Mapper writes into the file an entry for each map that is executed, an entry for each mapping assignment made within the map, and the generated data-label/data-value pairs.

The following is a sample debug file `minvox_o.log` generated from the minvox examples used in the Mapping Tutorial:

```
               MAP EDITOR Version V3.0
                   MAP LOG                        18-AUG-1995
12:48:31.96

Map Editor Table File: /var/adm/decedi/maps/minvox_o.fbo
Application File: /usr/users/test/minvox_o.dat
  amount_due = 0;
      VALUE SET TO: "0"
  vat_rate = 17.5;
      VALUE SET TO: "17.5"
  message_amount = 0;
      VALUE SET TO: "0"
  line_total = 0;
      VALUE SET TO: "0"
  taxable_amount = 0;
      VALUE SET TO: "0"
  tax_amount = 0;
      VALUE SET TO: "0"
      $DOCTYPE= "MINVOICE";
      VALUE SET TO: "MINVOICE"

SOURCE TREE:
```

```
  INVOICE_HDR{1}               B380DOC-MINVOX-206523
930401150100.
   NAME_AND_ADDRESS{1}      NIIDigital Equipment Company Ltd
Digital
Par
   NAME_AND_ADDRESS{2}       NBYShiny New Systems Ltd
Unit 1
    DELIVERY_LOC{1}             D20 Goods Inwards    .
    PAY_TERMS{1}
P02930401003000000000010000000000100MC/VISA
payment
   LINE_ITEM{1}            L000001DSR-01
MG
0000
      PACKAGE{1}               C000002BE     .
   LINE_ITEM{2}            L000002DSR-02
MG
0000
      PACKAGE{1}               C000002CF     .
   LINE_ITEM{3}            L000003DSR-03
MG
0000
      PACKAGE{1}               C000003WB     .
   LINE_ITEM{4}            L000004DSR-04
MG
0000
      PACKAGE{1}               C000002BE     .

Selecting the Mapping Set Based on:
  $DOCTYPE: MINVOICE
  $PARTNER: SHINY-NEW-SYSTEMS
  Using MAPPING SET #1

 Using MAP #1
    MAP ID: Unspecified
  FB SEGMENT: heading;
  DEFAULT SOURCE INSTANCE: (nil)
  amount_due = 0;
      VALUE SET TO: "0"
  message_amount = 0;
      VALUE SET TO: "0"
  line_total = 0;
      VALUE SET TO: "0"
  taxable_amount = 0;
      VALUE SET TO: "0"
  tax_amount = 0;
      VALUE SET TO: "0"

 Using MAP #2
```

```
   MAP ID: Unspecified
 FB SEGMENT: BGM;
 DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}
 BGM_C002_1001 = DOC_NAME_CODE;
     VALUE SET TO: "380"
 BGM_1004 = DOC_REF_NUM;
     VALUE SET TO: "DOC-MINVOX-206523                  "
 BGM_C031_2001 = DOC_DATE;
     VALUE SET TO: "930401"
 BGM_C031_2002 = DOC_TIME;
     VALUE SET TO: "1501"
 BGM_1225 = DOC_FUNC_CODE;
     VALUE SET TO: "00"
Using MAP #3
   MAP ID: Unspecified
 FB SEGMENT: NAD;
 DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.NAME_AND_ADDRESS{1}
 NAD_3035 = PARTY_QUAL;
     VALUE SET TO: "II "
 NAD_C080_3036_1 = PARTY_NAME;
     VALUE SET TO: "Digital Equipment Company Ltd      "
 NAD_C059_3042_1 = STREET_1;
     VALUE SET TO: "Digital Park                       "
 NAD_C059_3042_2 = STREET_2;
     VALUE SET TO: "Worton Grange                      "
 NAD_C059_3042_3 = STREET_3;
     VALUE SET TO: "Imperial Way                       "
 NAD_3164 = CITY;
     VALUE SET TO: "Reading                            "
 NAD_3229 = COUNTY;
     VALUE SET TO: "Berks    "
 NAD_3251 = POST_CODE;
     VALUE SET TO: "RG2 0TE  "
 NAD_3207 = COUNTRY;
     VALUE SET TO: "GB"
Using MAP #4
   MAP ID: Unspecified
 FB SEGMENT: LOC;
 DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.DELIVERY_LOC{1}
 LOC_3227 = LOC_QUAL;
     VALUE SET TO: "20 "
 LOC_C087_3224 = LOC_NAME;
     VALUE SET TO: "Goods Inwards    "
Using MAP #3
   MAP ID: Unspecified
 FB SEGMENT: NAD;
 DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.NAME_AND_ADDRESS{2}
 NAD_3035 = PARTY_QUAL;
```

```
        VALUE SET TO: "BY "
    NAD_C080_3036_1 = PARTY_NAME;
        VALUE SET TO: "Shiny New Systems Ltd              "
    NAD_C059_3042_1 = STREET_1;
        VALUE SET TO: "Unit 1                             "
    NAD_C059_3042_2 = STREET_2;
        VALUE SET TO: "Thames Valley Park                 "
    NAD_C059_3042_3 = STREET_3;
        VALUE SET TO: "                                   "
    NAD_3164 = CITY;
        VALUE SET TO: "Henley-on-Thames                   "
    NAD_3229 = COUNTY;
        VALUE SET TO: "Oxon     "
    NAD_3251 = POST_CODE;
        VALUE SET TO: "RG9 1JB  "
    NAD_3207 = COUNTRY;
        VALUE SET TO: "GB"

   Using MAP #4
      MAP ID: Unspecified
    FB SEGMENT: LOC;
    DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.DELIVERY_LOC{1}
    LOC_3227 = LOC_QUAL;
        VALUE SET TO: "20 "
    LOC_C087_3224 = LOC_NAME;
        VALUE SET TO: "Goods Inwards    "

   Using MAP #5
      MAP ID: Unspecified
    FB SEGMENT: PAT;
    DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.PAY_TERMS{1}
    PAT_4279 = TERMS_TYPE;
        VALUE SET TO: "02"
    PAT_C012_2001 = TERMS_DATE;
        VALUE SET TO: "930401"
    PAT_C012_2005 = DATE_QUAL;
        VALUE SET TO: "003"
    PAT_5306 = MIN_DUE;
        VALUE SET TO: "1000.00"
    PAT_5484 = PERCENT_PAYABLE;
        VALUE SET TO: "100"
    PAT_C104_4276_1 = PAYMENT_TERMS;
        VALUE SET TO: "MC/VISA payment                    "

   Using MAP #6
      MAP ID: Unspecified
    FB SEGMENT: detail;
    DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.PAY_TERMS{1}
```

```
<ONLINE_CHUNK>
<ENDCODE_EXAMPLE>
<CODE_EXAMPLE>
 Using MAP #7
    MAP ID: Unspecified
  FB SEGMENT: LIN;
  DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.LINE_ITEM{1}
  LIN_1082 = ITEM_NO;
      VALUE SET TO: "1"
  LIN_C198_7020_1 = ARTICLE_NO;
      VALUE SET TO: "DSR-01                                "
  LIN_C198_7023_1 = ARTICLE_ID;
      VALUE SET TO: "MG "
  LIN_C118_5110 = UNIT_PRICE;
      VALUE SET TO: "50.00"
  LIN_C118_5375 = PRICE_TYPE;
      VALUE SET TO: "RT"
  LIN_6170 = NUM_UNITS;
      VALUE SET TO: "2"
  amount_due = UNIT_PRICE * NUM_UNITS;
      VALUE SET TO: "100"
  message_amount = message_amount + amount_due;
      VALUE SET TO: "100"
  line_total = message_amount;
      VALUE SET TO: "100"
  LIN_5116 = $ROUND(amount_due,2);
      VALUE SET TO: "100.00"

 Using MAP #8
    MAP ID: Unspecified
  FB SEGMENT: PAC;
  DEFAULT SOURCE INSTANCE:
INVOICE_HDR{1}.LINE_ITEM{1}.PACKAGE{1}
  PAC_7224 = NUM_PACKAGES;
      VALUE SET TO: "2"
  PAC_C202_7065 = PACKAGE_CODE;
      VALUE SET TO: "BE     "
  PAC_C202_7064_1 = $LOOKUP(package_types,PACKAGE_CODE);
      VALUE SET TO: "bundle                                "

 Using MAP #7
    MAP ID: Unspecified
  FB SEGMENT: LIN;
  DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.LINE_ITEM{2}
  LIN_1082 = ITEM_NO;
      VALUE SET TO: "2"
  LIN_C198_7020_1 = ARTICLE_NO;
      VALUE SET TO: "DSR-02                                "
```

```
  LIN_C198_7023_1 = ARTICLE_ID;
      VALUE SET TO: "MG "
  LIN_C118_5110 = UNIT_PRICE;
      VALUE SET TO: "250.00"
  LIN_C118_5375 = PRICE_TYPE;
      VALUE SET TO: "RT"
  LIN_6170 = NUM_UNITS;
      VALUE SET TO: "2"
  amount_due = UNIT_PRICE * NUM_UNITS;
      VALUE SET TO: "500"
  message_amount = message_amount + amount_due;
      VALUE SET TO: "600"
  line_total = message_amount;
      VALUE SET TO: "600"
  LIN_5116 = $ROUND(amount_due,2);
      VALUE SET TO: "500.00"
 Using MAP #8
    MAP ID: Unspecified
  FB SEGMENT: PAC;
  DEFAULT SOURCE INSTANCE:
INVOICE_HDR{1}.LINE_ITEM{2}.PACKAGE{1}
  PAC_7224 = NUM_PACKAGES;
      VALUE SET TO: "2"
  PAC_C202_7065 = PACKAGE_CODE;
      VALUE SET TO: "CF     "
  PAC_C202_7064_1 = $LOOKUP(package_types,PACKAGE_CODE);
      VALUE SET TO: "coffer                                "

 Using MAP #7
    MAP ID: Unspecified
  FB SEGMENT: LIN;
  DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.LINE_ITEM{3}
  LIN_1082 = ITEM_NO;
      VALUE SET TO: "3"
  LIN_C198_7020_1 = ARTICLE_NO;
      VALUE SET TO: "DSR-03                                "
  LIN_C198_7023_1 = ARTICLE_ID;
      VALUE SET TO: "MG "
  LIN_C118_5110 = UNIT_PRICE;
      VALUE SET TO: "100.00"
  LIN_C118_5375 = PRICE_TYPE;
      VALUE SET TO: "RT"
  LIN_6170 = NUM_UNITS;
      VALUE SET TO: "3"
  amount_due = UNIT_PRICE * NUM_UNITS;
      VALUE SET TO: "300"
  message_amount = message_amount + amount_due;
      VALUE SET TO: "900"
```

```
  line_total = message_amount;
      VALUE SET TO: "900"
  LIN_5116 = $ROUND(amount_due,2);
      VALUE SET TO: "300.00"

 Using MAP #8
    MAP ID: Unspecified
  FB SEGMENT: PAC;
  DEFAULT SOURCE INSTANCE:
INVOICE_HDR{1}.LINE_ITEM{3}.PACKAGE{1}
  PAC_7224 = NUM_PACKAGES;
      VALUE SET TO: "3"
  PAC_C202_7065 = PACKAGE_CODE;
      VALUE SET TO: "WB     "
  PAC_C202_7064_1 = $LOOKUP(package_types,PACKAGE_CODE);
      VALUE SET TO: "wickerbottle                          "

 Using MAP #7
    MAP ID: Unspecified
  FB SEGMENT: LIN;
  DEFAULT SOURCE INSTANCE:  INVOICE_HDR{1}.LINE_ITEM{4}
  LIN_1082 = ITEM_NO;
      VALUE SET TO: "4"
  LIN_C198_7020_1 = ARTICLE_NO;
      VALUE SET TO: "DSR-04                                "
  LIN_C198_7023_1 = ARTICLE_ID;
      VALUE SET TO: "MG "
  LIN_C118_5110 = UNIT_PRICE;
      VALUE SET TO: "50.00"
  LIN_C118_5375 = PRICE_TYPE;
      VALUE SET TO: "RT"
  LIN_6170 = NUM_UNITS;
      VALUE SET TO: "2"
  amount_due = UNIT_PRICE * NUM_UNITS;
      VALUE SET TO: "100"
  message_amount = message_amount + amount_due;
      VALUE SET TO: "1000"
  line_total = message_amount;
      VALUE SET TO: "1000"
  LIN_5116 = $ROUND(amount_due,2);
      VALUE SET TO: "100.00"

 Using MAP #8
    MAP ID: Unspecified
  FB SEGMENT: PAC;
  DEFAULT SOURCE INSTANCE:
INVOICE_HDR{1}.LINE_ITEM{4}.PACKAGE{1}
  PAC_7224 = NUM_PACKAGES;
```

```
      VALUE SET TO: "2"
  PAC_C202_7065 = PACKAGE_CODE;
      VALUE SET TO: "BE     "
  PAC_C202_7064_1 = $LOOKUP(package_types,PACKAGE_CODE);
      VALUE SET TO: "bundle                                "

 Using MAP #9
    MAP ID: Unspecified
  FB SEGMENT: summary;
  DEFAULT SOURCE INSTANCE:
INVOICE_HDR{1}.LINE_ITEM{4}.PACKAGE{1}

 Using MAP #10
    MAP ID: Unspecified
  FB SEGMENT: TMA;
  DEFAULT SOURCE INSTANCE:
INVOICE_HDR{1}.LINE_ITEM{4}.PACKAGE{1}
  tax_amount = (message_amount*vat_rate) * 0.01;
      VALUE SET TO: "175"
  message_amount = message_amount + tax_amount;
      VALUE SET TO: "1175"
  TMA_5356 = $ROUND(message_amount,2);
      VALUE SET TO: "1175.00"
  TMA_5360 = $ROUND(line_total,2);
      VALUE SET TO: "1000.00"
  TMA_5338 = $ROUND(message_amount,2);
      VALUE SET TO: "1175.00"
  TMA_5492 = $ROUND(tax_amount,2);
      VALUE SET TO: "175.00"
Document DEC-DIRECT-UK-LTD_O_0000000001
```

File I/O Debug

Another debug facility provided by the Mapper lets you generate a listing of all disk I/O performed by the Mapper in reading or writing the application files.

You activate the I/O debug facilities by specifying an **io_debug** qualifier when you send or fetch a document using the Client call TRADE POST or TRADE FETCH.

This facility is useful in resolving problems where the RMS attributes of the application file do not match those expected by the Mapper, or the Mapper creates a file using RMS attributes other than those expected by the application software.

The following is a sample I/O debug file `minvox_o.io` generated from the minvox examples used in the Mapping Tutorial:

```
        id=4, access = 1,
        name=.//minvox_o.dat, buflen=1024
rec# 1; len=35,(first 30 chars)="B380DOC-MINVOX-206523
93040115"
rec# 2; len=199,(first 30 chars)="NIIDigital Equipment Company
L"
rec# 3; len=199,(first 30 chars)="NBYShiny New Systems Ltd
"
rec# 4; len=22,(first 22 chars)="D20 Goods Inwards    ."
rec# 5; len=70,(first 30
chars)="P02930401003000000000100000000"
rec# 6; len=72,(first 30 chars)="L000001DSR-01                "
rec# 7; len=15,(first 15 chars)="C000002BE     ."
rec# 8; len=72,(first 30 chars)="L000002DSR-02                "
rec# 9; len=15,(first 15 chars)="C000002CF     ."
rec# 10; len=72,(first 30 chars)="L000003DSR-03
"
rec# 11; len=15,(first 15 chars)="C000003WB     ."
rec# 12; len=72,(first 30 chars)="L000004DSR-04
"
rec# 13; len=15,(first 15 chars)="C000002BE     ."
            name=
```

Mapper Messages Written to Standard Output

The following is a small sample file (`minvox_o.out`) of additional messages output by the Mapping Service during the mapping run. Note that these messages include the generated Document ID and the Map Run ID:

```
Copyright Digital Equipment Corporation 1990,1997. All rights
reserved.
Mapping Table Tracking Run ID: 001942
Document DEC-DIRECT-UK-LTD_O_0000001108 generated

normal completion - SEND
```

Error Log

An Error Log file collects the output of any severe errors that cause the Mapper to abort processing. This can be useful if the Application Client call returns a severe error condition. The following output is an example:

Tru64 UNIX

```
DECEDI__INPUT_OPEN_ERROR (e), unable to open input file
```

```
/usr/users/test/minbad_o.dat
```

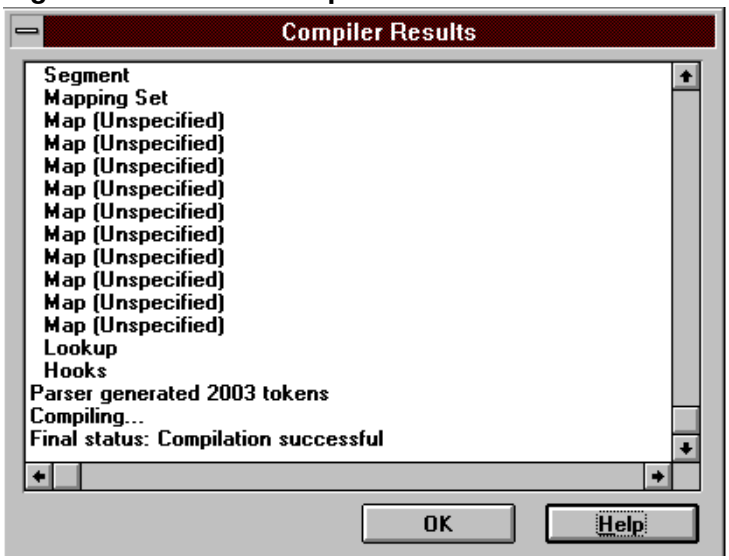
OpenVMS

```
DECEDI$_INPUT_OPEN_ERROR (e), unable to open input file
USER:[NAME]MINBAD_O.DAT;
```

Debugging Compilation Errors

When you compile a Mapping Table, the Mapping Table Editor displays a **Compiler Results** screen output of events during the various stages of the process. The **Compiler Results** screen is shown in Figure 6-1 *The Compiler Results Screen*.

Figure 6-1 The Compiler Results Screen



The screen output is collected in the file `mapper.err` that can be useful in debugging.

Information about errors is collected in a `.lis` file, generated by the compiler. Within the compiler results screen, and compiler listing file there may be referenced to particular line numbers that contain errors. To examine the contents of the line referenced, refer to the `.src` file, which is the input file produced by the Mapping Table Editor, prior to the compilation phase.

It is recommended you use a suitable programmer's editor to examine this file, to enable you to locate the specific line number referenced within the `mapper.err` file.

Chapter 7 Problem Solving in the Mapper

This chapter describes common problems encountered when using the Mapper, and the steps that should be taken to rectify or avoid them.

Map Failed

When the Mapping Service fails to successfully process an Application File or set of Documents, the following error message is returned:

UNIX

```
# trade post DEC-DIRECT-UK-LTD minvox_o.dat -table_name=minvox
Failed to post file 1
Map Failed
```

Take the following steps to determine where the error occurred:

- Application Client Parameters — Check that the parameters have been specified correctly on the Application Client call.
- Server Configuration — Ensure that the Application Client has been configured using the Management Services Editor.
- Mapping Table — Ensure that the Mapping Table specified has been compiled and copied to the Server node.
- Audit Records — Use the Cockpit to view the Current Mapper Audit Database to see if any new audit records have been generated by the Mapper.

  If one or more HARD ERROR or SOFT ERROR records have been generated for the specified Map Run ID, then examine the record details to view the error message text.

If an error message has been generated, refer to the description of the error in the appendixes in this book. Take appropriate action to correct the error.

- Debug Log — If no errors have been logged in the Mapper Audit Database, then it may be necessary to obtain further debugging information by running the Map request again.

Obtaining Further Details of a Map Run

To obtain further information about how far the map run progressed before encountering the error, reissue the map request specifying the `-debug` option. This will generate a detailed log of the map run, along with any error messages.

Under some circumstances, the information provided in the Debug Log file will be insufficient to determine the cause of the error.

In this case, reissue the map request specifying the `-output_file` and `-error_log` options. This will generate a more detailed output from the Mapper, along with any error messages logged.

In the following example, the table name is specified incorrectly:

UNIX

```
# trade post DEC-DIRECT-UK-LTD minvox_o.dat \
               -table_name=mi -debug=minvox_o.log\
               -output_file=minvox_o.out\
               -error_log=minvox_o.err
Failed to post file 1
Map Failed
```

The following is the contents of the Map Debug log:

```
# more minvox_o.log
                FILEBRIDGE Version V3.2
                    MAP LOG                        02-AUG-1995
13:36:14.61

FileBridge Table File:
Application File: /usr/users/edi/test/minvox_o.dat
DECEDI__TABLEFILE_OPEN_ERROR (e), unable to open the
FileBridge table
```

The following is the contents of the Map Output file:

```
# more minvox_o.out
```

```
Copyright Digital Equipment Corporation 1990,1994. All rights
reserved.
get_mod_date: cannot access /var/adm/decedi/maps/mi.fbo
DECEDI__TABLEFILE_OPEN_ERROR (e), unable to open the
FileBridge table

processing terminated with error - SEND
```

The following is the contents of the Error Log file:

```
# more minvox_o.err
DECEDI__TABLEFILE_OPEN_ERROR (e), unable to open the
FileBridge table
```

Server Error

Under some circumstances, the Mapper may not be able to access one or more of the files in the specified directory.

This is becase the Mapper process normally runs under the decedi user id, which may not have the correct ownership or group privileges to access the specified files. For example:

UNIX

```
#  trade post DEC-DIRECT-UK-LTD minvox_o.dat \
             -table_name=minvox_o -debug=minvox_o.log \
             -output_file=minvox_o.out -error_log=minvox_o.err
Server Error
```

In this case, examine the Digital DEC/EDI Error log, using either the Cockpit or the decedi_look command on the Server:

Tru64 UNIX

```
# decedi_look | tail -20

Wed Aug  2 13:49:42 1995 PID =  8386 NAME = Post/Fetch Server
(CORBA)
DECEDI__OPERATION_ABORTED (e), operation Aborted
DECEDI__LOGERR (e), unable to open log file
/usr/users/edi/test/fred/minvox_o.err
```

OpenVMS

```
$ trade post DEC-DIRECT-UK-LTD minvox_out.dat /table=minvox_o
%DECEDI-E-OPENFILEERR, Cannot open file.  See decedi.err for
more information.
$ TYPE DECEDI.ERR
Fri Nov 21 12:26:33 1997
Function : fopen
Reason : permission denied
Details : Can't open file DECEDI$DATA:DECEDI_SERVERS.DAT.
```

Examine the file attributes of the specified file or the parent directory to ensure that they have the correct protection, and can be written to by the process user id under which the Mapper process is running.

Reproducing the Error in Mapper Test Mode

To assist with narrowing a problem down to the Mapper, it is useful to rerun the test specifying the option `-test_indicator=mapper_test`, along with the `-local_test` option to specify the Digital DEC/EDI Internal format file to be used or generated.

In this case, no matter how far the Mapping progresses, no documents will be submitted to the Digital DEC/EDI Translation Services for processing.

This is is useful during Mapping Table development, where errors may occur more frequently, and you do not wish to send test data to your Trading Partner.

For an Incoming Mapping Table, you also do not need to receive data through the Digital DEC/EDI translation services, as you may use a local copy of the Internal Format file as many times as required.

When you think you have discovered a Software Problem, ensure that the problem is reproducible in Mapper Test mode.

The following example shows the use of Mapper Test mode to generate an Outgoing Digital DEC/EDI Interal Format file, and the subsequent use of that file for an Incoming mapping request:

```
# trade post DEC-DIRECT-UK-LTD -test_indicator=mapper_test \
    minvox_o.dat -table_name=minvox_o -local_test=minvox_o.ihf
Returned internal reference : 000563
```

The following is the contents of the Digital DEC/EDI Internal Format file. This shows the data generated for the individual data elements within each EDI segment, and the groupings of EDI segments within the EDI document definition.

You may examine the contents of this file to determine whether the correct data has been generated:

```
# more minvox_o.ihf
$SENDER-ID : DEC-DIRECT-UK-LTD
$RECEIVER-ID : SHINY-NEW-SYSTEMS
$DOCUMENT-TYPE : MINVOICE
```

```
$USER-REFERENCE : UNSPECIFIED
$TEST-INDICATOR : 3
$GROUP
BGM_C002_1001 : 380
BGM_1004 : DOC-MINVOX-206523
BGM_C031_2001 : 930401
BGM_C031_2002 : 1501
BGM_1225 : 00
$GROUP
...
$GROUP
TMA_5356 : 1175.00
TMA_5360 : 1000.00
TMA_5338 : 1175.00
TMA_5492 : 175.00
$END-GROUP
$END-OF-FILE
```

The following command may be used to fetch the local Digital DEC/EDI Internal Format file:

```
# trade fetch SHINY-NEW-SYSTEMS -test_indicator=mapper_test \
   minvox_i.dat -table_name=minvox_i -local_test=minvox_o.ihf
Returned internal reference : 000564
```

The following is the resulting Application File generated during the Incoming mapping request:

```
# more minvox_i.dat
B380DOC-MINVOX-206523 930401150100
NIIDigital Equipment Company Ltd    Digital Park
\
   Worton Grange                  Imperial Way
\
    Reading                             Berks    RG2 0TE  GB
NBYShiny New Systems Ltd           Unit 1
\
    Thames Valley Park                                    Henley-
on-Thames  \
                Oxon     RG9 1JB  GB
D20 Goods Inwards
P0293040100300000000010000000100MC/VISA payment
L000001DSR-01                                 MG
000000000005000RT00000002000000000010000
C000002BE     bundle
L000002DSR-02                                 MG
000000000025000RT00000002000000000050000
C000002CF     coffer
```

```
L000003DSR-03                                MG
000000000010000RT00000000300000000030000
C000003WB     wickerbottle
L000004DSR-04                                MG
000000000005000RT00000000200000000010000
C000002BE     bundle
T000000000117500000000000100000000000000117500000000000017500
```

Data Does Not Agree With Specified Source Hierarchy

This error is due to the Application File being of a different record structure or sequence from that defined in the Mapping Table. This may be due to using an incorrect Mapping Table.

If this error occurs, use the `-debug` option to determine at what point in the Application File the incorrect record occurred:

```
# trade post DEC-DIRECT-UK-LTD minvox_inv_hier_o.dat \
                  -table_name=minvox_o -debug=minvox_o.log
Failed to post file 1
Map Failed
# more minvox_o.log
               FILEBRIDGE Version V2.1
                   MAP LOG                          02-AUG-1995
15:58:02.70

FileBridge Table File: /var/adm/decedi/maps/minvox_o.fbo
Application File: /usr/users/opedi1/test/minvox_inv_hier_o.dat
      VALUE SET TO: "0"
      VALUE SET TO: "17.5"
      VALUE SET TO: "0"
      VALUE SET TO: "0"
      VALUE SET TO: "0"
      VALUE SET TO: "0"
      VALUE SET TO: "MINVOICE"
SOURCE TREE:
  INVOICE_HDR{1}               B380DOC-MINVOX-206523
930401150100.
   NAME_AND_ADDRESS{1}      NIIDigital Equipment Company Ltd
Digital Par
DECEDI__INVHIERKY (e), data does not agree with specified
source hierarchy
DECEDI__OPERATION_ABORTED (e), operation Aborted
```

In this case, the invalid hierarchy error occurs while processing the record just after the NII record.

Examine the record structure and data, and ensure that it corresponds to the record definition in the Mapping Table Editor.

Often such errors will be due to an invalid definition of the record sequence, such as having more than one record at level 1.

Mapping Compilation Errors

Mapping Compilation Errors may occur when the Mapping Table is compiled from within the Mapping Table Editor.

Mapping Compilation Consists of three phases:

1. Source Generation

   A file with the file extension `.src` is generated. This is an ASCII text file that is used as input to the Compiler.

2. Parsing

   The contents of the source file is parsed, and each section of the file is logged to the Compilation output screen as it is processed.

   This phase detects any syntax errors in the Mapping Statements, or any necessary parts of the Mapping Table that have not yet been defined.

   Syntax Errors are logged to the Compiler Results screen, and written to the a file with extension `.lis`.

   This is an ASCII text file that contains a copy of the `.src` input file, with any error messages embedded.

3. Compilation

   The tokens generated during the parse phase are compiled into binary form, and the semantic content of the Mapping Table is checked.

   Any errors encountered during this phase are written to the Compiler Results screen. This displays the error message and line number within the `.src` file associated with the error.

   This phase produces a file with the file extension `.fbo`, which may be copied to the server if compilation is successful.

Supplying Further Information on Mapping Errors

Runtime Errors

If you encounter an undocumented problem, please contact your Digital Support Representative supplying the following details to enable them to reproduce the problem:

- Command Used — The Client program or shell command used to submit or receive the Application File, plus any messages or status information returned.
- The Application File — This is the Application File submitted by the Application Client in an Outgoing mapping request.
- The Debug File — This is the contents of the file generated by the `-debug` option. The convention is to give this a file extension of `.log`
- The Error Log File — This is the contents of the file generated by the `-error_log` option. The convention is to give this a file extension of `.err`
- The Output File — This is the contents of the file generated by the `-output_file` option. The convention is to give this a file extension of `.out`
- The IO Debug File — This is the contents of the file generated by the `-io_debug` option. The convention is to give this a file extension of `.io`
- The Mapping Table source file — This is the uncompiled Mapping Table, which has a file type of `.fbi`, which normally resides in the directory `C:\EDICC` on the PC where the Command Center is installed.
- The Compiled Mapping Table file — This is the corresponding compiled Mapping Table, which has a file extension of `.fbo` and normally resides on the Server in the directory

  ```
  /var/adm/decedi/maps
  ```

- The Local Inhouse file — If you specified `-test_indicator=mapper_test`, then this is the local Digital DEC/EDI Internal Format file specified in the `-local_test` option.
- The Incoming Document Inhouse files — This is the collection of documents that have been processed by the Mapper during an Incoming

mapping fetch request. These files have a file extension of `.in_house_file` and reside in the directory

```
/var/adm/decedi/store_n
```

where *n* is the Digital DEC/EDI store directory number.

The documents processed during the fetch request can be determined from the `-output_file` output, which specifies the document counts for those files, up to the point where the error occurs. All files must be included to assist with reproducing the error.

- Details from the Digital DEC/EDI Error Log — If any relevant errors are logged in the Digital DEC/EDI Error log, capture the output from `decedi_look` to a file, and include that.

Collect the relevant files from the above list, and create a tar archive of the files. Some files may need to be copied from the PC using FTP or equivalent. The tar file may then be sent to your Digital Support Representative.

If using electronic mail to send binary files, you may use the `uuencode` utility to encode them into ASCII format.

Mapping Table Editor Errors

If an undocumented problem occurs within the Mapping Table editor, either during the definition or compilation phase, please contact your Digital Support Representative supplying the following files to enable them to reproduce the problem:

- The Mapping Table source file — This is the uncompiled Mapping Table, which has a file type of `.fbi`.
- The Mapping Table source listing file — This is the source listing corresponding to the Mapping Table, which has a file type of `.src`.
- The Mapping Table Compilation listing file — This is the compilation listing produced by the Mapping Table Compiler, which has a file type of `.lis`.
- The Compilation Output log file — The output from the Compiler Results screen is written to the file `mapper.err`, in the CommandCenter installation directory. This file should also be included.

Chapter 8 Application Client Error Messages

This chapter shows the error messages generated by the Application Client. Two versions of each message are shown; the first line is the status returned by the API to the calling routine, while the second line represents the text output by the CLI.

DECEDI_BADFILESPEC — Bad file name

Severity: Error.

Explanation: One of the files specified in the request was not valid. Possible reasons for this are:

- The filename contained invalid characters.
- The file contained a reference to a non-existent directory.
- The filename contained no directory information and the current directory does not exist.

The request fails.

User Action: Resubmit the request using a valid filename that references existing directories.

DECEDI_BADITMLST — Bad item List

Severity: Error.

Explanation: An item list passed to the call is invalid. Either the item list was empty when it should contain values, or some of the items in the item list are not supported by this request. The request will be ignored.

User Action:

1. Check all parameters used to build up the item list against those supported by the request.
2. A `decedi.err` file may have been created. Check the contents this file to get more details about the error.
3. Resubmit the request.

DECEDI_BADPARAM — Bad parameter

Severity: Error.

Explanation: A parameter, or a member of an item list passed as a parameter to the call is not supported. The request will be ignored.

If you are using the TCP/IP Network Interface, check the associated error message text, for further details as to the original problem.

For example:

```
Error : DECEDI_POST,  DECEDI_BADPARAM (e), Bad parameter
Connection_id <CVP_E> has not been configured
for BYPASS, SPLIT or MIXED
```

User Action:

1. Check all parameters against those specified in the application programming manual to ensure they are correct.
2. Resubmit the request.

DECEDI_CLIBADCMD — Command syntax incorrect

Severity: Error.

Explanation: The command specified was not of the format defined for the Digital DEC/EDI Client command line interface. The request will be ignored.

User Action:

1. Correct the command's syntax.
2. Enter the correct command.

DECEDI_CLIERROR — Command Line Interface Error

Severity: Error.

Explanation: The Application Client command line interface failed to perform the request and has reported the reason. The Application Client command line interface will exit.

User Action: None.

DECEDI_FETCHFAIL — Failed to fetch file `<fetch file number>`

Severity: Warning.

Explanation: The application client's command line request to fetch a file failed on the specified instance in the sequence. The reason for the failure will be reported along with this message. The other files in the fetch request will still be processed.

User Action: None.

DECEDI_INSUFVM — Insufficient Virtual Memory

Severity: Error.

Explanation: The application client could not fulfil the request because there is insufficient virtual memory in the caller's process.

This can be caused by either of the following:

- Bad parameters causing corruption
- Client processes having too little virtual memory

The request will be failed.

User Action: Check that the parameters in the request are correct. A common cause of this error is passing a string into the request that has not been null terminated.

Check that the client application has sufficient virtual memory in which to operate.

Resubmit the request.

DECEDI_INTERROR — Internal Error

Severity: Fatal.

Explanation: An internal error has occurred in the application client. The request will be failed.

User Action:

1. Check the file `decedi.err` for error information.
2. Contact your Digital Support Center, giving details of what the application client was attempting and the information from `decedi.err`.

DECEDI_MAPFAIL — Map Failed

Severity: Error.

Explanation: A file request failed mapping. Details of what the mapper found can be obtained from the mapper output files, if specified. Other files in the request will be processed.

User Action:

1. Examine the mapper output files for the cause of the problem.
2. Correct the problem.
3. Resubmit an amended request to obtain the unmapped objects for that file.

DECEDI_NOCLIENTLIC — No active client license was found

Severity: Error.

Explanation: The Digital DEC/EDI application license was not found or had expired. The request will be ignored.

User Action:

1. Obtain a valid Digital DEC/EDI application license.
2. Install it as specified in the installation manual.
3. Resubmit the request.

DECEDI_NOEDISYS — EDI Server System not running

Severity: Error.

Explanation: The Digital DEC/EDI system is not running on the Server. The request will be ignored.

User Action:

1. Contact the Digital DEC/EDI System administrator to get Digital DEC/EDI started.
2. Resubmit the request.

DECEDI_NOMAPOUTPUT — Map produced no output

Severity: Warning.

Explanation: The file requests map ran successfully but produced no output. Other files in the request will be processed.

User Action: None.

DECEDI_NOTAUTH — Not authorized to access the EDI Server System

Severity: Error.

Explanation: The client application making the request has not been registered with the Digital DEC/EDI Server as a valid user on this node. The request has been ignored.

User Action:

1. Get the Digital DEC/EDI System Administrator to register the application and node in Digital DEC/EDI.
2. Resubmit the request.

DECEDI_NOTFOUND — No objects found

Severity: Warning.

Explanation: A file request could not be satisfied because no objects matched.

This can occur if specific criteria are either used in the call to Fetch, or embedded in the map table which the call may reference.

The file request has been ignored. Other file requests in the request will be processed.

User Action:

1. Check that there are files to be fetched.
2. If there are files to be fetched, check that they match the selection criteria given.
3. Resubmit the request.

DECEDI_OPENINPERR — Could not open one of the input files

Severity: Error.

Explanation: One of the input files specified in the request could not be opened. The request has failed.

User Action:

1. Check that the file exists.
2. If a remote node then check that the application client information Server can access the file.
3. If a local node then check that the caller's process can access the file.
4. Resubmit the request.

DECEDI_OPENOUTERR — Could not open one of the output files

Severity: Error.

Explanation: One of the output files specified in the request could not be opened or created. The request fails.

User Action:

1. Make sure each file name specified in the request belongs in an existing directory.
2. If a remote node then check that the application client information Server can create files in that directory.
3. If a local node then check that the callers process can create files in that directory.
4. Resubmit the request.

DECEDI_PARTIALMAP — Map partially completed

Severity: Warning.

Explanation: A file request was only partially mapped. Details of what the mapper found can be obtained from the mapper output files, if specified. Other files in the request will be processed.

User Action:

1. Examine the mapper output files for the cause of the problem.
2. Correct the problem.
3. Resubmit an amended request to obtain the unmapped objects for that file.

DECEDI_POSTFAIL — Failed to post file `<post file number>`

Severity: Warning.

Explanation: The application client's command line request to post a file failed on the specified instance in the sequence. The reason for the failure will be reported along with this message. The other files in the post request will still be processed.

User Action: None.

DECEDI_SRVERROR — Server Error

Severity: Error.

Explanation: The Digital DEC/EDI Server could not perform the requested operation because of problems on the Server. The problems are reported on the Server node in the Digital DEC/EDI error log. The request will be failed.

User Action:

1. Contact the Digital DEC/EDI System administrator to correct the Server problem.
2. Resubmit the request.

DECEDI_WARNING — Partial completion, check individual returns

Severity: Warning.

Explanation: The application client request was completed, but some parts of it were not totally successful.

If individual file return status variables have been placed in the request to the application programming interface (this is done automatically with the command line interface) then the individual status returns indicate which file requests failed and the cause of that failure.

User Action: None.

DECEDI_ZEROMATCH — No objects were found that matched the selection criteria

Severity: Warning.

Explanation: There were no objects that matched the selection criteria in the track request.

User Action: If the result is not as expected:

1. Change the selection criteria.
2. Submit the new request.

Chapter 9 Mapper Error Messages

Mapper error messages are syntax errors reported during the parsing phase at compilation. These are annotated in the compilation listing file which has a file extension of `.lis`.

The error messages that follow are in alphabetical order. Each has a description of what caused the error and a corrective action you can take, if applicable.

Error Handling in the Compiler

When the Mapping Table Compiler detects an error it will display as many lines of the mapping table listing as are needed to establish some context and then issue an error message.

You can store the source file any time, even if it contains errors. However, a mapping table cannot be used at runtime until it has a clean compile.

Error Handling at Runtime

Runtime errors can occur due to errors in the table, errors in the data, or an incompatibility between the two. The Mapper runtime routines separate the errors into two types.

1. Soft Errors — Those that are isolated to a single document.
2. Hard Errors — Those that indicate that the file is corrupted or that the Mapper cannot detect the beginning of the next document boundary.

For Soft Errors, the Mapper aborts the current document and continues with the next. For incoming documents, the data is held back in Digital DEC/EDI

as an aborted document and requires manual intervention to reprocess. For outgoing documents, the document is skipped in the application file and the next document is processed.

For hard errors, the Mapper terminates and does not try to continue.

When the Mapper detects an error at runtime, and a SOFT ERROR or HARD ERROR is generated, the error message text is stored in the audit field ERROR_MESSAGE. If specified as a default runtime qualifier in the Mapping Table, a mail message is sent.

In addition, the Application Client call may specify the OUTPUT_FILE qualifier to specify an output file to which errors are logged. The DEBUG qualifier may also be used to specify a file to contain a log of the entire mapping process for this POST or FETCH request.

If a fatal error occurs in the Mapping process, the errors may be captured using the ERROR_LOG qualifier. This specifies a file to which information about the internal Mapping error is written.

If the error was something related to mapping, a few lines of the source table giving the assignment statement are printed to provide some hint of the context in which the error occurred.

There is a hook point that will be called when a soft error occurs. The hook has the option of continuing with the processing or converting the error to a hard error. A hook point is also called on a hard error. This is primarily for custom error reporting purposes.

Mapper Error Codes and Messages

DECEDI__ERROR

Explanation: An error occurred during processing. See the Digital Digital DEC/EDI error log file for more details.

DECEDI__INVPARAMS

Explanation: Invalid parameters were specified. Correct the parameters and re-execute FBR$DECEDI_EXTRACT from the Mapping Table Editor on the PC.

DECEDI__NOTAUTHORIZED

Explanation: A request has been received by the server for which it cannot find a matching entry in its authorization database for the application node from which the application is sending the request.

DECEDI__SUCCESS

Explanation: Normal, successful completion. Be sure to examine the log file to see if the document definitions themselves had any errors or inconsistencies.

DECEDI__UNKNDOC

Explanation: A document type was specified that is not known to the current Digital DEC/EDI installation.

DECEDI__UNKNELE

Explanation: This is a likely indication of a problem in Digital DEC/EDI's copy of the document definition.

DECEDI__UNKNSEG

Explanation: This is a likely indication of a problem in Digital DEC/EDI's copy of the document definition.

DECEDI__UNKNSTDVER

Explanation: An EDI standard and/or version unknown to Digital DEC/EDI was specified.

DECEDI__UNKNSUBELE

Explanation: This is a likely indication of a problem in Digital DEC/EDI's copy of the document definition.

DECEDI__TERMINATED_NODIR — Processing Terminated With Error

Explanation: This error message usually indicates an error on the command line. The error message specifies that the operation was unable to process the POST or FETCH command.

DECEDI__TERMINATED_RECEIVE — Processing Terminated With Error — RECEIVE

Explanation: The Mapper FETCH operation did not run to completion and terminated with an error.

DECEDI__TERMINATED_SEND — Processing Terminated With Error — SEND

Explanation: The Mapper POST operation did not run to completion and terminated with an error.

DECEDI__ABORT — Operation Aborted

Explanation: The Mapper runtime has detected a hard error and is aborting the current run. The error messages prior to this indicate what the problem is.

DECEDI__ALLOC_ERROR — Unable to Allocate Memory

Explanation: The Mapper runtime was attempting to allocate additional dynamic memory and the allocation failed. It is most likely that your working set size is set too low. See your system manager.

DECEDI__ALREADY_OPEN — Record Level Already Open

Explanation: The Mapper runtime is calling the record level initialization a second time in the same run. This is a Mapper program error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__API_ERROR — Error Received from Application Interface

Explanation: There was an error reported by Mapping Service as it sends or fetches a document. The previous message indicates the specifics of the error.

DECEDI__APPLID_ABSENT — No application specified

Explanation: The Mapper runtime is setting up for a POST or FETCH operation and found that the application ID was not specified. You can specify it by assigning a value to the global variable $APPLICATION in the initialization section of the Mapping Table. This can be overridden by using the Mapping Table Editor to specify an application ID. The application ID given must be one of those listed in the application ID section of the Mapping Table.

DECEDI__APPLID_MISMATCH — Mapping Table does not support specified application

Explanation: The Mapper runtime is setting up for the POST or FETCH operation and found an application ID that did not match one of those given in the application ID section of the Mapping Table.

DECEDI__AUD_BAD_RUN_ID — Invalid Run-id number in /var/adm/decedi/data/mapper_run_id.dat

Explanation: The Mapper is attempting to obtain a run ID number from the file /var/adm/decedi/data/mapper_run_id.dat and has found that the run number has been corrupted. Edit the file with a text editor and set the first 6 bytes to “000000” or to any number that is larger than any run ID number already in the audit database. This file should contain nothing other than this six digit number with the leading zeros.

DECEDI__AUD_CANNOT_RESTART — Cannot Restart this Mapper Run

Explanation: The Mapper recovery process is attempting to restart a run that was not complete, but is having some problem. A previous message will indicate what the problem is.

DECEDI__AUD_DBERROR — Error while updating the tracking database

Explanation: The Mapper audit facilities could not insert an event record in the audit database. A previous RDB message should indicate the problem.

DECEDI__AUD_MISSING_DB_FILE — An Audit Rdb Database file is missing in decedi_db

Explanation: The Mapper audit facilities could not locate the audit database. It should be located in the directory identified by the system logical decedi_db. See your system manager.

DECEDI__BADPARAM — Invalid Argument Value on Argument

Explanation: The Mapper runtime is just starting up and is evaluating the command line qualifiers or the arguments from the callable interface. It has found a qualifier with an invalid value.

DECEDI__BAD_HOOK_STAT — Abnormal Return from Hook

Explanation: The Mapper runtime has just called a customization routine and is evaluating the return value and status. If the return status is anything other than the values defined for SUCCESS and UNDEFINED, it is considered an error.

DECEDI__BLD_CANNOT_CLOSE_FILE — Cannot close the file

Explanation: The Mapper could not close one of the temporary work files during Digital DEC/EDI extract process. Retry the extract.

DECEDI__BLD_CANNOT_OPEN_EDI_INPUT — Cannot open the specified EDI input file

Explanation: An error occurred opening a previously generated extract listing file. Either the filename specification is in error, the specified file does not exist or a privilege violation has occurred.

DECEDI__BLD_CANNOT_OPEN_OUTPUT — Cannot create the parse output file

Explanation: An error occurred creating or opening either the output listing file or the log file. Either the filename specification is in error or a privilege violation has occurred.

DECEDI__BLD_INCOMPATIBLE_FILES — Standard or Version of Digital DEC/EDI files differs

Explanation: The Mapper found an inconsistent set of reports in appended extract files during processing. Retry the extract.

DECEDI__BLD_NO_DOC_DEFN — Digital DEC/EDI didn’t find the requested document definition

Explanation: The Mapper received a blank report from Digital DEC/EDI. This indicates that the requested document was not in Digital DEC/EDI. Check that you have the correct document name and version number.

DECEDI__CANNOT_OPEN_SOURCE — Cannot open the Mapping table source file

Explanation: The Mapper compiler needs to read the listing file generated by the report writer. Be sure you have write permission into your local directory for the report writer.

DECEDI__CVT_ERROR — Data Conversion Error

Explanation: The Mapper runtime is evaluating a field reference in an expression and the value in that field does not match the format for the data type of the field. This is usually an indication of bad data.

DECEDI__DEC_OVF — Decimal value overflows specified data size

Explanation: The Mapper runtime is assigning a numeric string value to a data label or field on a record and the size of the data exceeds the size of the space defined for the field or data label. This is usually an indication of bad data.

DECEDI__DIVBY_ZER — Division by zero

Explanation: The Mapper runtime is evaluating a division operator in an expression and found that it would be dividing by zero. This is usually an indication of bad data.

DECEDI__DOCTYPE_MISSING — The FBdoctype for this document was not specified in $DOCTYPE

Explanation: The Mapper runtime is attempting to send a document, and it found that the $DOCTYPE global variable was not assigned a value indicating the FBdoctype to use for the document. In addition, the Mapping Table Editor was not used to specify any overrides.

DECEDI__DOCUMENT_ERROR — Hook Requests Abort of the Current Document

Explanation: The Mapper runtime has just called the customization routine at the END DOCUMENT hook point. The return status was UNDEFINED, indicating that the document is to be aborted.

DECEDI__DOC_DEF_MISSING — No Mapping set matching specified Doc. Type and Partner ID

Explanation: The Mapper runtime is processing a document and is attempting to identify which mapping set to use. However, the doctype and partner for the document do not match any of the mapping sets. Enable the mapping log listing, and it will show you the values that it was trying to use. You might be using the wrong table.

DECEDI__DOC_NO_OUTPUT — Document generated nothing in Output File

Explanation: The Mapper runtime has processed all of the mappings for a document, but no records or segments were produced as a result. This is an indication that something might be wrong with the maps or that the data is bad.

DECEDI__EOF — No more records in the stream

Explanation: This is an internal flag, not an error condition.

DECEDI__ERRORS — Completed but One or more errors detected (SOFT ERRORS)

Explanation: The Mapper runtime has completed a run, but during the run documents were aborted because of soft errors. The preceding error messages should indicate the cause of the soft errors.

DECEDI__ERROR_IN_FILE_SECTION — Error in the File Section

Explanation: The parser was unable to find a keyword to identify where the file section of the table begins. This is a Mapper internal error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__EXP_ERROR — Error in Expression

Explanation: The Mapper runtime cannot process an expression in a map. Check the syntax of the expression.

DECEDI__FOR_PARM_ERROR — Error in FOR parameter

Explanation: The Mapper runtime is executing a FOR loop in a mapping expression, and found that one of the parameters (starting value, ending value, or increment) in the FOR is invalid.

DECEDI__GROUP_IMBAL — Probable bad data from DECEDI — excess $ENDGROUPs

Explanation: The Mapper runtime is accepting data labels for a document from Digital DEC/EDI and found that the number of $GROUPS did not match the number of $END-GROUPS. This is an indication of bad data.

DECEDI__ILL_INST — Illegal Instance specification

Explanation: The Mapper runtime is evaluating a record or segment qualification and found an instance specification that was invalid. The instance number must be greater than 0.

DECEDI__ILL_NDX — Illegal Subscrip

Explanation: The Mapper runtime routine is evaluating a field reference, and found a subscript with an illegal value. The value must be greater than 0 and less than or equal to the size of the array.

DECEDI__ILL_RANGE — Lower bound in NEXT LIST Range greater than upper bound

Explanation: The Mapper runtime is evaluating the NEXTLIST specification in the repeat pattern field of a map. It has found that an element in the list contains a range and that the first element of the range is larger than the second element of the range.

DECEDI__ILL_$NEXT — Functions $NEXT, $FIRST, $TESTNEXT can appear only in a Map

Explanation: The Mapper runtime is evaluating an expression that is someplace other than a map, and it found one of the NEXTLIST related functions. These are allowed only in maps where the NEXTLIST has been declared.

DECEDI__INCOMING_REC_MISSING — Required record not generated in Application File

Explanation: The Mapper runtime is processing an incoming document. The mappings did not generate a required record. This is a record that is defined in the record sequence section of the table with a MIN greater than 0.

DECEDI__INPUT_CLOSE_ERROR — Error on Close of Input File

Explanation: The Mapper runtime has completed the read of an application file and is attempting to close the file. The file might have been corrupted.

DECEDI__INPUT_OPEN_ERROR — Unable to Open Input File

Explanation: The Mapper runtime is setting up to send documents, but cannot open an application file. Check that the file name is correct and that the file is accessible.

DECEDI__INPUT_READ_ERROR — Error on Read of Input File

Explanation: The Mapper runtime is reading in records from an application file, and encountered a read error. This is usually a disk I/O error. See your system manager.

DECEDI__INPUT_RMSERR — Error Received While Evaluating Input File Specification

Explanation: The Mapper runtime is checking for wildcard characters in the application file specification. This is an indication that the file specification (value in $FILENAME) is bad or contained illegal characters.

DECEDI__INTERNAL_ERROR — Mapping Table Internal Error

Explanation: The Mapper runtime has encountered a problem or situation that should not have happened. This is usually a Mapper programming error, although it could be caused by bad data. Submit a Software Problem Report (SPR) to Digital.

DECEDI__INVHIERKY — Data does not agree with Specified Source Hierarchy

Explanation: The Mapper runtime is reading data into the source tree and has found a record or segment (depending on direction) that does not fit correctly in the source tree. This can be caused by records being out of order or too many records or segments (MAX is exceeded). This is normally an indication of bad data. In the incoming direction, it can be caused by misplaced $GROUP/$END-GROUP labels or a mismatch between Digital DEC/EDI tables and the extracted document definitions.

DECEDI__INVRECTYP — Application Record has Invalid Record Type

Explanation: The Mapper runtime is reading application file records and it cannot determine the record type of a record it just read. None of the recognition expressions (from the record sequence attributes sections of the table) found a match. That is, they all evaluated to FALSE. This is

an indication of bad data, the wrong table file, or a record type that was not specified in the table.

DECEDI__INV_LABEL — Invalid or undefined Label

Explanation: The Mapper runtime is reading in data labels and data value pairs from Digital DEC/EDI (incoming document). It has found a data label it does not recognize. This most likely occurs when Digital DEC/EDI has changed its document definitions since the last extract. Perform another extract and adjust the mappings to deal with the new data labels.

DECEDI__INV_LOCAL — Invalid format of data in FBR$LOCAL_TEST_IN file

Explanation: The format of the internal-format file provided as input in local test mode does not have the correct format.

DECEDI__INV_TBL — Mapping Table is invalid or not compiled

Explanation: The Mapper runtime is starting up and the table that was specified did not have the compiled data in it. If you modified (or even looked at) the table using the Mapping Table Editor since the last compile, it must be compiled again before it can be used by the runtime.

DECEDI__LOGERR — Unable to Open Log File

Explanation: The Mapper runtime opens a temporary log file to capture error messages during a run. This error occurs if the Mapper cannot create the temporary log file. Make sure you have write access to the default directory.

DECEDI__MAP_EXCD_LIMIT — Exceeded limit on output of particular record or segment

Explanation: The Mapper runtime is processing maps and tried to generate more records or segments than specified as the MAX or LIMIT number. This error only occurs if the ERROR ON MAX EXCEEDED flag is set in the map. Otherwise, the map iteration loop terminates when the MAX or LIMIT is reached and no error is generated. Normally, the map loop terminates when it runs out of source data.

DECEDI__MAP_OUT_OF_DATA — Ran out of source data of a required type during mapping

Explanation: The Mapper runtime is processing maps and ran out of source data. This error only occurs if the ERROR ON OUT-OF-DATA flag is set in the map. Otherwise, the map iteration loop terminates and no error is generated. This is the normal situation, but there can be cases where running out of data before all of the maps have been generated is an indication of an error. This is where the ERROR ON OUT-OF-DATA flag is useful.

DECEDI__MISSING_APPLICATION — APPLICATION name invalid or missing

Explanation: The Mapper compiler is attempting to compile a table with a missing application in the table attributes section. Edit the table and enter at least one application ID.

DECEDI__MISSING_BOUND — INCOMING or OUTGOING direction is not specified

Explanation: The Mapper compiler is attempting to compile a table with a direction not specified. Edit the table and enter the direction.

DECEDI__MISSING_ENDVARIANTS — Missing the END VARIANT keywords

Explanation: The Mapper compiler is attempting to compile a table with a missing END VARIANT for a VARIANT keyword. This could be either in the record sequence or record layout section of the table. Edit the table and add the missing keyword.

DECEDI__MISSING_FIELDS — No fields defined for this record

Explanation: The Mapper compiler is attempting to compile a record layout without field definitions. Edit the files section, record layout of the table and add the missing field definitions.

DECEDI__MISSING_FIELD_SCALE — Missing field scale factor

Explanation: The Mapper compiler is evaluating the field attributes and found a keyword SCALE without a factor. Edit the files section, record layout of the table and add the missing scale factor.

DECEDI__MISSING_FIELD_SIZE — The Field size is missing

Explanation: The Mapper compiler is evaluating the field attributes and found a keyword SIZE without a size specified. Edit the files section, record layout of the table and add the missing size specification.

DECEDI__MISSING_MODE — Test Mode Must be specified

Explanation: The Mapper compiler is attempting to compile a table without a test mode specification. Edit the table attributes section and enter the test mode.

DECEDI__MISSING_RECTYPE — Missing record type name

Explanation: The Mapper compiler has found the keyword, RECORD, but without a name following it. This is a Mapper internal error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__MISSING_SEGMENTS — No Segments defined for this document definition

Explanation: The Mapper compiler is evaluating a document definition and did not find any segments associated with it. Since these are extracted from Digital DEC/EDI, this is a program error in the Digital DEC/EDI extract part of the UI. Submit a Software Problem Report (SPR) to Digital.

DECEDI__MISSING_SEMI — Semicolon missing from end of previous line

Explanation: The Mapper compiler could not find the end of the statement. It is most likely a missing semicolon (;). Edit the table and add the semicolon or look for invalid syntax.

DECEDI__MISSING_SUBSCRIPT — Array subscript missing or invalid

Explanation: The Mapper compiler found the ARRAY or OCCURS keyword but did not find a numeric value for the size of the subscripts. The Mapper will accept up to three subscripts separated by a space. The array subscript specification might also be given as lower bound and upper bound separated with a colon (:).

DECEDI__MPFL_OPEN_ERROR — Unable to open Map Debug File

Explanation: The Mapper runtime is starting up and is creating the map_debug.log file. Make sure that you have write access to the default directory.

DECEDI__NEXT_LIST_ERR — $NEXT, $FIRST, or $TESTNEXT found in Map with no NEXT LIST

Explanation: When evaluating the functions $NEXT(), $FIRST(), or $TESTNEXT() it was found that there was not NEXT LIST specified in the repeat pattern. The NEXT LIST is required when using these functions.

DECEDI__$NEXT_IN_LIST — $NEXT, $FIRST, $TESTNEXT not allowed in NEXT LIST element

Explanation: The Mapper runtime is evaluating the NEXTLIST elements in a repeat pattern part of the map, and has found an element that contains a $NEXT(), $FIRST(), or $TESTNEXT() function call. This recursive use of the NEXTLIST is not allowed.

DECEDI__NONALPHA — Non-Alphabetic Character assigned to ALPHABETIC field

Explanation: The Mapper runtime is making an assignment to a record field that has been declared as an ALPHABETIC data type. In this case, the data being assigned contains a non-alphabetic character.

DECEDI__NO_LOCAL_TEST — Could not create the inhouse file for LOCAL TEST

Explanation: Could not create the inhouse file for outgoing LOCAL TEST mode. It is most likely that the file is not valid, or references a directory for which the user does not have write permissions.

DECEDI__NO_OUTPUT — Mapper session generated no output

Explanation: The Mapper runtime has completed all of the maps with no errors and has processed all of the records in the application file (outgoing) or has processed all data labels available from the Digital DEC/EDI Application Server (incoming), and yet has not been able to generate a single document. This is usually an indication of bad data, the wrong table, or incorrect maps.

DECEDI__NOT_OPEN — Record Interface not Properly Opened

Explanation: The Mapper runtime is trying to continue processing of a document when the record interface encountered an error. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__OUTGOING_SEG_MISSING — Required segment not generated in Document

Explanation: The Mapper runtime is processing maps and found that a required segment was not generated. The maps must generate the minimum number of segments specified in the segment sequence section of the document.

DECEDI__OUTPUT_CLOSE_ERROR — Error on Close of Output File

Explanation: The Mapper runtime has finished writing to the application file (incoming direction) and is attempting to close the file. This is an indication that there might have been an I/O error. The map_debug.log file will contain more information about the attempted file close.

DECEDI__OUTPUT_OPEN_ERROR — Error on Open of Output File

Explanation: The Mapper runtime is attempting to create the application (incoming direction) and has received an error. Make sure that the specified directory has write permission. The map_debug.log file will contain more information about the attempted file open.

DECEDI__OUTPUT_WRITE_ERROR — Error on Write of Output File

Explanation: The Mapper runtime is writing to the application file (incoming direction) and encountered a write error. Make sure the device has enough space and try again. The map_debug file will contain more information about the attempted file write.

DECEDI__PARSE_ERROR — Error During parse of Extract

Explanation: Means the parse got lost. look for a syntax error in the general area.

DECEDI__PARTNER_MISSING — The Partner for this document was not specified in $PARTNER

Explanation: The Mapper runtime is attempting to send a document but the $PARTNER global variable was not given a value, and the * partner id has not been specified using the Mapping Table Editor. The Mapper does not know what partner to send it to.

DECEDI__POLL_EMPTY — No mapping sets in Mapping Table match specified Partner+DocType

Explanation: The Mapper runtime is setting up to receive documents, and is trying to find the partner and Digital DEC/EDI internal doctype to use as document selection criteria. To do this, it finds all of the mapping sets that match the Mapper doctype and partner given in the global variables $DOCTYPE_SELECT and $PARTNER_SELECT or overridden with * DOCUMENT TYPE of PARTNER ID qualifiers specified either with the Mapping Table Editor or their default values in the table. In this case, the Mapper found no mapping sets that matched the selection criteria. See the map_debug log file for more information.

DECEDI__P_BAD_ALIGN — Invalid Alignment Keyword

Explanation: The Mapper compiler found the keyword, ALIGN, but the alignment qualifier keyword is wrong. Edit the files section, record layout of the table.

DECEDI__P_BAD_JUST — Invalid justification Keyword

Explanation: The Mapper compiler found the keyword, JUSTIFIED, but the qualifier keyword is wrong. Edit the files section, record layout of the table.

DECEDI__P_BAD_OCCURS — Invalid OCCURS specification

Explanation: The Mapper compiler found the OCCURS keyword but did not find a numeric value for the size of the subscripts. It only accepts one numeric value. Edit the files section, record layout of the table.

DECEDI__P_INV_DATATYPE — Invalid Data Type

Explanation: The Mapper compiler is evaluating the field attributes and found a keyword that does not match any of the data types. Edit the files section, record layout.

DECEDI__P_INV_EXP — Invalid expression operand

Explanation: The Mapper compiler is evaluating an expression and found an illegal operator or illegal mathematical construction. Edit the map section or the initialization of the table.

DECEDI__P_INV_INST — Invalid Instance Specification

Explanation: The Mapper compiler is evaluating a field reference and found the brace indicating an instance qualifier but the syntax inside the braces is wrong. Edit the map section of the table.

DECEDI__P_INV_LEVEL — Invalid Level Number

Explanation: The Mapper compiler is evaluating either the record sequence section or the record layout section and found a level number out of place. Level numbers must be either the same as, one greater than, or one less than the one above. You cannot skip numbers.

DECEDI__P_INV_NAME — Invalid Name Syntax

Explanation: The Mapper compiler is evaluating a record name or a field name, and found an illegal character. Edit the files section, record layout of the table.

DECEDI__P_INV_NBR — Invalid Syntax for a Numeric Constant

Explanation: The Mapper compiler is evaluating a numeric constant and found an illegal character.

DECEDI__P_INV_PATT — Invalid Repeat Pattern Specification

Explanation: The Mapper compiler is evaluating a map and found an unrecognized keyword in the repeat pattern specification. Edit the map section of the table.

DECEDI__P_SIZE_REQ — Field SIZE is required

Explanation: The Mapper compiler is evaluating the field attributes and found a data type which required a size without a size specified. Edit the files section, record layout of the table and add the missing size specification.

DECEDI__QUALIFIER — Invalid or Missing Qualifier Value or Keyword

Explanation: The Mapper runtime is starting up and is evaluating the command line qualifiers. It has found one that has an invalid or missing value.

DECEDI__RECORD_SHORT — Attempt to access data beyond end of application file record

Explanation: The Mapper runtime is evaluating a field reference in a mapping expression and found that the position of the field is beyond the current length of the record. This is likely to be an indication that the data has been corrupted. If you are using the RECORD or SWITCH hooks, be sure to use VARYING_STRING data type of the argument passing the $RECORD global variable, or make sure all records have the same size and pass as a TEXT data type.

DECEDI__RECOVERY_AMBIGUOUS — Last Document May Not Be Processed

Explanation: The Mapper recovery process has found that the incomplete run terminated between the END DOCUMENT and COMMIT DOCUMENT events. This means that the Mapper does not know if the Digital DEC/EDI end-send call for this document was complete. Check the Digital DEC/EDI tracking log and see what documents were sent, and then manually restart the run.

DECEDI__RECOVERY_NO_ACTION — No recovery processing necessary

Explanation: The Mapper recovery process did not find any incomplete runs in the audit database.

DECEDI__RT_HOOK_NOLINK — Could not link Custom Function

Explanation: The Mapper runtime program is starting up and is trying to dynamically link to all of the images holding the customization routines. It has some problem finding the shared image file or in finding the entry points in the shared image file. Be sure the entry points are linked with the UNIVERSAL keyword. Check the customization routine declaration to make sure the file name for the shared image file is correct and that you have read access to the files.

DECEDI__RT_INVALID_ARG — Invalid Function argument

Explanation: The Mapper runtime is evaluating a built-in function and found that one of its arguments was not valid.

DECEDI__RT_INVALID_DATE — Invalid Date format for data in $DATE() conversion

Explanation: The Mapper runtime is evaluating the $DATE function and the date conversion encountered a conversion error.

DECEDI__STACK_ERROR — Mapping Table Internal Error — Stack not empty after expression

Explanation: This indicates that the run-time interpreter stack was corrupted. Submit an SPR.

DECEDI__STACK_OVERFLOW — Mapping Table Internal Error — Stack overflow

Explanation: This indicates that the run-time interpreter stack was corrupted. Submit an SPR.

DECEDI__STACK_UNDERFLOW — Mapping Table Internal Error — Stack underflow

Explanation: This indicates that the run-time interpreter stack was corrupted. Submit an SPR.

DECEDI__SYNTAX_ERROR — Parser Syntax Error

Explanation: The Mapper compiler cannot evaluate the table. This error is usually due to a misplaced or misspelled keyword or illegal syntax.

DECEDI__TABLEFILE_CLOSE_ERROR — Error on Close of the Mapping Table File

Explanation: The Mapper has completed processing and has encountered an error while closing the table file. This is an indication of a disk I/O error condition. See the system manager.

DECEDI__TABLEFILE_OPEN_ERROR — Unable to Open the Mapping Table File

Explanation: The Mapper is starting up and has encountered an error while attempting to open the Mapping Table file specified in the command line. Make sure the table filename is correct and that you have

read access to the file. You will also get this error if you cannot create the new version of the file. Be sure that you have write access in the directory in which the file resides.

DECEDI__TABLEFILE_READ_ERROR — Error on Read of the Mapping Table File

Explanation: The Mapper is starting up and has encountered a read I/O error while reading in data from the Mapping Table file. This is an indication that the table file was corrupted (or not a table file). Try recompiling the table.

DECEDI__TABLEFILE_WRITE_ERROR — Error on Write of the Mapping Table File

Explanation: The mapper is creating a new version of the table file and encountered an I/O error writing to the file. This is most likely a disk I/O error problem. See your system manger.

DECEDI__TBL_IN — Cannot Send using an INCOMING Mapping Table

Explanation: The Mapper runtime is starting up a POST run and found that you have specified a Mapping Table file setup for the INCOMING direction. Make sure you have the correct table.

DECEDI__TBL_OUT — Cannot Receive using an OUTGOING Mapping Table

Explanation: The Mapper runtime is starting up a FETCH run and found that you have specified a Mapping Table file setup for the OUTGOING direction. Make sure you have the correct table.

DECEDI__TERMINATED — Processing Terminated With Error

Explanation: The Mapper runtime has encountered a hard error and is terminating (or returning from the callable interface). The previous error message indicates what the problem is.

DECEDI__TG_AMBIG_VAR_REF — Field referenced via Variant name different in two or more alternatives

Explanation: References to a field with the same name on two variant records is not the same in size, type, and position. Qualify the field name

with the variant record name so it is obvious as to which field is being referenced.

DECEDI__TG_BAD_FLD_REF — Unknown name referenced

Explanation: The Mapper compiler is evaluating an expression and found a name it did not recognize. This name could be a field on a record, global variable, or a data label. Look for a misspelling or an global variable that should have been defined in the initialization section of the table.

DECEDI__TG_BAD_FOR_EACH_REF\FOR EACH record or segment not in line of ascent

Explanation: The Mapper compiler is evaluating a repeat pattern in a map and has found the FOR EACH keywords with a reference to a record or segment that is not the parent or in the current path of the record or segment specified in the SET CONTEXT field.

DECEDI__TG_BAD_GLOBAL_ASST — Global Variable must not be subscripted or qualified

Explanation: The Mapper compiler is evaluating an expression and has found a reference to a global variable with an array subscript or a record or segment qualifier attached.

DECEDI__TG_BAD_GV — Assignment to unknown Global Variable

Explanation: The Mapper compiler is evaluating an assignment statement in a map and has found a reference to a name on the left side of the assignment, which it does not recognize. This could be a misspelling or an uninitialized global variable.

DECEDI__TG_BAD_LKUP_REF — Unknown Lookup Table referenced

Explanation: The Mapper compiler is evaluating an expression and found a $LOOKUP function. The first argument is not the name of one of the lookup tables. You might have forgotten to define a lookup table.

DECEDI__TG_BAD_MANY — MANY may be specified for the last subscript only

Explanation: The Mapper compiler is evaluating an array specification in the record layout section and found the keyword MANY as the size of the array. This is only permissible on a field at the end of a record.

DECEDI__TG_BAD_REF — Unknown record or segment referenced

Explanation: The Mapper compiler is evaluating an expression and found a qualifier on a field reference for a record or segment that is unknown. It could be a misspelling, or it could be a missing record definition in the record sequence.

DECEDI__TG_BAD_SIZE — Size must be positive number

Explanation: The Mapper compiler is evaluating a field attribute in the record layout section and found a size that is zero or a negative number.

DECEDI__TG_BHDR_BEGTRM_CNFL — BATCH HEADER and BEGINS or TERMINATES on same record

Explanation: The keywords BATCH HEADER and BEGINS or TERMINATES are mutually exclusive. Edit the record sequence attributes of the table.

DECEDI__TG_BKDOC_FLD_UNRECOG — BREAK field not recognized

Explanation: The Mapper compiler is evaluating the record sequence attributes and found a BREAK field that is not defined in the record layout section associated with this record sequence entry.

DECEDI__TG_BKON_NODEF — UNTIL with no default record or segment specified

Explanation: The Mapper compiler is evaluating a map repeat pattern specification and found the keyword UNTIL, but no record or segment specified in the SET CONTEXT field.

DECEDI__TG_BKON_NOSRC — UNTIL with Set Context not specified

Explanation: An “UNTIL field CHANGES” clause was specified in the repeat pattern but no context was specified in the set context.

DECEDI__TG_CUST_ARG_ARY — Argument of Custom Function cannot be array

Explanation: The Mapper compiler is evaluating an expression and found a customization function with an argument passed to the function that is the name of an array. You must qualify the name with the subscript indicating which element you are passing.

DECEDI__TG_CUST_OPT_RET — Return-value argument cannot be optional

Explanation: The Mapper compiler is evaluating an expression for a function called at one of the hook points where the Mapper expects a return value. The customization function did not have one of its arguments defined as a return value.

DECEDI__TG_CUST_RET_GT_NARGS — Return-value index greater than number of arguments

Explanation: The Mapper compiler is evaluating a declaration of a customization function. The index for the return value does not identify one of the defined arguments. The first argument is index number 1.

DECEDI__TG_CUST_VAL_RET — Return-value argument cannot be designated By-Value

Explanation: The Mapper compiler is evaluating a declaration of a customization function. The argument identified as the return value is passed by value, which does not return anything. Change it to passed by reference or by descriptor.

DECEDI__TG_CUST_VAL_SIZE — By-Value argument size must not exceed 4 bytes

Explanation: The Mapper compiler is evaluating a declaration of a customization function. The size of the argument being passed by value is larger than the DEC calling standards allow.

DECEDI__TG_DEF_UNRECOG — Unknown default specified

Explanation: The Mapper compiler is evaluating the run time defaults section of the table. It found an unknown qualifier. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_DMY_ATTRIB — No attributes permitted if no record associated

Explanation: The Mapper compiler is evaluating the record sequence section of the table. It found a sequence entry with attributes specified indicating MIN, MAX, and so forth. However, no corresponding record was found in the record layout section.

DECEDI__TG_DST_UNRECOG — Unknown destination specified

Explanation: The Mapper compiler is evaluating a map and found the destination record or segment that does not match with the destination hierarchy. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_DUP_ALIGN — Alignment specified more than once for structure

Explanation: The Mapper compiler is evaluating record layout attributes. It found more than one attribute for alignment.

DECEDI__TG_DUP_ARRAY — More than one array specified for same data item

Explanation: The Mapper compiler is evaluating record layout attributes. It found more than one array declaration for the field.

DECEDI__TG_DUP_ASN — Assignments specified twice

Explanation: There were more than one set of assignments for $DOCTYPE and $PARTNER in the record attribute section. This is an indication of a programming bug, submit an SPR.

DECEDI__TG_DUP_BEG_DOC — BEGINS DOCUMENT specified twice for record

Explanation: The Mapper compiler is evaluating record sequence attributes. It found two BEGINS DOCUMENT keywords. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_DUP_BHDR — BATCH HEADER specified twice for record

Explanation: The Mapper compiler is evaluating record sequence attributes. It found two BATCH HEADER keywords. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_DUP_BKDOC — BREAKON specified twice for record

Explanation: The Mapper compiler is evaluating record sequence attributes. It found two BREAKON keywords. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_DUP_DOCDEF — Two mapping sets have same INT. DOCTYPE and PARTNER ID

Explanation: The Mapper compiler is evaluating the mapping set definitions for an incoming table and found two mappings for the same Digital DEC/EDI internal doctype and partner. It is ambiguous as to which mapping set to use. Delete one of them.

DECEDI__TG_DUP_FLDTYPE — Type specified more than once for field

Explanation: The Mapper compiler is evaluating record layout attributes. It found the data type specified more than once for the same field.

DECEDI__TG_DUP_FLD_NAME — Field or structure has same name as preceding entity in group

Explanation: The Mapper compiler is evaluating the record layout and found the field name has already been used.

DECEDI__TG_DUP_FLOATING — FLOATING specified twice for segment

Explanation: The Mapper compiler is evaluating the document definition. It found the keyword, FLOATING, twice for the same segment. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_DUP_JUST — Justification specified more than once for field

Explanation: The Mapper compiler is evaluating record layout attributes. It found a justification attribute specified more than once for the same field.

DECEDI__TG_DUP_LBLTYPE — Type specified more than once for label

Explanation: The Mapper compiler is evaluating the document definitions and found a data type specified twice for this data label. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_DUP_LBL_IN_DOC — Label has same name as a preceding label in this Document

Explanation: The Mapper compiler is evaluating the document definitions and found a data label defined more than once for the same name.

DECEDI__TG_DUP_LKUP_DEF — Lookup Table has same name as previously specified table

Explanation: The Mapper compiler is evaluating the lookup table section. It found a table with the same name as the previous table. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_DUP_OCC — OCCURS specified more than once for record or segment

Explanation: The Mapper compiler is evaluating record layout attributes. It found an OCCURS specification or an array attribute specified more than once for the same field.

DECEDI__TG_DUP_REC_NAME — Record/segment has same name as another in File or Document

Explanation: The Mapper compiler is evaluating record sequence or document definition and found that the record name or the segment name is not unique. Record names cannot be the same as segment names.

DECEDI__TG_DUP_SEG_IN_DOC — Segment type has same name as a preceding Segment type

Explanation: The Mapper compiler is evaluating the document definition and found that the segment name is not unique. This is a Mapper programming error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_DUP_SRC — Same record or segment designated twice in Context

Explanation: The Mapper compiler is evaluating the SET CONTEXT field of a map and found the same record or segment specified twice. If more than one record or segment is specified in the SET CONTEXT field, they cannot be subordinate to each other.

DECEDI__TG_DUP_SRC_VAR — Records or segments in same variant designated in Context

Explanation: The Mapper compiler is evaluating the SET CONTEXT field of the map and found two records that are part of the same variant. Only one record of a variant can be mentioned in a SET CONTEXT field.

DECEDI__TG_DUP_TRM_DOC — TERMINATES DOCUMENT specified twice for record

Explanation: The Mapper compiler is evaluating the record sequence attributes and found more than one TERMINATES DOCUMENT keyword. This is a Mapper error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_EMPTY_REC — Record type specified with zero length

Explanation: The Mapper compiler is evaluating record layout section and found a record with no fields with a SIZE attribute specified.

DECEDI__TG_EMPTY_VARIANT — No records/segments specified for VARIANT

Explanation: The Mapper compiler is evaluating the record sequence fields and found a VARIANT and END VARIANT keywords, but no records were enclosed.

DECEDI__TG_FLTNG_IN_Application — Record may not have FLOATING attribute

Explanation: The Mapper compiler is evaluating the record sequence attributes and found the keyword FLOATING. The keyword FLOATING is designed only for segments.

DECEDI__TG_FLTNG_SUBORD — Floating segment may not have subordinates

Explanation: The Mapper compiler is evaluating a document definition and found the keyword FLOATING on a segment that had subordinate segments. This is an illegal structure.

DECEDI__TG_FOR_IN_INITS — FOR clause not allowed in Initializations

Explanation: The Mapper compiler is evaluating the initialization section and found the FOR keyword. The FOR clause is reserved for mapping expressions.

DECEDI__TG_ILL_BEG_DOC — BEGINS DOCUMENT specified twice in line of descent

Explanation: The Mapper compiler is evaluating the record sequence attributes and found a BEGINS DOCUMENT keyword on a record that is subordinate to another record that was also marked with the keyword BEGINS DOCUMENT. You cannot put a document inside of another document.

DECEDI__TG_ILL_BHDR — BATCH HEADER precedes BEGINS DOCUMENT in line of descent

Explanation: The Mapper compiler is evaluating the record sequence attributes and found the keyword BATCH HEADER marking a record inside of a document. The keyword BATCH HEADER means that it is a record outside of a document.

DECEDI__TG_ILL_FLTNG — Floating segment has prohibited attribute

Explanation: The Mapper compiler is evaluating a document definition and found attributes associated with records. This is a Mapper error. Submit a Software Problem Report (SPR) to Digital.

DECEDI__TG_ILL_FN_REF — Improper function reference

Explanation: The Mapper compiler is evaluating an expression and found a name followed by open and close parentheses, indicating that it is a function, but it is not one of the built in functions and it is not one of the customization functions that have been declared.

DECEDI__TG_ILL_INC — Data in same line of descent specified for incrementing

Explanation: The Mapper compiler is evaluating the SET CONTEXT field of a map and found two fields or segments specified where one is subordinate to the other.

DECEDI__TG_ILL_INSTANCE — Only 1 instance allowed here

Explanation: The instance specifier was not valid or missing in a field reference that used { }'s.

DECEDI__TG_ILL_JUST — Justification may not be specified with present type

Explanation: The Mapper compiler is evaluating the record layout attributes and the data type is not compatible with justification.

DECEDI__TG_ILL_QUAL — Qualification not allowed on present item

Explanation: The Mapper compiler is evaluating an expression and found a record or segment with an illegal qualification.

DECEDI__TG_ILL_RANGE — RANGE allowed only in NEXTLIST

Explanation: The Mapper compiler is evaluating an expression and found a an array index specifying a range but is not in the list associated with a NEXTLIST repeat pattern specification.

DECEDI__TG_ILL_TIMES — TIMES value must be positive number

Explanation: The Mapper compiler is evaluating the repeat pattern field of a map. It has found a REPEAT n TIMES specification, but the value is not greater than 0.

DECEDI__TG_IMBEDDED_MANY — Variable-length field must be last field in record

Explanation: The Mapper compiler is evaluating the record layout section of the table. It has found a field with an array specification and an array size of MANY, indicating that it is variable length. However, to be variable length the field must be the last of a record and in this case, it is not.

DECEDI__TG_INC_DOC_PARMS — Illegal parameter for INCOMING table

Explanation: The Mapper compiler is evaluating the record sequence attributes and has found attributes associated with an OUTGOING direction.

DECEDI__TG_INC_UNRECOG — Unknown source specified

Explanation: The Mapper compiler is evaluating the SET CONTEXT field of a map and found an unrecognized record or segment name.

DECEDI__TG_INST_ON_FLD — Instances not allowed on fields, only on records/segments

Explanation: The Mapper compiler is evaluating an expression in a map and has found braces *{}* indicating an instance specification as a qualifier on a field. See the manual for complete field reference syntax.

DECEDI__TG_MAP_DOC_NOMATCH — Map Set does not match any Document specified for Table

Explanation: The Mapper compiler is evaluating a mapping set specification, but cannot find a document specification that matches. Maybe you deleted the document definition after you created the mapping set.

DECEDI__TG_MAP_UNDER_DUMMY — Map defined with no parent map

Explanation: The Mapper compiler is evaluating a mapping set. It found that a map was defined in the index of maps, but its parent does not have a map. This map could never be executed.

DECEDI__TG_MISMATCH_FOR_EACH — Context items relate differently to FOR EACH item

Explanation: The Mapper compiler is evaluating a repeat pattern specification of a map. It found a FOR EACH pattern, but more than one record or segment is specified in the SET CONTEXT field. The FOR EACH pattern only works with one record or segment in the SET CONTEXT specification.

DECEDI__TG_NO_REC_DEFS — No Record types defined

Explanation: The Mapper compiler is evaluating the record sequence section of the table, but found no record types.

DECEDI__TG_NO_SIZE — No size specified for decimal number

Explanation: The Mapper compiler is evaluating the field attributes in the record layout section of the table. It did not find a size for a data type that required a size.

DECEDI__TG_NO_TYPE_FLD — No type specified for field

Explanation: The Mapper compiler is evaluating the field attributes in the record layout section of the table, but did not find a data type. It was not a structure name.

DECEDI__TG_OCC_MIN_GT_MAX — Lower limit for OCCURS greater than upper limit

Explanation: The Mapper compiler is evaluating the record sequence attributes for a record and found the MIN specification was larger than the MAX specification.

DECEDI__TG_REC_MISMATCH — Specified name has no corresponding record or segment

Explanation: The Mapper compiler is evaluating the record sequence or the document sequence section and found a record or sequence that had no layout specification.

DECEDI__TG_REP_BKON — UNTIL appears twice in MAP

Explanation: The Mapper compiler is evaluating the repeat pattern specification of a map and found the UNTIL keyword twice.

DECEDI__TG_REP_FOREACH — FOR EACH appears twice in MAP

Explanation: The Mapper compiler is evaluating the repeat pattern specification of a map and found the FOR EACH keyword twice.

DECEDI__TG_REP_NEXTLIST — NEXT LIST appears twice in MAP

Explanation: The Mapper compiler is evaluating the repeat pattern specification of a map and found the NEXT LIST keyword twice.

DECEDI__TG_REP_NOCH — NO CHANGE appears twice in MAP

Explanation: The Mapper compiler is evaluating the repeat pattern specification of a map and found the NO CHANGE keyword twice.

DECEDI__TG_REP_TIMES — TIMES appears twice in MAP

Explanation: The Mapper compiler is evaluating the repeat pattern specification of a map and found the REPEAT n TIMES keywords twice.

DECEDI__TG_SRC_VAR_NAME — Cannot use VARIANT name in Context specification

Explanation: The Mapper compiler is evaluating the SET CONTEXT clause of a map and found the name given to the VARIANT. Specify one of the records within the variant.

DECEDI__TG_SUBSCR_MISMATCH — Number of subscripts disagrees with number declared

Explanation: The Mapper compiler is evaluating an expression and found a field reference containing the wrong number of subscripts. The number of subscripts must match the declaration.

DECEDI__TG_SUBSCR_ON_REC — Subscripts not allowed here (only on fields)

Explanation: The Mapper compiler is evaluating an expression and found a global variable or record name qualified with subscripts. Only fields on records and data labels in segments can be arrays.

DECEDI__TG_TOO_MANY_POINT_AT — Too many Context items. Max. is 25

Explanation: The Mapper compiler is evaluating the SET CONTEXT clause of a map and found more than 25 items listed.

DECEDI__TG_TRM_DOC_SUBORD — TERMINATES DOCUMENT cannot have subordinates

Explanation: The Mapper compiler is evaluating the record sequence attributes and found that a record marked with TERMINATES DOCUMENT has subordinate records. The keyword TERMINATES DOCUMENT means that this is the last record of a document and cannot have a subordinate.

DECEDI__TG_UNK_FUNC — Unknown function called

Explanation: The Mapper compiler is evaluating an expression and has found a function, but the function name is not a built-in function nor is it one of the user-defined customization routines.

DECEDI__TG_VACANT_FLD — No data or space specified for field

Explanation: The Mapper compiler is evaluating the record layout and has found a field that does not have a size that is explicitly specified by the SIZE attribute or implied by the data type.

DECEDI__TG_VACANT_STRUCTURE — No data or space specified for structure

Explanation: The Mapper compiler is evaluating the record layout and has found a field defined as a structure (or implied as being one because it has no attributes). However, no fields are subordinate to the structure.

DECEDI__TG_VACANT_VARIANT — No data or space specified for any field under VARIANT

Explanation: The Mapper compiler is evaluating the record layout and has found VARIANT and END VARIANT keywords. It contains fields, but none of the fields have a size. Most likely, something is wrong with the field layout specification.

DECEDI__TI_BAD_VALUE — TEST_INDICATOR value, as specified in the Mapping Table, illegal

Explanation: This error message usually indicates an error in the command line. The TEST INDICATOR value submitted is not allowed.

DECEDI__TI_MISMATCH — TEST_INDICATOR value incompatible with the Mapping Table

Explanation: The Mapper runtime is starting up and found that the test indicator specified using the Mapping Table Editor (or its default in the table) is not compatible with the test mode (TEST or LIVE) set in the application ID of the table attributes section.

DECEDI__TI_MISMATCH_DOC — Document’s TEST_INDICATOR incompatible with Mapping Table

Explanation: The Mapper runtime has just received a document from the Digital DEC/EDI Translation Service and has found that the test

indicator value on the document does not match the specified value of the test indicator as defined in the Mapping Table Editor (or the default in the table).

DECEDI__TRUNCATED — Record Returned was Truncated

Explanation: Incoming data label is too big for space provided. User should not get this error, submit an SPR.

DECEDI__UI_NO_DIRECTION — The Mapping table does not have the direction set

Explanation: The Mapper compiler is evaluating the table attributes section of the Mapping Table, and found that the direction was not set to INCOMING or OUTGOING.

DECEDI__UNKNOWN_ATTRIBUTE — Unknown Attribute

Explanation: The Mapper compiler is parsing the field attributes in the record layout section of the table, and found an unknown keyword. Check for a missing or misplaced semicolon (;).

DECEDI__USER_PGM_ERROR — User-specified error ($ERROR)

Explanation: The Mapper runtime is processing a mapping assignment and has found the value $ERROR. This means that the user is declaring a Soft Error.

DECEDI__WRONG_NO_ARGS — Incorrect number of function arguments

Explanation: The Mapper runtime is evaluating a mapping expression and has found a function call. The number of arguments being passed do not match with those in the declaration.

DECEDI__NO_OUTPUT_RECEIVE — Normal completion — no documents generated — RECEIVE

Explanation: Informational message stating that the FETCH operation completed successfully and did not generate any documents.

DECEDI__NO_OUTPUT_SEND — Normal completion — no documents generated — SEND

Explanation: Informational message stating that the POST operation completed successfully and did not generate any documents.

DECEDI__NO_OUTPUT_TM_RECEIVE — Normal completion — timed out, no documents generated — RECEIVE

Explanation: Informational message stating that FETCH operation timed out because it could not generate any documents from Digital DEC/EDI. This error message may occur because Digital DEC/EDI can not find any of the data, or if Digital DEC/EDI is not running.

%NONAME_I_MSG %X00DA2C3 followed by a Bugcheck

Explanation: This is an Rdb message. It may mean that your account does not have a high enough Enqueue quota. Ask your system manager to check that you have enough quotas on your account to run Rdb.

DECEDI__SUCCESS_RECEIVE — Normal completion — RECEIVE

Explanation: Mapper FETCH operation completed successfully.

DECEDI__SUCCESS_SEND — Normal completion — SEND

Explanation: Mapper POST operation completed successfully.

DECEDI__ONESOFT — SOFT ERROR: Document Aborted

Explanation: This is the same as %DECEDI__SOFTERROR except there are no more documents that follow. This would be used in a single document run or on the last document of a run if it should have an error.

DECEDI__PART_RECEIVE — One or more documents generated — one or more aborted — RECEIVE

Explanation: The Mapper FETCH operation completed successfully with one or more documents generated and one or more documents aborted.

DECEDI__PART_SEND — One or more documents generated — one or more aborted — SEND

Explanation: The Mapper POST operation completed successfully with one or more documents generated and one or more documents aborted.

DECEDI__SOFTERROR — SOFT ERROR: Document Aborted, Continuing with the next document

Explanation: Indicates that the preceding error messages have been overridden and the Mapper has aborted the current document. It will attempt to continue processing with the next document. On incoming documents, this means that the current document has been aborted and will require manual intervention in Digital DEC/EDI to reprocess the document. For an outgoing document, the document in the application file has been skipped. Use the RESTART and MATCH=FIRST options in the Mapping Table Editor.

DECEDI__ZERO_RECEIVE — No documents generated — one or more documents aborted — RECEIVE

Explanation: The Mapper FETCH operation completed successfully with no documents generated and one or more documents aborted.

DECEDI__ZERO_SEND — No documents generated — one or more documents aborted — SEND

Explanation: The Mapper POST operation completed successfully with no documents generated and one or more documents aborted.

COSI__DECOVF

Explanation: Packed decimal overflow error. Severe error. Number to big too fit.

COSI__FLTOVF

Explanation: Floating overflow error. Severe error. Number too big to fit.

COSI__FLTUND

Explanation: Floating underflow error. Severe error.

COSI__INTOVF

Explanation: Integer overflow error. Severe error. Number too big to fit.

COSI__INVCLADSC

Explanation: Invalid class in descriptor. This class of descriptor is not supported. Severe error. Users should never see this error. Please submit an Software Problem Report (SPR) to Digital.

COSI__INVCLADTY

Explanation: Invalid class and data type in descriptor. This class and data type combination is not supported. Severe error. Users should never see this error. Please submit an Software Problem Report (SPR) to Digital.

COSI__INVCVT

Explanation: If the source value is negative and the destination data type is unsigned, this error is returned.

COSI__INVDTYDSC

Explanation: Invalid data type in descriptor. This data type is not supported. Severe error. Users should never see this error. Please submit an Software Problem Report (SPR) to Digital.

COSI__INVNBDS

Explanation: Invalid number. There is an invalid character in the input, or the value is outside the range that can be represented, or there is a programming error.

COSI__OUTSTRTRU

Explanation: Output string truncated. This is returned only when NBDS is both source and destination and no scaling is requested. The result is truncated.

COSI__ROPRAND

Explanation: Reserved operand error. Severe error. Users should never see this error. Please submit an Software Problem Report (SPR) to Digital.

Part III Digital DEC/EDI Mapping Topics

This part of the book contains detailed information on the following mapping topics:

- Attributes
- Files
- Importing Digital DEC/EDI Data
- Detail
- Expressions
- Lookups
- Constructs
- Hooks

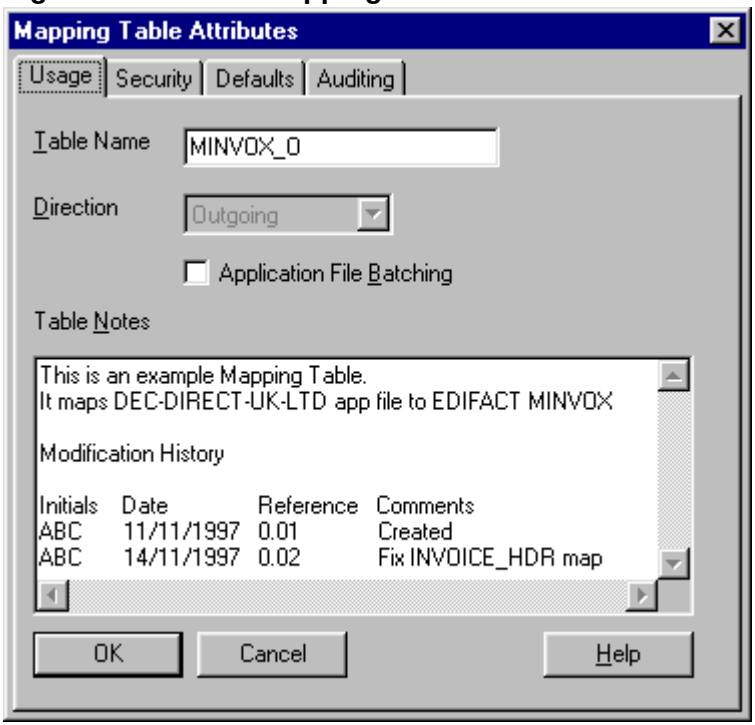
The final chapter is a glossary of general Digital DEC/EDI terms.

Chapter 10 **Mapping Table Attributes**

This chapter describes how to specify table attributes when defining a Mapping Table. You can do this either when you first create a Mapping Table, or (for most of the possible attributes) at a later date.

The following illustration shows the Mapping Table Attributes screen.

Figure 10-1 The Mapping Table Attributes Screen



When you define a new Mapping Table, the screen is displayed automatically. In editing the attributes of an existing table, you can display the screen by double-clicking on the Mapping Table header.

The screen has the following four tabbed parts:

- Usage

  Use this to supply mandatory details such as, the name of the table and the transmission direction to which it applies. You may also include a brief description of the table, purely for personal notes.
- Security

  Use this to specify which applications can use the Mapping Table.
- Defaults

  Use this to define the default values for the Mapping Table.
- Auditing

  Use this only if you want to select points on the transmission route where you may collect auditing and debugging information.

The following sections discuss each part in more detail. Note that the online help contains information about entries for all the fields in each part.

Usage

In Usage, you may specify the following:

Table Name	Give the table a name of the form, `NAME.FBI`. You can not change the name of an existing Mapping Table.
Direction	This indicator determines whether the Mapping Table is defined for **Incoming** or **Outgoing** files. You can not change the direction of an existing Mapping Table.
Table Notes	Enter any notes you wish to make about the table. These are for documentation purposes only.

Security

You use the Security tab dialog to define the applications that may use a particular mapping table.

The following rules apply:

- The screen allows you to assign a maximum of twenty application IDs to a table. You must assign at least one application ID before you compile the Mapping Table.
- The application IDs you list here must correspond exactly with application IDs defined in the Digital DEC/EDI trading partner profile.
- At runtime, when the application sends or fetches a file of document data, the Application Client call specifies the required Mapping Table and the application ID; the application ID can alternatively be specified as a default runtime qualifer. The application ID specified in the call, or as a default, must match one of the list of application IDs assigned to the mapping table.
- You may specify the special value of “ANY” to allow any registered application to use this mapping table. In this case, you need to give the **named_application** option in the Application Client call.

Defaults

You assign default qualifiers to a Mapping Table by selecting the **Defaults** option. These apply at runtime, however, if required, you can override the defaults in the Application Client call.

For example, you can assign a default application, document type and trading partner to an incoming table, to specify that the table will be used for mapping documents of a particular type, received from a particular trading partner, and destined for a particular receiving application.

In general, it is recommended that you use qualifiers in the Application Client call to specify how a file of document data is to be mapped, rather than relying on runtime default qualifiers. In this way the control of document data rests at the Client end, where it properly belongs; also there is less likelihood of a file receiving unexpected treatment because of defaults that had been set up and then overlooked.

On the **Defaults** part of the screen, you enter any default values you require in the fields described below.

Application ID	Enter a default value for the Application ID to be used in the processing of the documents.
Object Name	Enter a default value for the Object Name to be processed by the Mapper.
Partner ID	Enter a default value for the name of the Trading Partner with whom you want to exchange documents mapped with this table. The Trading Partner Profile must be defined in the Digital DEC/EDI Server.
Timeout	In the incoming direction only, enter the number of seconds after which the mapper should time out when fetching documents.
Mail	Enter the E-mail address of an account to be notified of any SOFT ERROR or HARD ERROR failures.
User Reference	Enter a default override for the global variable $USERREF in the Mapping Table.
Test Indicator	Select a default value for the test indicator, or leave the field blank. You can choose from **Live**, **Partner Tes**t, **Translator Test** or **Local Test**.
Priority	Select **Normal**, **Immediate** or leave the field blank to signify **Unspecified**. This qualifier applies in the outgoing direction only.
Match	Select **First, All** or leave the field blank to signify **Unspecified**.

Auditing

You may use Audit Controls to specify the level of audit and debugging information — if any — that is to be recorded when the Mapping Table is used. On the Auditing part of Mapping Table screen, you select various points at which auditing and debugging information is to be gathered. The use of the Audit Controls is optional.

Note that as you increase the number of points where you require auditing information, the time taken to transmit a given document also increases. In order to acheive optimum transmission time, it is recommended that you refine the amount of auditing information you require in relation to the volume of traffic on the links to your various trading partners.

Some of the following controls can be applied to only one direction of transmission: the remainder can be applied to either direction.

Start of Processing

For outgoing Mapping Tables only.

When you set the **Process Start** indicator, the Mapper keeps a copy of the Outgoing Application File submitted by the Application Client at the start of processing. The Mapper does not try to interpret the data.

After Preprocessing

For outgoing Mapping Tables only.

When you set the **Preprocess** indicator, the Mapper keeps a copy of the Outgoing Application File after it has been processed by the User Defined Custom Routine, defined at the Preprocess hook point. The Mapper does not try to interpret the data.

After Record Hook

For outgoing Mapping Tables only.

When you set the **Record Hook** indicator, the Mapper keeps a copy of the Outgoing Application File records after each record has been modified by the User Defined Custom Routine, defined at the Record hook point. The Mapper does not try to interpret the data.

Digital DEC/EDI Internal Format File

For outgoing and incoming Mapping Tables.

When the direction is outgoing, the Mapper keeps a copy of the Digital DEC/EDI Internal Format File generated by the Mapper for processing by the Digital DEC/EDI Translation Service.

When the direction is incoming, the Mapper keeps a copy of the Digital DEC/EDI Internal Format File received from the Digital DEC/EDI Translation Service for processing by the Mapper. The Mapper does not try to interpret the data.

Custom Routine Arguments	For outgoing and incoming Mapping Tables. When you set the **Custom Routine Arguments** indicator, the Mapper creates a file detailing each call to a User Defined Custom Routine for which auditing is enabled. The Mapper does not try to interpret the data.
Mapping Table	For outgoing and incoming Mapping Tables. When you set the **Mapping Table** indicator, the Mapper keeps a copy of this Mapping Table in compiled form, for every mapping request it is used in. The Mapper does not try to interpret the data.
Before Postprocessing	For incoming Mapping Tables only. When you set the **Before Postprocessing** indicator, the Mapper keeps a copy of the incoming Application File before it has been processed by the user-defined custom routine defined at the Postprocess hook. The Mapper does not try to interpret the data.
End of Processing	For incoming Mapping Tables only. When you set the **End of Processing** indicator, the Mapper keeps a copy of the incoming Application File at the end of processing, as fetched by the Application Client. The Mapper does not try to interpret the data.

Chapter 11 Specifying Application Files

This chapter describes how to specify application files when defining a Mapping Table using the Mapping Table Editor.

An **application file** is the file containing document data, sent or received by the user application. For outgoing documents, the file is created by the application and passed to the Mapper. For incoming documents, the file is created by the Mapper and passed to the application.

For each type of file, in each direction, you need to specify the the sequence of records in the file and the layout of the fields in each record.

Digital DEC/EDI provides predefined templates to help you do this. You may also develop your own templates.

Getting Record Layouts

In specifying the Record Layouts in a Mapping Table, you can import a named Record Layout template file that is either predefined and supplied with Digital DEC/EDI or one that you have generated yourself. Record Layout files that you generate are based on the Record Layouts of existing Mapping Tables.

You access these options from the **Layout** menu which is included on the menu bar when the Records window is active in the Mapping Table Editor.

To get a Record Layout, select the **Add** option, the **Layout** option, and then enter the name of the layout in the dialog that is displayed.

To get a predefined Record Layout template, select the **Pilot...** option, and then select the **Template** button, displayed on the Application File Pilot dialog.

Note that you also use this dialog to generate layout.

Refer to the online help for details on your options.

Entering or Editing the Record Sequence

You define (or edit) a record sequence for the application file, and define the nested relationships between records by using the options on the **Sequence** menu. The menu is added to the menu bar when the Sequence window in the Mapping Table Editor is active. The Mapper uses the information you enter to determine the order in which it can expect to encounter records in the data.

Refer to the online help for detailed information.

The fields on the entry screen are as follows:

LEVEL This is the level number of the record. The level number indicates the record's nested position in relation to other records in the file. The first record will be a level 1 record. Those records that are siblings of the first record type will also be level 1.Records subordinate to the first will be level 2. Those records subordinate to it will be level 3 and so forth, defining the nesting of the hierarchy. Level 0 is not used.

Record Type This is the name of the record type as defined in the **Enter/Edit Record Layouts** screen. This record type name is used during the mapping process to identify a record in the source.

Although not very common, the same record layout can appear in the source as subordinate to different parent record types, the Mapper has to know which place in the source tree the record refers to. In such cases, the name has to be unique. To make it unique, the Mapper requires the suffix, hash sign and a number (#n). If the Mapper discovers an Record Type name that is duplicated, it displays an error message, which recommends adding the suffix.

You can also use the keywords VARIANT and END VARIANT in this field to indicate a variant record. The rules governing VARIANT and END VARIANT keywords are:

- The level number of the VARIANT and END VARIANT must be the same.
- The keywords VARIANT and END VARIANT bracket and apply to records with the same level number and indicate that the records can appear in any order subordinate to a parent record. The level number of the VARIANT and END VARIANT keywords must be the same as records they bracket.
- The keyword VARIANT can have a name associated with it. The name then becomes a synonym for the list of record names that the keywords VARIANT and END VARIANT bracket and apply to. The syntax is:

```
2 VARIANT name
2 record
    .
    .
    .
2 END VARIANT
```

The Mapper assumes that any given Application File might contain more than one record type. It also assumes that the data in the Application File is a

tree structured sequence and that the highest level record appears in the record stream before a record that is subordinate to it.

You must specify to the Mapper the order, or sequence, in which it can expect to find the records in an Application File. You have to specify the kinds of records that can follow in a subordinate relationship any other kind of record. You do this using level numbers to show subordination. Generally, there are three basic types of subordinate relationships between records in tree structures:

- Parent to child relationships
- Sibling relationships
- Variant relationships

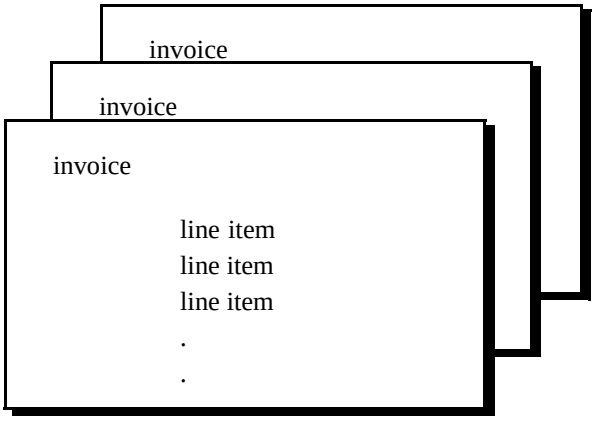
The following subsections describe how to specify these types of relationship.

Parent to Child Relationships

A parent is a record type that has zero or more subordinate (child) record types. Each child record type in turn is a parent to zero or more child record types.

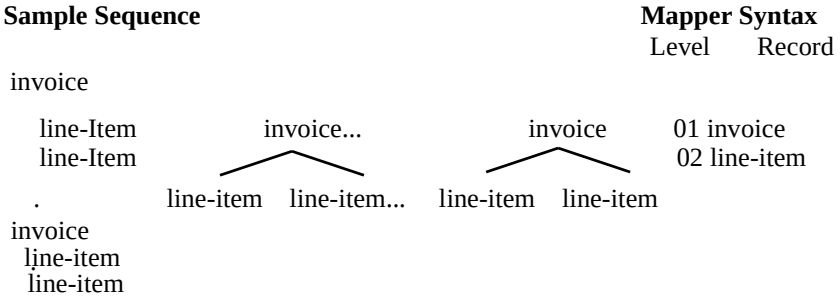
Suppose, for example, an application file contains data for invoices from an accounts receivable program. For each invoice record type there is at least one and possibly many line-item record types. Figure 11-1 *Invoices* illustrates such an application.

Figure 11-1 Invoices



This invoice application would be expressed in the tree structure shown in Figure 11-2 *Invoice Tree Structure*. Below is a description on how Figure 11-2 *Invoice Tree Structure* is set up.

Figure 11-2 Invoice Tree Structure



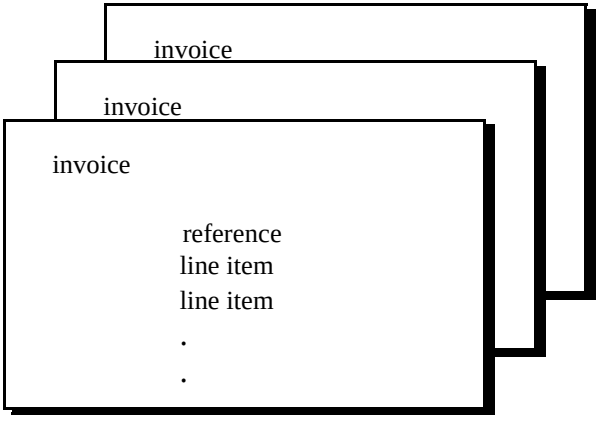
Sibling Relationships

In a sibling relationship, a record can have more than one type of record subordinate to it. In such a relationship, child records have the same parent, but are of different record types.

Suppose an Application File contains data for invoices from an accounts receivable program, and that each invoice has, in addition to line-item records, a reference information record that provides data relevant to the

invoice as a whole. Figure 11-3 *Invoices With Sibling Records* illustrates such an application.

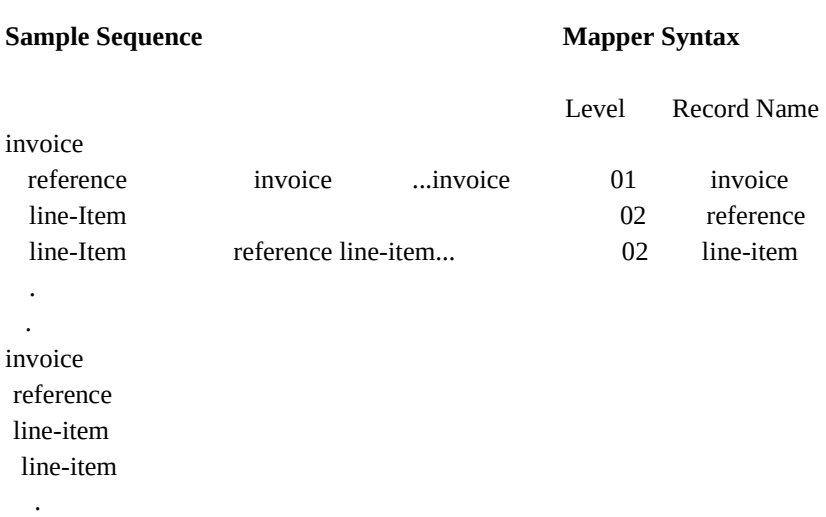
Figure 11-3 Invoices With Sibling Records



As you can see, records of type **reference** and records of type **line-item** are both subordinate to a record of type **invoice**.

Figure 11-4 *Sibling Construct* shows how a tree for a set of sibling records is expressed. The B's and C's are subordinate to A, but the C's are not subordinate to the B's as in the parent to child relationship described earlier. In the record stream for records in a sibling relationship, all of the B records subordinate to an A record come before all of the C records subordinate to that same A record.

Figure 11-4 Sibling Construct



Below is a description of how Figure 11-4 *Sibling Construct* is set up.

Sample Sequence Shows the sequence of records as it would occur in the application file.

Tree Shows how the Mapper would organize the sample sequence into a tree.

Mapper Syntax Shows the Mapper notation as you would enter it.

The 2 level number indicates there can be reference record types and line-item record types subordinate to each 1 invoice record type.

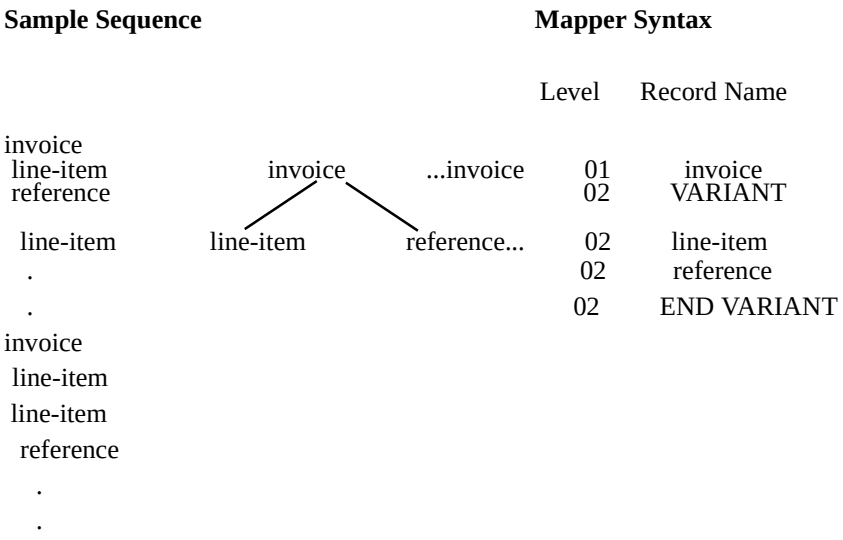
Notice that the reference record types always appear in the file before the line-item record types for each invoice record. Siblings are ordered so that all records of one type appear in the record stream before all records of the next type for the same parent record. This is demonstrated in the example. The 2 records are ordered so that the reference record type comes before the line-item record types for each invoice record.

Variant Relationships

Variant structures are similar to Sibling structures, except that their records are mingled together. Records from a set of variant record types subordinate

to the same parent can appear in the file in any order. This is demonstrated in Figure 11-5 *Variant Construct*.

Figure 11-5 Variant Construct



The invoice record is parent to both reference and line-item records. These reference and line-item records can appear in any order subordinate to the invoice record. It is still true that the reference record contains information that applies to the invoice record as a whole and not to a specific line-item or group of line-item records. The difference between the Variant structure and the Sibling structure is that you cannot be sure where in the record stream the reference record will appear. All you know for sure is that it will be subordinate to the invoice record.

Below is a description on how Figure 11-5 *Variant Construct* is set up.

Mapper Syntax Shows how you would describe this Variant relationship to the Mapper. The keywords VARIANT and END VARIANT tell the Mapper to expect the reference and line-item records in any order and to deal with them accordingly.

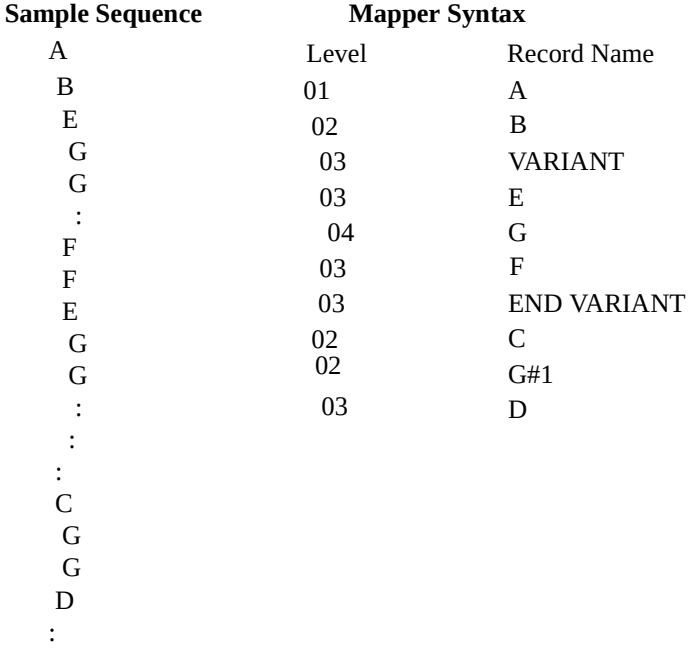
This type of structure is not very common, but it could occur in both the Application Data File and in the Digital DEC/EDI document type structure.

Use Variant structures when you cannot rely on having an ordered Sibling relationship.

Combined Relationships

By combining these constructs, you can describe any tree structure. Figure 11-6 *Combinations of Constructs* shows how parent to child, sibling and variant relationships can be combined in one description.

Figure 11-6 Combinations of Constructs



In Figure 11-6 *Combinations of Constructs* you can see a combination of the parent to child, sibling, and variant constructs. The A record has a parent to child relationship with 2 level B, C and D records. The B, C and D records have a sibling relationship. The 3 level E and F records have a variant relationship, and E is the parent record to the 4 level record G.

Notice that there can also be child G records to parent record C as well as to parent record E. The syntax, G#1, refers to the G records that are subordinate to C records and points out that they are the same record types but in a different position in the tree.

Entering Record Attributes — Outgoing

The **Record Attributes — Outgoing** screen screen allows you to enter attribute information about each record in the application file.

The fields for the **Enter Record Attributes — Outgoing** screen are:

MIN This is the minimum number of instances of this type that can appear at this point under a parent instance in the record sequence. Enter 0 if the instance is optional. If you enter 1 or more, and the Mapper finds less than that number of child instances under this parent, it generates a Mapper error at runtime. This is how you can specify mandatory records.

MAX This is the maximum number of instances of this type that can appear at this point under a parent instance in the record sequence.

If the Mapper finds more than the specified maximum number, it generates an error at runtime. If you want to allow some unknown large number of instances, enter the keyword MANY. This will prevent validation on occurs limits at runtime.

BATCH HEADER Mark this box if this record applies to more than one document. All records that are not at or subordinate to a level marked with BEGINS DOC should be marked as a batch header.

RECOGNITION EXPRESSION

This field is for a conditional expression that the Mapper uses to recognize a record type in the record stream. This field is necessary only when the Mapper is reading records from your Application Data File that contains data for an outgoing EDI document.

It is not needed to describe records in a Mapping Table for an Incoming document direction. Normally, the recognition expression is the record code, which is one of the fields on the record, but it can be any expression that evaluates to TRUE.

BEGINS DOC

This field specifies that each instance of this level is the start of a set of data for a new document. This information is used by the Mapper to identify the batch header records and the beginning of a new document when processing a file containing data for an outgoing document.

When the Mapper encounters a record that matches the type of the record type on a level marked as the beginning of a document, it knows that this record and all records subordinate to this level in the tree (or terminated with an end-of-file record) are part of one document. The Mapper stops reading the record stream and processes the data for the document before continuing to read records.

Several different levels can be marked as the beginning of a document if different types of records indicate the beginning of different types of documents. The rule is that a record or segment cannot be marked as the beginning of a document if it is subordinate to any other record or segment that is marked as the beginning of a document.

ENDS DOC

This field specifies a level with a record type that will always be the last record in a document. This field is optional. It is only needed when the input is not an actual application file but rather a record stream from a mailbox where there might be delays between documents.

In such cases, it would be useful to insert a record in the record stream to indicate that the previous document is complete so that the Mapper can proceed with processing the document without having to wait until it encounters the beginning of the next document.

Always make certain that the record identified with the ENDS DOC specifier is subordinate to the record that has the BEGINS DOC specification.

For example, if the BEGINS DOC was at the 2 level, the ENDS DOC record must be at level 3.

$DOCTYPE

This is a value the Mapper can use to obtain the value for the global variable, $DOCTYPE, for the document being processed. The Mapper uses this value, along with the value of the global variable, $PARTNER, to identify the mapping set to be used. The Mapper evaluates this expression each time a record is placed at this level in the source. The expression can contain constants, global variables, and fields from the record that was just placed at this level, or from the record's parent.

$PARTNER

This is a value the Mapper can use to obtain the value for the global variable, $PARTNER.

The Mapper uses this value, along with the value of the global variable, $DOCTYPE, to identify the set of mappings to be used. As the Mapper is filling the source tree using the data in the application file, it evaluates this expression each time a record is placed at this level. The expression can contain constants, global variables, and fields from the record that was just placed at this level, or the record's parent.

Enter Record Attributes — Incoming

The **Enter Record Attributes — Incoming** screen allows you to enter attribute information about each record in the application file.

The fields for the **Enter Record Attributes — Incoming** screen are:

MIN This is the minimum number of instances of this type that can appear under a parent instance in the record sequence.

Enter 0 if the instance is optional. If you enter 1 or more, and the Mapper finds less than that number of child records under this parent, it generates a Mapper error at runtime. This is how you can specify mandatory records.

MAX This is the maximum number of instances of this type that can appear under a parent instance in the record sequence. If the Mapper finds more than the specified maximum number, it generates a Mapper error at runtime.

If you want to allow some unknown large number of instances, enter the keyword MANY. This will prevent validation on occurs limits at runtime.

Entering or Editing Record Layouts

You use the Layout menu to insert, modify or delete Record Layouts. The Layout menu is included on the menu bar when the Layout screen is active.

Refer to the online help for detailed information. The fields on the entry screen are as follows:

When you reference indexes, note that the first element is numbered 1, not 0.

A record type is a specific layout of fields on a record. Record type layouts describe the order in which the fields appear for each record type that might be included in an application file. If you were writing a COBOL program, for example, these record type definitions would be in the FD section.

The fields must be described on this screen in the order that they appear on the record. The sequence of fields is notated in the same manner as COBOL records — with level numbers — except that the numbers must be

consecutive (levels cannot be skipped). Level numbers specify the nesting of fields on the record. The format of the level numbers is 1 for the highest, 2 for the next and so forth.

You can define fields that overlay or redefine the same space within a record. Similar flexibility is found in variant structures unions (in C and Pascal), and the REDEFINES clause (in COBOL). Such fields are implemented in the Mapper using the special keywords VARIANT and END VARIANT for the field name within the record layout. Those fields between the VARIANT and the END VARIANT keywords overlay or redefine the same space within the record.

Field names must be unique within the same structure (at the same level number) on the record. This is true in most programming languages. The special keyword FILLER can be used as a field name without regard to uniqueness.

Many languages insert default alignments for these fields. The Mapper assumes no implied alignment. Therefore, if a record has fields that have been aligned, it may be necessary to explicitly insert filler fields.

These data types are in two categories, those that are OpenVMS primitive data types and those that are not. In addition, the Mapper supports several of the more common IBM data types.

Tru64 UNIX

Note: *If a binary field must be defined as the first field of a record then the application file must have at least a carriage return character at the end of each record.*

Application File Batching

An Application File may contain data for several EDI documents, to be sent to the same Trading Partner. In some cases, documents must be batched together in the same EDI Transmission File, as the documents contain related information.

For example, in the UK, the EDIFACT TAXCON message must be sent in the same transmission file as all the INVOIC messages which it summarizes.

It is not always convenient to rely on the Transmission File Builder’s build schedule to place these in the same Transmission File.

To guarantee that documents are batched in the same transmission file, Application File Batching must be enabled by setting this field to “X” by selecting the Application File Batching box in the Usage tab of the Mapping Table Details screen.

At Runtime, if Application File Batching is enabled, then for each Application File submitted, the Mapper begins a new batch.

While processing the Application File, for each document in the Application file, the Mapper determines the Application ID, Trading Partner and Test Indicator. Each time one of these changes, it marks the the current batch as complete, and starts a new batch.

If these parameters remain unchanged for a given Application File, the Mapper marks all the documents from that Application File as belonging to the same batch.

If the parameters change, for example by having a Batch Header record that specifies a new Trading Partner, you may have an Application File contains batches of documents for several Trading Partners.

Chapter 12 Importing Digital DEC/EDI Document Data

This chapter describes how to import document definitions from the Translation Service into the Mapping Service.

A document definition consists of a document structure and a set of data labels representing an EDI document format. The Mapping Service needs this information, so that it can convert document data between the application file format passed between the user application and the Mapping Service, and the internal file format passed between the Mapping Service and the Translation Service.

Before you can import the data labels into the Mapping Service, they must already have been set up on the Translation Service. You need to understand how the data labels represent the EDI document format. These topics are covered in *Document Definitions and Data Labels* on page 12-1.

The subsequent sections of this chapter then describe how to import the definitions into the Mapping Table Editor. You need to do this for each document that is going to be handled by the Mapping Table Editor, in either direction.

It is important to make sure that Mapping Tables stay synchronized with the definitions of internal documents. Should any document definition or data labels change on the Translation Service, you must re-import the document definition into the Mapping Table, edit the mapping itself to take account of the new format or labels, and recompile the table.

Document Definitions and Data Labels

A **document definition** consists of a set of **data labels** representing an EDI document format. The Mapping Service needs this information, so that it

can convert document data between the application file format passed between the user application and the Mapping Service, and the internal file format passed between the Mapping Service and the Translation Service.

Each data label is a symbol representing a document (sub)element: that is, a subelement in the case of an EDIFACT/ODETTE document.

The internal file format consists essentially of a sequence of data labels and assigned data values. It also contains special data labels with no assigned value, to represent structures such as hierarchic groupings and repetitions.

Before you import a document definition into the Mapping Service, the required data labels must already be set up on the Translation Service. This can be done automatically, by using the **Data Label Generator**. Examples of the set-up procedure are described in the *Digital DEC/EDI: User's Guide*, and the Digital DEC/EDI commands are fully defined in this book.

Setting up data label tables on the Translation Service may or may not be done by the same person who sets up Mapping Tables on the Mapping Service. In either case, when you come to import a document definition from the Translation Service into the Mapping Service, it is helpful to understand the following points:

1. Each data label represents a (sub)element **path** within a segment. Taking each segment in turn, you need to devise a data label for every (sub)element.

   The purpose of a data label is to tell the Translation Service whereabouts *within* the segment a particular data value belongs.
2. Every label must represent a unique (sub)element portion.
3. If an entire segment appears more than once in a document type (and in other document types) then the same data labels can and must be used in each case, since they represent the same paths within the segment.

   For example, a purchase-order might have a header segment specifying the default delivery address, and then a number of detail segments specifying different delivery addresses for particular items on the order. In this structure you would use the same data labels for each occurrence of the segment.

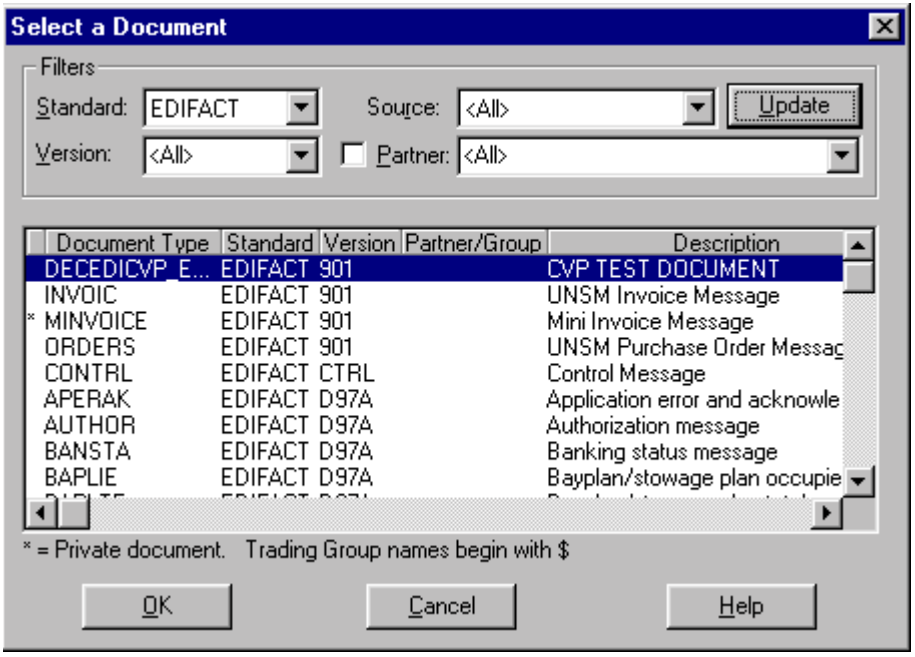
When you import a document definition from the Digital DEC/EDI Server into the Mapping Table Editor, you provide the Mapping Table with all the information it needs about the document format:

- The sequence of segments in the document. This includes repeating segments and repeating groups of segments (loops); loops can be nested.
- The data labels that represent each (sub)element path in a segment, for each segment in the document. These also represent nested groups, qualified segments, qualified (sub)elements, and repeating (sub)elements.

Import Document Definitions from Digital DEC/EDI

The **Select Document** screen enables you to extract a document definition directly from the Digital DEC/EDI Server. The **EDI Document Extract** screen is shown in Figure 12-1 *EDI Document Extract*.

Figure 12-1 EDI Document Extract



First, select the **EDI Documents** option from either the **Map Navigator** or the **Mapping** menu. Then, select the **Extract Document...** option from the **Server** menu to display the screen. Use the filters to narrow the scope of your search.

If you need any further information about the options on the screen, refer to the online help.

Chapter 13 **Mapping in More Detail**

This chapter describes mapping table parts and concepts with regard to the mapping between an application file format and an internal document format.

You need to specify a mapping for every combination of application file and internal document that the Mapper is going to process. Chapter 16 *Supported Mapping Constructs* gives further information about the mapping structures you can use.

Mapping is the process of rearranging and manipulating data as it is moved from a source tree to a destination tree into the correct format and order. In other words, mapping is the process which the Mapper goes through to move data from the document side to the file side, or the reverse.

During the mapping process, the Mapper uses a *pull* strategy to pull data elements from the source tree to data elements in the destination. The movement of data elements is done by assignment statements.

To keep track of all these assignments, you group them for the Mapper into those that are necessary to build one destination record or segment.

One **map** is a group of assignment statements that together build one record or segment. Whether it is a record or segment you generate depends upon the direction. A group of maps that together generate one document is a **mapping set**.

Many different kinds of documents can be generated from one Application File, each of which uses a different mapping set. If, for example, the same file contained data for invoice documents and data for payment advice, each document would be processed with a different mapping set.

You can also have variations on the same Digital DEC/EDI document definition type. For example, you could use the same document type for two different trading partners, even if each partner wanted to see the same data organized differently. To do this you would have a different mapping set for each trading partner.

The Index of Mapping Sets

The Mapping Table screen shows the hierarchy of all mapping sets, their component parts, and assignemnts. This is known as the **Index of Mapping Sets**.

The following pages provide more detail on general mapping terminology and concepts.

Partner

This is the name of a trading partner as defined in Digital DEC/EDI. It allows you to create trading-partner specific mapping sets using generic Digital DEC/EDI document definitions so that you can format documents according to trading partner specifications.

If the document definition for a mapping set was trading-partner specific, the PARTNER field is the partner for the document definition. If the document definition was not trading-partner specific, the GENERIC field will be automatically marked with an X and you can specify a trading partner or you can leave it blank to match all trading partners for this Object Name.

If you specify a trading partner, the mapping set applies only to that trading partner.

Generic

Mark this field with an X if the document definition from Digital DEC/EDI is not trading-partner specific. In other words, when the document definition was extracted from Digital DEC/EDI, the partner field was left blank.

Standard

This is the EDI standard that will be used to format the document.

Version

This is the version of the EDI standard that will be used to format the document.

Internal Doctype

This is the Digital DEC/EDI internal document type that will be used to format the document.

The following table describes the meaning of the possible combinations of partner and generic field.

Table 13-1 Combinations

Partner	Generic	Meaning
	[x]	Generic Digital DEC/EDI document definition used with all partners unless a partner later on list has the same name.
partner	[x]	Partner-specific map set with a generic Digital DEC/EDI document definition.
partner	[ ]	Partner-specific map set with a trading-partner specific Digital DEC/EDI document definition.
	[ ]	Not allowed. Partner and generic cannot both blank.

Each line on the **Index of Mapping Sets** represents a mapping set. Only one mapping set can be used on any one document that is processed by the Mapper.

At runtime, in the Outgoing direction the Mapper chooses which mapping set to use by looking up the $DOCTYPE and $PARTNER combination.

In the Incoming direction, the Mapper does two look-ups on this table. One to find the criteria to use in polling Digital DEC/EDI, and one to find the mapping set to use when it processes each document it returns.

The fields that specify a document definition (internal doctype, standard, version, generic, and partner (if not generic)) must match one of the valid

document definitions. It is best to use the **Find** option on the **Search** menu to be sure of a match.

You can change the document definition that will be used with a mapping set any time.

In the Outgoing direction, the Mapper attempts to match the segments in the new definition with those that have maps withthe same segment name from the previous definition. Any segments that are left over and have maps go to the bottom of the index of maps and are marked with a hash sign (#).

In the Incoming direction, any record sequence changes will cause the The Mapper to attempt to match up the record sequences in all the maps with the new record sequence. Left overs go to the bottom of the index and are marked with a hash sign (#).

If you change a document definition or your segment sequence, make sure that the mapping assignments are still viable.

Level

This is the level number of the record sequence or the segment sequence, depending on whether it is Outgoing or Incoming.

Segment or Rectype

This is the name for each level in the sequence.

For the Incoming direction, the column's title is **Rectype**. When this screen is first displayed, the Mapper fills in all the record sequence information that was entered in the **Record Sequence Definition**.

For the Outgoing direction, the title is Segment. When first displayed, the screen will be filled in from the segment sequence information extracted from the Digital DEC/EDI document definition. This is the same document definition that you associated with the Object Name in the **Index of Mapping Sets**.

Segment or Record

The minimum and maximum number records or segments that can be created by this map.

Map ID

This is the name you give to a map. The default is “Unspecified”. Its purpose is to identify the map on the **Index of Maps** screen and to indicate that a map has been created.

Error On

On any of the maps that specify a repeat, there is the possibility that the map will run out of data in the source or will exceed the maximum number of records or segments (depending on direction) for the destination.

In the simple case, this is the normal situation and you can leave the **MAX Exceeded** box blank. For more advanced usage, refer to the Description Section.

Navigation

Navigation is the instructions to the Mapper specifying how to find a starting place in the source tree for data mapping, and how to move that starting place as the map is processed. These instructions are found in the **Set Context**, **Repeat Pattern** and **Condition** entries.

Set Context

The information in this field tells the Mapper where to start in the source tree. The entry in the Set Context field specifies the record or segment in the source tree that is to be the source for the assignment statements on this map.

For most situations, you simply specify the Rectype (Outgoing) or Segment (Incoming) name as defined in the **Record Sequence Definition** or the Segment Sequence section of the document definition.

Not all of the information in the mapping needs to come from the place to which you set context. In most simple cases, it should be the Rectype or Segment that contains the most information for the record or segment you are generating.

For example, in the Outgoing direction if you are building an IT1 segment of an 810 invoice document from a line-item record (assuming the Rectype is LINE_ITEM) the context you would set on the IT1 map is LINE_ITEM).

If this specification does not cause the result you expect, that is, the data seems to be coming from the wrong places or you are not getting all of the records or segments you should be, then this simple specification might not be sufficient. For more advanced usage, refer to the Description Section.

If you do not specify a context, the context from the last map executed is still in force. As a general rule, always specify a context unless you are getting data only from global variables.

Repeat Pattern

This field specifies whether a map will be repeated or not, how many times, and how the repeat loop will be terminated. If you leave this field blank, the Mapper uses the auto-repeat mechanism. This will be sufficient for most simple cases. For example, given a map for the IT1 segment, it will be automatically repeated once for each line-item record for an invoice.

If this specification does not cause the result you expect, that is, the data seems to be coming from the wrong places or you are not getting all of the records or segments you should be, then this simple specification might not be sufficient. For more advanced usage, refer to the Description Section.

For the FOR EACH entry, you can enter a parent record or segment name.

Condition

This field is used to determine whether or not to execute the assignment statements in this part of the map. This does not effect other parts of the same map. The Condition field is for an expression.

After the context has been set and a map is being executed, this condition expression is evaluated. If the condition is true, non-zero, or not specified, all of the map assignments for this part are executed. If the condition is false or zero, none of the mapping assignments are carried out.

If no assignments are made to fields in the destination instance (record or segment being generated) then the instance is not generated. If nothing gets generated, none of the subordinate maps will be executed, but the Mapper will continue with the next iteration of the source according to the Repeat Pattern specified for this map. This pattern is useful where the map applies only to a subset of the source records being iterated.

Condition specifications can also be useful if there are mapping assignments that you want to conditionally execute. Using additional New Context Parts, one map can have several blocks of conditionally executed mapping expressions.

Mapping Assignments

This is where you describe to the Mapper how data is to be moved from the source to the destination. This field is for assignment statements. The syntax is:

```
destination = source-expression;
```

The left side of the statement is the destination (field or data label). The right side is source (data label or field) but can also be an expression. The equal sign (=) is required for each assignment statement. An assignment can extend over several lines, if necessary.

When the screen is first displayed, the left half of the assignment statement (field or data label depending upon direction) is filled in for you for all of the fields or data labels being generated by this map. Do not assign to this list values to fields or data labels not on the record or segment being generated by this map.

You can delete those that you do not plan to use. You can insert new assignment statements, but only to assign values to global variables. You can rearrange the list, if necessary.

This section presents more advanced topics about the **Map** screen. It includes a discussion of:

- Record Instance Numbering.
- Structure Mapping (Navigation).
- Data Mapping (Mapping Assignments).

If the automatic repeat pattern does not seem to be working for your application, you might need to use other repeat patterns. To do so, you will need to know about some advanced concepts.

Record Instance Numbering

An instance of a record is an individual record (or segment) in a file. For the Mapper to be able to set context to any record or segment, navigate through

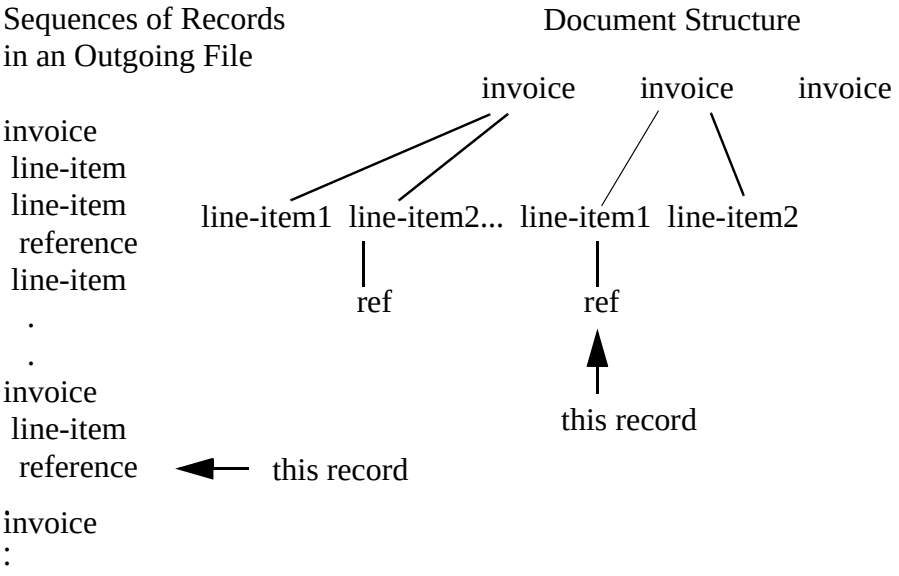
a tree of records or segments, and assign data to fields or data labels, it must be able to pick up any record or segment in a tree of records or segments.

Every record or segment must have a “handle”; a way for the Mapper to identify that record or segment. The handle is the instance number.

All records or segments in the tree derive their position from the records or segments above them. You describe a record’s or segment’s position to the Mapper by identifying where it is in the tree. In a set of records or segments that are children of one parent, an instance of a record or segment is the ordinal number associated with that child record or segment.

The way you identify the record’s or segment’s location is by a path, which includes the instance number of each of its parents down to the record or segment from the top of the tree. The path uniquely identifies the record or segment in the tree.jb

Figure 13-1 Current Instance Path



Consider Figure 13-1 *Current Instance Path*. The path to the record *reference*, pointed out in both the sequence and the conceptual tree, is:

1. Second invoice record.
2. First line-item record under the second invoice record.
3. First reference record under the first line-item record under the second invoice record.

Using the instance numbering to assign numbers to each of these records, the path is:

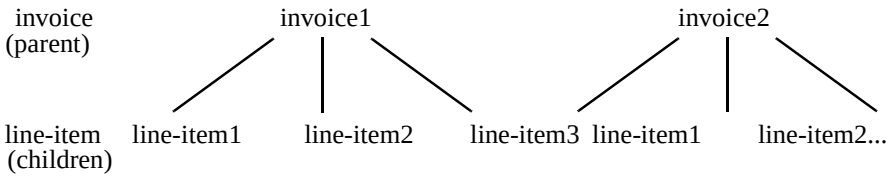
1. Invoice instance 2.
2. Line-item instance 1.
3. Reference instance 1.

This path to the record uniquely identifies the record. There is no other record in the tree of records with the same path.

Notating an Instance.When the Mapper loads the source data into the source tree, it links each of the records in order at each level of the tree and it links each record to the record above it (its parent record). Note that an instance cannot cross document boundaries.

Figure 13-2 *Linked Records in a Tree* shows the horizontal and vertical linking of records.

Figure 13-2 Linked Records in a Tree



As an aid in referencing specific records, the Mapper numbers them. All child records for a single parent record are part of a separate numbering sequence as shown in Figure 13-2 *Linked Records in a Tree*. The left-most child of any given parent is always number 1. These numbers are used to navigate to a record relative to a previous record.

When you identify a particular record in the tree of records in the mapping table, you use a special syntax shown in the following example.

```
record{number}
```

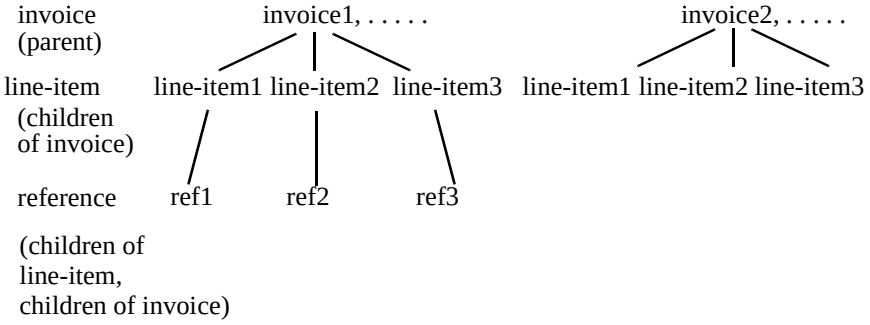
The **record** is the Rectype (or Segment for incoming) name. In the previous example, it would be **invoice**, **line-item**, or **reference**. The **number** is the record's or segment's position relative to other records under its parent. The braces (*{}*) are required to enclose the number. For example, a record with the following instance identifier is the first invoice record:

```
invoice{1}
```

Instance identifiers for parent records can be put together to define a path within the tree to a specific record. For example, in the tree shown in Figure 13-3 *Notating Linked Records*, the following instance notation defines the first reference record, which is under to the first line-item record, which is under the first invoice record.

```
invoice{1}.line-item{1}.reference{1}
```

Figure 13-3 Notating Linked Records



In the simple parent-child data structure in Figure 13-4 *Parent to Child Instance*, the column on the right is the syntax that you would use to specify an instance of a record in the structure.

Figure 13-4 Parent to Child Instance

Sample Sequence	Instance Path
invoice	invoice{1}
line-item	invoice{1}.line-item{1}
line-item	invoice{1}.line-item{2}
reference	invoice{1}.line-item{2}.reference
line-item	invoice{1}.line-item{3}
reference	invoice{1}.line-item{3}.reference

Specifying the entire instance path for each record that you want to get data from could be a tedious task. Fortunately, the notation can be shortened to the current instance.

Current Instance

As mentioned, there is a path to every instance of a record in the tree. At any point in the processing of the source table, there will be one record that is singled out for processing. The path to that record is the **current path**. Every instance of a record along the path is the current instance for that record or segment.

References to any other record can be described by a path relative to the current path.

The use of relative indexing is not recommended. The functions $INSTANCE and $EXIST are provided to accomplish forwards and backwards references.

A relative index can not be used when specifying the context.

A relative index can be used when specifying an instance in a map assignment. If an index begins with a sign followed by a digit, a symbol, or a parenthesized expression, the index is assumed to be a relative index. Otherwise, the index is assumed to specify an absolute instance.

For the following examples, assume that the current instance is A{2} and that the value of the global J is 2.

The assignments in the following example are then interpreted as follows:

Reference	Interpretation	Value
A{+1}:Af1;	Relative	A{3}:Af1
A{-1}:Af1;	Relative	A{1}:Af1
A{+J}:Af1	Relative	A{4}:Af1
A{-J}:Af1	Relative	A{0}:Af1 (Undefined)
A{+J-1}:Af1	Absolute	A{1}:Af1
A{+(J-1)}:Af1	Relative	A{3}:Af1
A{-J+4}:Af1	Absolute	A{2}:Af1
A{-(J-4)}:Af1	Relative	A{0}:Af1 (Undefined)

Difference Between Relative and Absolute Index

A reference with a relative index that is not valid, produces the value $UNDEFINED, whereas a reference with an absolute index that is not valid produces a hard error.

An index is invalid if it is less than 1 or greater than the number of instances available. The function $EXIST should be used to determine whether or not the instance exists before using an absolute index if there is any question about the index validity.

Relative Indexes with Negative Values

If the value of the symbol in a relative index is negative, the index is rejected before being interpreted as a relative index. For example, assuming the value of J is -2:

Reference	Result
A{-J}:Af1;	Invalid instance

Specification of Off-Path References

A reference can be made to an instance other than the current instance by specifying the path to the instance. As soon as an instance is specified, the reference becomes an off-path reference and all other instances must be specified.

For example, assuming that the current instance is A{2}.B{1}.C{3}, references are interpreted as follows:

Reference	**Interpretation**
Cf1	A{2}.B{1}.C{3}:Cf1
B{2}.C{1}:Cf1	A{2}.B{2}.C{1}:Cf1
B{2}.C:Cf1	Invalid instance specification
B{+2}.C{1}:Cf1	A{2}.B{3}.C{1}:Cf1
B{+2}.C{+1}:Cf1	Invalid instance specification

In the above invalid instance specifications, the instance of C was not explicitly designated for an off-path reference.

At any juncture, references to records that share parents along a current path can be dropped off the syntax. For example, assume a current path of:

```
invoice{3}.line-item{5}.reference{1}
```

Each of these records have been established as the current instance of a record along the path. You can refer to the record simply as *reference* because the Mapper processing has already established the current path to this record.

Assume another record at the following location:

```
invoice{3}.line-item{5}.reference{2},
```

The shortened notation can be simply *reference{2}* because the record shares the current path references to *invoice* and *line-item*. Figure 13-5 *Full and Shortened References* shows full and shortened references to other records, when the current path is `invoice{3}.line-item{5}.reference{1}`.

Figure 13-5 Full and Shortened References

Full Instance Reference	Shortened Instance Reference
invoice{3}.line-item{5}.reference{1}	reference
invoice{3}.line-item{5}.reference{2}	reference{2}
invoice{3}.line-item{1}.reference{2}	line-item{1}.reference{2}
invoice{1}.line-item{1} .reference{1}	invoice{1}.line-item{1} .reference{1}

Each record in the source tree has a current instance pointer. This pointer identifies an instance of this record in relationship to the current instance of its parent record.

Each time a current path changes, the current instance pointers of all records under it are set to the first instance under the new current instance. This is true regardless of how the current path might have been changed.

The entry in the Set Context field of a map sets one of these records as the current record for the map. Being the current record means the following:

- The current instance of this current record is the starting point for any Repeat Pattern specifications that might be applied to the map. Each time the map is executed, the next instance of that current record will become the new current instance of that record.
- The path for the current instance of the current record becomes the default path for all references. Fields on the current instance of the current record do not have to be qualified with a path name. Further, where paths are needed to refer to fields on other records, the path names are relative to the path from the root to the current instance on the current record.

  If a reference is made to any record that is not on the current path, it is necessary to specify the entire path. Failure to do so may result in unpredictable results.

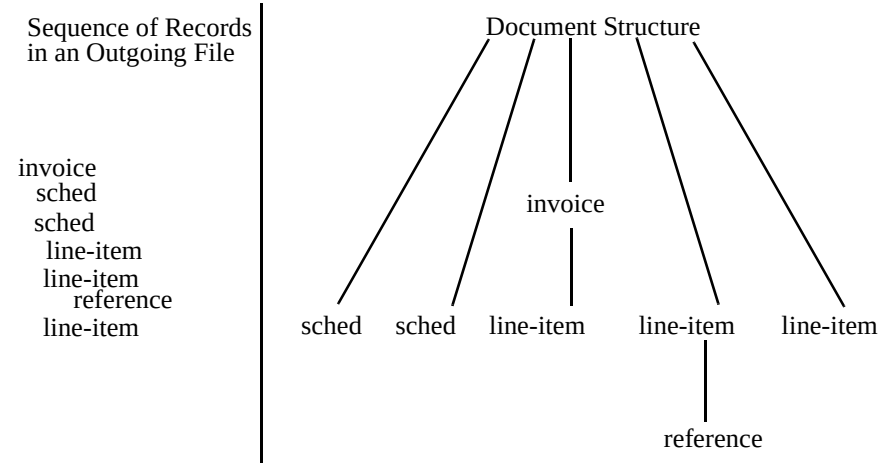
To reference the current instance of a child for another parent, the $INSTANCE function can be used to determine the instance number of the child and then an absolute index can be used.

For example, assume that the current instance is A{2}.B{2}.C{3}. To look at the current instance of C for the previous instance and following instance of B, use the following assignments:

```
K = $INSTANCE(C);
x = Cf1;                  the current value   A{2}.B{2}.C{3}
y = B{+1}.C{K}:Cf1;       the next B          A{2}.B{3}.C{3}
z = B{-1}.C{K}:Cf1;       the previous B      A{2}.B{1}.C{3}
```

The following example is similar to the one shown in Figure 13-1 *Current Instance Path*. Add a shipping schedule record under the invoice record, and assume the record sequence shown in Figure 13-6 *Explicit Instance Path*.

Figure 13-6 Explicit Instance Path



Assuming a current path of `invoice{1}.line-item{2}.reference{1}` and that you would like to reference the first sched record under the invoice record, you must specify the path as: `invoice.sched{1}` or `invoice.sched`

The first of these is reliable throughout the entire map. The second of these could reference other sched records in some cases, which could be useful in in some circumstances.

While your mapping table will compile and run if you simply specify sched (without invoice), which sched data will be returned is undefined and may therefore cause unpredictable results.

We recommend that you specify fully all references within mapping assignments and map conditions. To do this, include the name of the path from the record referenced to its 01 level parent. For segments, include the names of all segments on the path between the segment referenced and the its 02 level parent.

Examples

In any mapping assignment or map condition where the map's SET CONTEXT reference is to any segment other than PER#1, if you wish to reference the PER#1 segment (part of the N1 loop in the header of the 810), use the following:

```
BIG.N1.PER#1
```

To reliably reference the first PER#1 under the first N1, use the following:

```
BIG.N1{1}.PER#1{1}
```

Do not use PER#1 or PER#1{1} by itself.

In any mapping assignment or map condition where the map's SET CONTEXT reference is to any segment other than CTA#1, if you wish to reference the CTA#1 segment (part of the NAD group that is in the header of the EDIFACT INVOICE message), use the following:

```
NAD.CTA#1
```

To reliably reference the second CTA#1 under the third NAD, use the following:

```
NAD{3}.CTA#1{2}
```

Do not use CTA#1 or CTA#1{2} by itself.

If you know that you always want a particular occurrence, then the best method is to explicitly specify the instance that is wanted. For example, if you know that you always want to get the first CTA segment under the second NAD, specify the reference like this:

```
NAD{2}.CTA#1{1}
```

It is more common and normally more reliable to select an instance based on the value of a qualifier or other fields nearby. The method above is illustrated for those cases where you want to get a fixed instance.

An example showing how to reference application file data has been provided above.

The rules are the same for all standards even thought only X12 and EDIFACT examples are given.

We recommend that references to segments and records within the SET CONTEXT clause and the FOR EACH option of the REPEAT PATTERN clause be specified as single unqualified references. For example:

```
SET CONTEXT LINE-ITEM
```

and

```
Set Context:    SDQ
Repeat Pattern: FOR EACH PO1
```

See the section Supported Mapping Combinations for more information.

Structure Mapping (Navigation)

Structure mapping is the process of moving the current path around in the source tree in response to the instructions in the maps that the Mapper is executing. Structure mapping specifications go into the Navigation section of the **Map** screen. Structure mapping has two parts:

- Setting the context for the map
- Determining the pattern for moving the context

The source record or segment from which data will come for a particular destination map is the **context** for that map. Setting context is the process of establishing the current path to the source record or segment. A map contains two specifications for setting context. One is the specification for the starting location (the **Set Context** field on the **Map** screen and the other is the repeated pattern for moving the context (the **Repeat Pattern** field on the same screen).

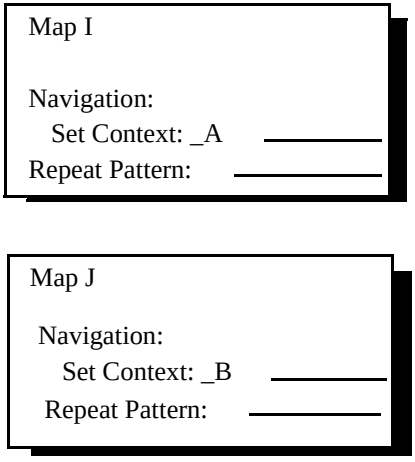
For example, two trees are shown in Figure 13-7 *Mapping Two Simple Trees*: one a simple source tree, and the other is a simple destination tree. The object is to map the data on record A into record I and the data on record B into record J. Assume that the specifications (the Record Sequence Definition) states that A and B and I and J can all have more than one record.

Figure 13-7 Mapping Two Simple Trees

Source Tree	Destination Tree
01 A	01 I
02 B	02 J

To accomplish this mapping, the Mapper has to follow the instructions in two maps, the I map and the J map. The I map sets the context to record A, as shown in Figure 13-8 *Setting Context*.

Figure 13-8 Setting Context



At the beginning of processing, the Mapper sets up an initial current path, which is always instance 1 of the top record in the tree. In the example, Map I shows setting context for the first instance at record A. The full path reference would be A{1}, but the shortened instance reference can be used because it shares the current path set up by the Mapper when it initialized.

The second part of navigation is determining the pattern for moving the context. In the example, the map for I uses the default iteration pattern, which is the auto repeat pattern. That is, it executes Map I for every instance of a record that it finds in A.

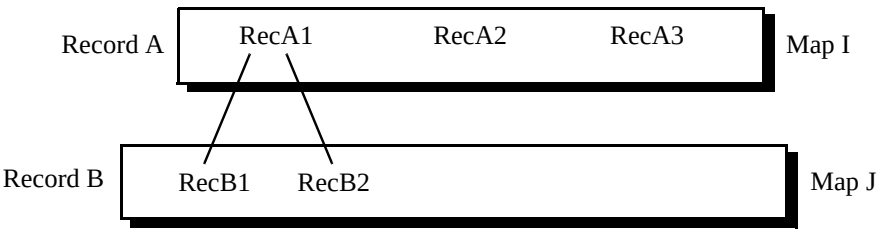
The general rule that applies to the auto repeat pattern is that the Mapper iterates through all of the records that have the same parent. Because Map I sets the context to the top record, however, everything at A is considered to be a child of an implied root, as shown in Figure 13-9 *Simple Repeat Pattern*.

Figure 13-9 Simple Repeat Pattern

The repeat pattern tells the Mapper to execute Map I once for each instance of A.

A second general rule states that every time the Mapper executes a map that builds a segment or a record, it has to process all of the maps that are subordinate in the destination tree before processing the next iteration of the same map. In the example shown in Figure 13-10 *Source Tree With Several Records*, every time the Mapper executes Map I using a record from A, it has to execute Map J.

Figure 13-10 Source Tree With Several Records



A third general rule of mapping is that each time a current instance changes (A in this case), the current instances of all subordinate records are set to the first instance subordinate to this new instance of the parent.

Notice that in Figure 13-8 *Setting Context*, Map J's Set Context specification was B. This is the shortened instance reference for A{current}.B{1}. It is instance 1 because of the third general rule. During the mapping, the Mapper sets Map J's starting context to a path that is the current instance of A and the first instance of B. It must then iterate through all of the children of the current instance of A that are in B because the repeat pattern for Map J was left blank, meaning that the Mapper will use the auto repeat pattern.

When the Mapper finishes iterating through all of the child records for Map J, it returns to Map I and continues where it left off. The Mapper builds a segment or a record from Map I using the next instance of Record A (A{2}), which in turn causes it to execute Map J again, iterating through all of the children of A{2}.

This pattern repeats until the source tree is out of data on records A and B.

Advanced Error On

On any of the maps that specify a repeat, it is possible that the map will exceed the maximum number of records or segments (depending on direction) for the destination. Normally, this is the way to terminate the iteration. However, in some applications, these could be indications that the data has been corrupted or that the description of the data is incorrect.

The map has a flag indicator that allows you to specify an error condition rather than just terminating the loop. It is the **MAX Exceeded** condition. Mark the box with an X. If the condition occurs, the Mapper aborts the document, enters an error message in the audit log, and calls the SOFT ERROR hook point. It then continues with the next document.

In the Incoming direction, this means that the Mapper aborts the document as a FAILED document. This requires manual intervention in Digital DEC/EDI to restart the document.

In the Outgoing direction, the offending document is aborted and records for that document in the application file are skipped. Processing continues with the records for the next document. You can direct the SOFT ERROR to be converted to a HARD ERROR by use of a small hook routine. The HARD ERROR will cause the records remaining in the application file to be left unprocessed. The Mapper processing terminates.

Advanced Set Context

The Set Context field on the **Map** screen identifies one or more records from the source tree that are to form the initial context for the mapping assignments on this map. Setting context has two functions:

- Specifies the default path to use when referring to data in the source tree.
- Determines the starting point for the repeat pattern used in this map.

The syntax for this entry is:

```
path [, path] ...
```

Where *path* is:

```
[record '{' instance'}' '.']... record ['{' instance '}']
```

The last instance of a record specified in the path is the default record (or segment) in the source tree. In the map assignment statements, references to fields on this record do not have to be qualified with a record name. References to fields on other records must be qualified with a the other record name, and must use the fully qualified path name.

The Set Context specification also sets the starting point for the repeat pattern used in the map. The Mapper performs the mapping assignment part of the map starting with the context at a point in the source tree and then repeats for each subsequent instance according to the repeat pattern specification.

There are several variations on the syntax used to set the context for a map:

- **The field can be left blank**

  If no context is set, there is no source context for repeat patterns so there will be no repeating. The default qualifier for field references will remain set to the context established in the previous map. Maps that reference only global variables do not need a context in the source tree and therefore do not need a context specification.

- **Specify one record or segment**

  This must be a single, unqualified record or segment name.

- **Variants**

  An entry in the Set Context field that refers to a variant in the record sequence must specify the name given to the variant as a whole or to a list of all the record types within the variant from which data is to be obtained. The Mapper filters out only those record types that are listed.

  For Example, suppose a variant contains records A,B, and C. Setting the context to A retrieves only the A record instances from the variant; the type B and C records are skipped. If you do not want to filter the records,

either the variant name or record A, B and C must be entered in the Set Context field.

Note that the variant as a whole becomes the default qualifier for the mapping assignments in this case. Unqualified fields will be mapped to the record type of the current instance. For example, suppose A, B, and C all have a field **X**. Even if this field is not in the same place on each of the record types, the appropriate X value will be retrieved regardless of the type of the current instance. However, if the field is qualified (A:X), then it will be UNDEFINED if the current instance is not of the type specified in the qualification. If a field Y is defined on A, but is not defined on B or C, the unqualified field Y will be UNDEFINED when the current record type of the variant is a B or a C.

- **Floating Segments**

  A floating segment (such as the X12 NTE segment) need only be defined in one place in the source tree and identified as FLOATING.

  When Incoming floating segments are received, they are inserted as if they were subordinate to the non-floating segment just preceding it. To get the data from these floating segments, you must create maps that reference every place in the source tree where floating segments are expected. If a segment is not there, then the reference fails and the mapping assignments on the map are not executed.

  A reference to the floating segment is made by specifying its path as with any other record as if the segment had actually been defined there. For example, if you want to set the context for a map at a note (NTE) segment subordinate to the current instance of the N1 segment, specify a Set Context as N1.NTE.

Advanced Repeat Pattern

A Repeat Pattern is how the map will be repeated. A repeat pattern starts with the record established by the Set Context field. The Mapper then proceeds to visit subsequent instances of the records in the source tree each time the map is executed.

There are various qualifiers that control how many times it will repeat if any.

- **Auto-Repeat Pattern**:

  This is the default pattern. If you leave the Repeat Pattern field blank, the Mapper uses this pattern. The Auto-Repeat Pattern repeats the mapping for each instance of the source (specified in the Set Context) until it runs out of data or exceeds the destination limit set by the Limit field or the maximum repeat specification for the record.

  If the maximum is set to 1 on either the source or the destination record, the map is not repeated. Also, if the same source record is specified on more than one map, only the highest one in the destination tree (the top-most) will actually do the incrementing. This way, all of the maps that refer to the same record see the same data.

  Specifying any other Repeat Pattern cancels the Auto-Repeat feature.

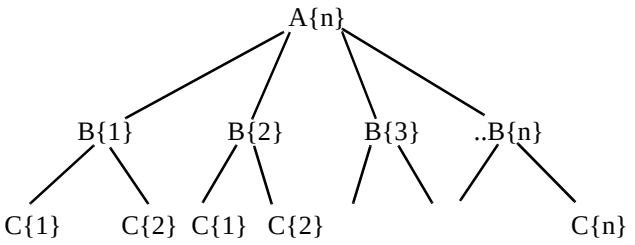
- **For-Each Pattern**:

  This pattern directs the Mapper to repeat the map not only for all children of the current parent, but for all children of all siblings of the parent. By extension, it can also can be used to iterate through as many levels as you want to specify, which in effect combines these levels. This pattern can be specified by supplying the following entry in the Repeat Pattern field:

  ```
  FOR EACH parent
  ```

  The **parent** is the highest parent that is to be incremented. The Mapper iterates through all instances of records (or segments) of the type specified in the Set Context entry, which are subordinate to all instances of this parent. For example, see the tree and map fragment in Figure 13-11 *Tree and Map Fragment.*

Figure 13-11 Tree and Map Fragment



Set Context: C;

Repeat Pattern: FOR EACH B;

- **No Change**:

  There are times when you will not want a map to be repeated or the source to be incremented. This can be done by overriding the Auto-Repeat pattern with the entry, NO CHANGE, in the Repeat Pattern field.

- **$THROWAWAY Special Value**:

  The $THROWAWAY is a special value that causes the Mapper to discard the current instance of the destination record or segment being generated as if the condition failed and to go on to the next map.

- **$ENDMAP Special Value**:

  The $ENDMAP is a special value that causes the Mapper to skip the remaining assignments in the map being processed, but use the destination instance.

New Context Parts

New context is a continuation of a map. You use it to specify a different set of assignment statements, grouped under a different condition, for the same record or segment as the first part of the map. Use a new context specification if you want to get data from one set of the records if one set of conditions is true, and a different set of records if a different set of conditions is true.

A map form contains a header, specifications for Set Context, Repeat Pattern, and Condition, and a list of mapping assignments. In the upper right corner of the map form are the NEXT PART and PREV PART fields. These allow you to step through each of the parts that can be attached to a map.

The Map Part contains a Set Default field, a Condition field, and a place for another set of mapping expressions. The Set Default specification is similar to the Set Context, except that it specifies a different default path to be used with the mapping assignments in the map part.

It does not affect the initial value for the repeat pattern. This can be useful if a lot of data is coming from more than one record or segment and you do not want to put long qualifications on fields on all records or segments except the one specified in the Set Context field.

The Condition specification on a new context part determines if the mapping expressions are to be executed. See the discussion about the Condition specification.

Advanced Data Mapping (Mapping Assignments)

Data mapping is the assignment of field data to data labels within the destination segment (in the Outgoing direction) or assignment of the data label values to fields within the destination record (in the Incoming direction).

The Mapper evaluates the mapping expressions on the right sides of each assignment. If there are no references to nonexistent records and the right side does not evaluate to one of the special constants $UNDEFINED, $ERROR, $ENDMAP, or $THROWAWAY, the assignment will be made. If this is the first assignment to the destination record (or segment depending on direction) then a destination instance is created. If there is more than one assignment to the same left side field (or data label depending on direction), within the same set of mappings for the destination record or segment, its value is overlaid.

The left side of the assignment is a field on the destination record, a data label on a destination segment, or a global variable. The right side can be an expression of just about anything: an equation using the math operators, data manipulation functions, global variables, field or data label names, constants, or any combination of these. The only thing that it cannot have on the right side is a reference to a record or segment on the destination tree other than the one being created.

Frequently, there is a one to one assignment of source to destination. That is, for an Outgoing document the left side would be a data label on the segment

associated with the destination segment and the right side would be a field on the source record.

However, in some cases the data must be manipulated in some way before being assigned. Normal type conversions are handled automatically, but if two or more fields must be combined, a field must be scaled, or some other modification, then the right side of the assignment must be an expression.

The left side of the assignment is the destination and the right side is the source of the assignment. Data type conversions from any type to any type are handled automatically.

The syntax of a mapping assignment is shown below. The square brackets ([]) mean that the enclosed parts are optional. A set of three dots means the clause can be repeated. Upper case words are keywords. Each of the parameters is explained in detail following the example.

```
Mapping Expression Diagram:
     fieldname = [ for-clause]... expression ;

For-Clause Diagram:
     FOR ( for-variable [= [min TO ] max [ , incr]] )
```

Parameters

fieldname — The fieldname is the left side of the assignment. For a mapping for an Incoming document where records are being generated, the left side can only be a reference to a global variable or to fields on the record associated with the destination record. If it is a field, it can be qualified with structure names and array subscripts as necessary to identify its position on the record.

An array subscript can be a for-variable if the right side has a FOR clause.

For a mapping for an Outgoing document, where data labels and data values are being generated for Digital DEC/EDI, the variables are reversed. The left side can only be a reference to a global variable or to data labels on the segment associated with the destination segment. This array subscript can be a for-variable if the right side has a FOR clause. If the data label is a subelement, it should not be qualified with the element name. It is not the same as a record field qualified with a structure name.

for-clause — A for-clause is used to indicate that the assignment is to be repeated during each execution of the map for the number of times indicated

in the range of the FOR statement. A for-clause is useful for filling arrays in a destination segment or record.

The for-variable, which is defined by the use of the FOR statement on the right side, can be used as an index on both sides, but its extent only ranges over the one assignment statement. If more than one index is needed, additional FOR statements can be given following the first. This makes the FOR statements nested such that the index from the FOR on the most right increments the fastest. Note that all array subscripts start with 1. A multidimensional array can be row-major or column-major depending on its definition.

for-variable — The for-variable is the name of the index variable associated with the FOR clause. Its name must be declared as a global variable, and must be unique from field names, and data label names. As a convention, these should be single letters.

min — The min is any expression made up of any component allowable in a right side expression. If not given it defaults to 1. If given, the expression must evaluate to a positive number.

max — The max argument is any expression made up of any component allowable in a right side expression. The expression must evaluate to a positive number.

incr — The incr component is an expression that indicates the size of the increment that is used to increment the for-variable. If not given, it defaults to 1. This expression must evaluate to a positive number greater than 0.

An example of the use of a FOR statement is as follows:

```
field-name[J] = for (J=1 to 6) ab{J}:label;
```

In the previous example, for each instance of the data label, make an assignment to the Jth element of the array. The FOR clause defines a single letter variable, J, which ranges from 1 to 6. The J is only defined within one assignment statement. The value inside the square brackets and braces might be expressions. The ‘ab{J}:label’ portion specifies the data label on a specific instance of record or segment **ab**.

expression — The expression is a combination of values that make up the right side of the assignment. For a mapping in the Incoming direction, (records are being generated) the right side can be an expression made up of

field names from the record associated with the destination record, data labels, global variables, for-variables, operators, functions, if-statements, or constants. The data labels must be qualified with a segment qualifier if they are not in the current instance of the segment associated with the source.

For a mapping in the outgoing direction, (so data labels and data values are being generated for a document passing through Digital DEC/EDI), it is also reversed. The right side can be an expression made up of data labels from the segment associated with the destination record, record fields, global variables, for-variables, or constants. The record fields must be qualified with a record qualifier if they are not in the current instance of the record associated with the source record.

Chapter 14 **Mapping Expressions**

This chapter describes the types of expressions you can define, and how to use them in mappings.

Expressions

An expression can be made from a combination of any of the following:

- Field
- Data Label
- Numeric Constant
- Quoted String
- Special Constant
- Global Variable
- An Equation Using Operators
- IF clause
- Function

These terms are explained in the following sections. The explanations observe the following conventions:

- A **name** used in the Mapper can be the name of a record type, segment type, a field on a record or any of its structure names, a data label in a segment, a for-variable, or a global variable. For-variables and global variables cannot be qualified. A name in the Mapper can be constructed from the following characters: A-Z, a-z, 0-9, $,-,_

  The name must contain at least one alphabetic letter. It must not begin or end with the hyphen (-).

- A **node** as used in this section is the name associated with a point in the source tree. It can refer to either a record or a segment. Node names can be qualified with an instance specification. Node names can be appended with a period between them to build a path name that can be used to uniquely identify an instance in a source tree.
- An **instance** as used in the Mapper is the occurrence of a record or segment at a node in the tree. An instance qualifier on a node specifies an instance of the node relative to the current instance of that node's parent.

Record Fields

A field is a component of a record in the application file. In the Incoming direction, field values come from the data labels by way of mapping expressions. Before assignments are made, however, the Mapper clears a new record and fills it with null characters. This means that binary integers default to a zero value, while ASCII strings have null characters, and numeric strings will contain an illegal value. If the application needs a particular value in a field as an initial value, be sure you initialize it in the mapping assignment before assigning values from data labels.

Structure Name Arrays

The field name must be qualified by structure names if the field is a part of a structure in the record definition.

If the field is an element of an array, a subscript must be specified. The subscript can be any expression that evaluates to a number. The first element is element 1 (not 0).

Incoming Mapping Tables

When a field name appears on the left hand side of an equals sign in an assignment statement the field name cannot be qualified. This is because the Mapper creates each record in order and cannot revisit them.

Outgoing Mapping Tables

For outgoing table mappings, the field name reference can be qualified by a record and instance. The record qualification is used to specify that the record being referenced is different than the default source record. To

specify a particular instance of a record, use an instance qualifier. If not given, the current instance of the specified source record is assumed.

The full syntax for a field name is defined as follows:

```
[[record ['{' instance '}']]'.']... [record
['{'instance'}']':'
      [struct '.']... fieldname ['['subscript']']]...
```

For example, if a fragment of the record layout looks like:

```
01  address                 structure;
02  city                    text size 20;
02  state                   text size 2;
```

to reference city and state in a mapping assignment say:

```
address.city
address.state
```

Frequently, during mapping assignments, data label values will be filled in with data obtained from the fields in the record of the current instance of the source record. In other words, very often one will say something like:

```
Map ID:  Line Item Map
FB Record:    IT1
SET CONTEXT:  LINE_ITEM
.
.
.
Mapping Assignments:

IT1_QUANTITY_INVOICED  =  quantity;
IT1_UNIT_PRICE         =  price;
.
.
.
```

In this case, it is not necessary or desirable to further qualify the field references (quantity, price, and so on.)

The DEFAULT Record Qualifier

The Mapper implements the concept of a **default record qualifie**r. The default record qualifier is set in the SET CONTEXT clause of the map for the mapping assignments that appear in part 1, and in the SET DEFAULT clause of the map for mapping assignments appearing in map parts 2 and higher.

Example:

```
Map ID:    Line Item Map
Part 1 of 3

FB Segment  IT1
SET CONTEXT:   Line_Item_Rec

...
Map part 2
Part 2 of 3
SET DEFAULT:   Line_Item_Adj_Rec

...
Map part 3
Part 3 of 3
SET DEFAULT:   Line_Item_Ship_Rec
```

The default record qualifier for the first map part is `Line_Item_Rec`. For part 2 it is `Line_Item_Adj_Rec`. For part 3 it is `Line_Item_Ship_Rec`.

This means that any field in the first map part which does not have a record name specified will reference a field on the current instance of the `Line_Item_Rec`. Any field in the second map part which does not have a record name specified will reference a field on the current instance of the `Line_Item_Adj_Rec`, and so on.

Explicit Record References

While most data labels will typically get their values from the fields in current instance of the default record, there are times when information must be obtained from a different record in the application file. References to data in records other than the default are always relative to the current instance of the default source record and the current instance of each of its parents.

For example, if the map has SET CONTEXT to RECORD D and you want to refer to a field on the current instance of record C, qualify the reference with the record name path between record C and its 01 level parent. Separate the last record from the field name by a colon (:). If the record sequence definition for the application file is:

```
01      A
 02     B
  03    C
 02     D
```

specify the current instance of record C as

```
A.B.C:field
```

In cases where something other than the current instance is to be referenced, the record name can be suffixed with the instance number (shown in braces).

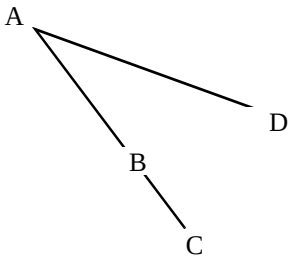
In the following example, the field referenced is from the first instance of record C. This is the first instance of record C under the current instance of the parent of C. In other words, C has a parent B. If the current instance of B is instance 8, (perhaps because the Mapper is currently processing the eighth B record), then the field referenced will be the field from B{8}.C{1}.

```
data label = A.B.C{1}:field;
```

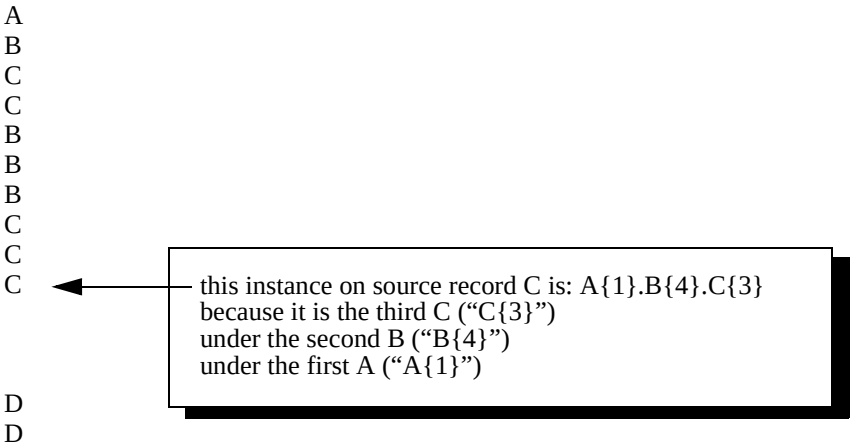
Absolute Instance References

An absolute reference to a specific instance of a field must be qualified by all parent records with the instances specified for each. For examle, assume the application file has the following structure:

File Structure:



A sample record sequence for the file structure would be: A field on the



instance pointed out in the example above is referenced as follows:

```
A{1}.B{4}.C{3}:field
```

While the Mapper will let you specify a reference to a field using a partial path, for example `C:field`*, which instance you will get is undefined and may therefore lead to unpredictable results. Therefore, when specifying a record name explicitly, always include all of the record names between the 01 level record name and the record on which the field occurs. Use care when specifying paths that use explicit record names without explicit instances since these use the current path, which may sometimes be hard to predict.*

Relative Instance References

A ‘+’ or ‘-’ unary operator on the instance number refers to the instance offset relative to the current instance. For example, C{-1} refers to the previous instance and C{+1} refers to the next instance. But C{1}, an absolute reference, refers to the first instance, regardless of what the current instance of the record is.

The only case where it is desirable or necessary to qualify a field that is in the default record is when one wishes to refer to a instance other than the current record.

One might want to see if the part number on the next line item is the same as the part number on the current line item. A fragment from a map to do this might look like:

```
Map ID:  Line Item
FB Segment: IT1
SET CONTEXT: LINE_ITEM
...
Mapping Assignments:
IT1_PRODUCT_SERVICE_ID = partnum;
nextpartnum            = if ( $EXIST(LINE_ITEM(+1))
       then LINE_ITEM(+1):partnum
       else  0;
...
```

Data Labels

A data label is the name given to an element of data in a document's internal file. A data label corresponds to a data (sub)element in the EDI standard. All data labels used must be defined in Digital DEC/EDI. The data labels are grouped by segments as defined in Digital DEC/EDI. The order in which data labels are assigned to within a map does not matter.

If the data label is a subelement for an EDIFACT segment, then the data label can be qualified with its element name in the same way that a field on a record is qualified with its structure name.

Outgoing mapping tables

When a data label appears on the left hand side of a mapping assignment the data label may not be qualified by a segment reference. Each map generates the data to create one output segment. The data labels associated with the output segment being created and global variables may appear on the left hand side of the mapping assignments. The data labels may not be qualified. This is because the Mapper creates one segment at a time.

Incoming mapping tables

The DEFAULT Segment Qualifier

In incoming mapping tables, the Mapper implements the concept of a **default segment qualifier**.

The default segment qualifier is precisely analogous to the default record qualifier described in the previous section.

In incoming tables, data labels may be qualified with a segment name of when the segment to be referenced is not the default segment. As with the record references, the segment references may include an instance qualifier.

Please read the guidelines for using instance qualifiers found in the previous section. The guidelines for data labels are the same. They are summarized here.

1. When the data label specified is part of the current instance of the default segment, it is not necessary (or desirable) to explicitly specify the segment name.
2. When referencing an instance of the default segment which is different from the current instance, specify the path between the 02 level segment and the segment being referenced and use an explicit instance number.
3. When referencing a segment that is not the default segment, fully qualify the reference with all segment names between the segment in which the data label occurs and the 02 level segment. (It is not necessary to include the 01 levels ‘heading’, ‘detail’, ‘summary’).

As in the case of fields, please use care when using explicit segment references. Especially be sure to understand whether the current instance is what is wanted.

$UNDEFINED Special Constant

When an optional data label is referenced and the data label does not exist in the input data, the value evaluates to the special constant $UNDEFINED. A value of $UNDEFINED is never assigned to the left hand side of an expression. In a simple assignment statement $UNDEFINED has the same effect as if the assignment were skipped.

Example:

In ANSI X12, the N1 segment identifies an organization. It has a qualifier that specifies the type of organization, N101, and then two different ways the organization may be identified. An organizational id qualifier and coded

organization id may be used (N103 and N104) or an alphanumeric string may be used (N102), or the sender may specify both. For example, N102 might have a value of “International Shipping Inc”. N103 might have the X12 code for Dun’s Number (a set of published codes for identifying companies) and N104 might have the Dun’s number for “International Shipping Inc.”. However, the sender might elect to use only N103 and N104 and not send N102.

In this case a fragment from a Mapper map might look like:

```
companyname = N102;
companyid = if ( N103 EQ $STR(7) )
            then $LOOKUP(DUNSLOOKUP,N104);
```

If the N1 segment does not contain a value for N102, then N102 will evaluate to the special constant $UNDEFINED. When the assignment:

```
companyname = $UNDEFINED;
```

is processed, **the value of the field companyname does not change**. It keeps whatever value it had before. Note that since it often happens that optional data is not sent, it is often a good idea to initialize fields to an appropriate value in case they are not found in the input. For example:

```
companyname =  " no name rec'd from partner";
                                /*initialize in case missing*/
companyname =  N102;
```

Qualifiers

Note that a qualified data label (in the sense of EDI Standard qualifiers, for example Product_id_code_qualifier) is just another data label.

The full definition of a data label reference is as follows:

```
[[segment ['{' instance '}']]'.']... [segment ['{'instance'}'] ':'
       [struct '.']  data label ['['subscript']']...
```

The following example shows the data label N3_ADDRESS from the first instance of a segment N3 and the data label N3_ADDRESS from the second instance of segment N3 are assigned to two fields on a record.

```
ADDRESS1 = N3{1}:N3_ADDRESS;
ADDRESS2 = N3{2}:N3_ADDRESS;
```

Data Label Attributes

The data label attributes found in these extractions define the data type of the data label.

Data label type R — The ‘Rn’ indicates that the data element is numeric type with scaling and the ‘n’ indicates the number of decimal places to the right of a fixed, implied decimal point. The element value does not have a decimal point. For example, ‘R2’ would indicate two decimal places. If the value were 125 in a data element with a type of ‘R2’, this would represent the number 1.25. The value must be left justified, but leading zeros are acceptable. Note that this corresponds to ‘N’ in the X12 data types.

Data label type N — The ‘N’ data type indicates an unscaled decimal type of data element. A decimal point is included for fractional values, but is optional for integers. The number of this type might have a leading + or - (the + is optional). The value must be left justified, but leading zeros are acceptable. Note that this corresponds to ‘R’ in the X12 data types.

Data label type ID — The ‘ID’ data type indicates a value from a predefined list of values as defined in the Digital DEC/EDI tables. The Mapper treats these the same as an ‘AN’ type. It does not verify that it is one of the predefined values. The validation will be performed by Digital DEC/EDI. The value must be left justified.

Data label type AN — The ‘AN’ data type indicates an alphanumeric string type of data element. The value must be left justified.

Data label type CH — The ‘CH’ data type is the same as AN.

Data label type DT — The ‘DT’ data type indicates a date type element with the format of ‘YYMMDD’. The size of a DT type is always 6. The value must be left justified.

Data label type TM — The ‘TM’ data type indicates a type with the format of ‘HHMM’. The size of a TM type can be 4 or 6. The value must be left justified.

SIZE — This is a required key word that introduces the size specification. The size specification is an integer indicating the maximum number of characters an element of this type can hold. This number should come from the Digital DEC/EDI tables indicating the maximum length of the data label

value. Digital DEC/EDI processing will pad to take care of the minimum field length.

Numeric Constant

A numeric constant is a decimal number such as 123. A numeric constant can be prefixed with a minus sign (-) to indicate that it is negative. Fractional numeric constants, those that contain decimal points, will be treated as signed numeric strings that are scaled as indicated by the decimal point position.

Quoted String

A quoted string can be enclosed in double or single quotes. The following table shows some examples of this.

Table 14-1 Quotation Marks in Character String Literals

Character String Value Expression	Value
“JONES”	JONES
'JONES'	JONES
'JONES"S'	JONES'S
'JONES”	[invalid]
“”””	""
“""	[invalid]
'My name is “Lefty”.'	My name is “Lefty”.
'My "handle" is “Lefty”.'	My 'handle' is “Lefty”.

Single quotation marks appear as 'nn': a double quotation mark is indicated as “nn”.

Special Constant

The Mapper supports some special constants. They have special meaning in handling mapping assignments.

- **TRUE**

  Same as a 1.

- **FALSE**

  Same as a 0.

- **$BLANK**

  $BLANK is a constant that can be used to clear the contents of a segment or record, or to test if a segment or record contains nothing.

- **$ERROR**

  Often returned as the value of a function or as the ELSE clause in an IF statement. If there is an attempt to assign this value as part of a mapping assignment, it indicates that there was an error and the processing of this document aborts. This constant can be used to force a soft error condition.

- **$UNDEFINED**

  Often returned as the value of a function or as the ELSE clause in an IF statement. If there is an attempt to assign this value as part of a mapping assignment, it indicates that no assignment is to be made. This is not an error, it only indicates that the assignment is to be ignored.

- **$THROWAWAY**

  Discard the current instance of the destination record or segment being generated as if the condition failed. Does not terminate the iteration of the next map.

- **$ENDMAP**

  Skip the remaining assignments in the map being processed, but use the destination instance.

- If no output is generated before the $ENDMAP occurs, no segment/record is generated. In this case, no output will be generated for children.

  However, if some output is generated before the $ENDMAP, the segment/record is generated and the Mapper proceeds to check the children.

- If the $ENDMAP occurs in a part of a multi-part map, it terminates the entire map. Successive map parts are not generated.

With the exception of special constants TRUE and FALSE, whenever one of these constants appears with an operator, the result of the operation is always that constant. In other words, the result of the following expression is always $UNDEFINED:

```
ABC = 1 + $UNDEFINED;
```

The exception to this rule are the operators equal (=) and not equal (<>). In these cases, you can compare for $UNDEFINED. Such a comparison results in either a TRUE or FALSE. The comparison does not work for the other special constants, $ENDMAP, $THROWAWAY, or $ERROR.

If any one of these special constants is passed as an argument to either a built-in function or a customization routine, the function or routine is not even called. The result is the special constant. The result of the following expression is $THROWAWAY:

```
ABC = $SUBSTR(1,1,$THROWAWAY);
```

If an expression contains more than one special constant, the one with the highest precedence is returned. The order of the precedence, from highest to lowest is as follows:

1. $ERROR
2. $THROWAWAY
3. $ENDMAP
4. $UNDEFINED

In the following example, the result would be $ERROR:

```
ABC = $ENDMAP + $ERROR;
```

Global Variables

A global variable is a name given to a variable that exists throughout the life of a Mapper process.

Global variables are defined on the Initializations screen. This screen is reached by selecting Table Attributes, and then Initializations. All global variables must be defined and initialized before they are used.

Since global variables exist through the life of a Mapper process they can be used as accumulators. They can also be used to hold values temporarily so expressions can be simplified.

Global variables are single-valued; they cannot be arrays.

Global variables are sometimes referred to as temporary variables since they do not exist in the input or output of the Mapper; they only exist during the processing.

Global variables can be used in expressions in the same way that Data Labels and Fields are used.

Predefined Global Variables

The Mapper provides a set of predefined global variables. The Mapper assigns values to these global variables: it is not necessary to initialize them.

You can create temporary global variables that you will be using throughout the mappings, and assign initial values to them. These are variables you will be using in addition to the predefined variables provided by the Mapper.

You create these variables by first selecting the the **Initializations** option on **The Map Navigator** when you are creating or editing a Mapping Table. The **Initializations** screen is then displayed, and the **Initializations** menu is included on the Menu Bar. Note that you can also select the **Initializations...**option from the **Mapping** menu. The option becomes available only when you are creating or editing a Mapping Table.

The **Initializations** menu enables you to create or delete an Initialization. When you create an Initialization, it is undefined. You then enter an assignment statement to declare and initialize a global variable by first selecting the undefined entry on the **Initializations** screen. For example, you might enter the following assignment statements:

```
TOTAL=0;
VAT_RATE=17.5;
```

The following rules apply to variables and the expressions you assign to them:

- **Variable:** — This is the name of a global variable. The name must be unique and not be confused with any field names in the records or any data label names in segments. To distinguish this variable from Digital-defined global variables, do not start this name with a dollar sign ($).
- **Expression:** — This is the value with which to initialize the variable. The value can be a number or it can be an expression. The expressions you use here will be mapping expressions and will follow the same rules and format requirements. You cannot refer to records or data labels here because they will not yet have been defined. You can refer to other global variables, functions, constants, and so forth.

  Temporary variables are temporary space in which the Mapper can place data while it is performing the data mappings. They can also serve as processing flags, names for constants, and in any other way you would ordinarily use global variables. The data type that the variable takes on is determined by the data type of the value assigned in an assignment statement.

Before you can use a global variable in a mapping assignment, you must declare and initialize it on the **Initializations** screen. To declare and initialize it, you assign it an initial value. If you use a variable without declaring it here, the compiler issues an “undefined variable” error message.

- Once you have assigned an initial value, that value remains in effect until changed with another assignment or until the Mapper run ends. It is *not* cleared by the Mapper between documents.
- It is possible for the global variable to change data type during the mappings if a value of a different data type is assigned.

The Mapper has several predefined global variables that it uses for processing, or to which it assigns values as a way of accessing internal data. You do not have to declare or initialize these variables before using them, although you might want to assign an initial value, depending upon usage. The Mapper’s predefined variables are listed below.

$APPLICATION

The application name. The value is assigned by the Mapper in the following precedence order:

1. From the first application specified in the Application IDs screen.
2. Overridden when you assign a value to the $APPLICATION variable in the **Initializations** screen.
3. Overridden when you assign a value to the /APPLICATION qualifier on the **Defaults** screen.
4. Overridden at run-time when you specify a /NAMED_APPLICATION qualifier in the Application Client call.

This value is used by the Mapper in the Incoming direction as one of the document selection criteria along with the $PARTNER_SELECT and the Digital DEC/EDI document type derived from the $DOCTYPE_SELECT variable.

This value is used by the Mapper in the Outgoing direction as one of the identifiers passed with a document to the Digital DEC/EDI Translation Service.

The value is made available to the mappings as the value of the application qualifier for this run.

$AUDIT_ID

This is the identifier for posting messages to audit. Within a customization routine, you can call this function for posting audit messages. One of the arguments that is needed is the identifier for this process. This is available in the $AUDIT_ID variable and can be passed to any customization routine at a Hook point or in a Mapping Expression.

$BUSINESS_REF1...$BUSINESS_REF5

One of up to five business references that may be assigned or evaluated during the mapping process. These may be used to associate user defined values with a document, for subsequent tracking, or for making decisions during the mapping process.

These business references are then populated in the document audit trail, and may subsequently be accessed using the **trade track** command, **DECEDI_TRACK** API routine, or by using the Cockpit.

With application to application routing, business references that are assigned in the outgoing direction are available in the incoming direction.

Business references might typically be used to record an invoice number, purchase order number or trading partner name associated with a document.

Business References are not available through the Mapper development interface.

$DOCCOUNT

This is a document count number that Digital DEC/EDI assigns to an incoming document. It is part of the unique document identifier by which Digital DEC/EDI knows the document, made up as follows:
`applid_direction_doccount`

`applid` is the application name you use to request the document from Digital DEC/EDI; it is the value for the $APPLICATION variable. `direction` is either I for incoming, or O for outgoing. `doccount` is the value of $DOCCOUNT.

The $DOCCOUNT applies only to the Incoming direction. When the Mapper receives a document, it places the document count value, which is provided with the document, in the $DOCCOUNT variable. This makes the value available for use during mappings. Digital DEC/EDI uses the document count as part of the identifier for a document in the audit trail.

$DOCTYPE_SELECT

This is one of the Mapper document selection criteria. The value is the Mapper document type (Object Name) as defined in the **Index of Mapping Sets** screen (under Mapping). Case is not significant.

This is *not* the Digital DEC/EDI document type, nor is it the document type of any of the standards. The value is filled in by the Mapper in the following precedence order:

1. First from the value you assign to the $DOCTYPE_SELECT variable in the **Initializations** screen.
2. Overridden when you assign a value to the **Object Name** field on the Defaults screen for the Mapping Table attributes.
3. Overridden at run-time when you specify an /OBJECT_NAME qualifier in the Application Client call.

This value is used by the Mapper only in the Incoming direction. The Mapper uses the $DOCTYPE_SELECT and $PARTNER_SELECT variables as look-up criteria in the **Index of Mapping Sets** screen to get the Digital DEC/EDI internal document type, when for polling for documents.

You can leave the $DOCTYPE_SELECT variable blank to indicate a wildcard. This causes the Mapper to accept any document for the application that matches the $PARTNER_SELECT criterion. If the selection criteria matches several mapping sets, the Mapper polls for documents matching each set.

$ERROR_CODE

This is the Digital DEC/EDI Server System error code. When the Mapper detects an error, it fills in the variable with the binary value of the error code. The error code can be passed into a customization routine at the Soft Error Hook point or the Hard Error Hook point.

$PARTNER_SELECT

This is the partner selection criterion. The value is the partner-id defined in a trading partner profile. The case is not significant. The value is filled in by the Mapper in the following precedence order:

1. First from the value you assign to the $PARTNER_SELECT variable in the **Initializations** screen.
2. Overridden when you assign a value to the **Partner ID** field on the **Defaults** screen for the Mapping Table attributes.
3. Overridden at run-time when you specify a `-partner_name` qualifier in the Application Client call.

This value is used by the Mapper only in the Incoming direction. The Mapper uses the $PARTNER_SELECT variable and the $DOCTYPE_SELECT variable as look-up criteria in the Mapping Table to get the Digital DEC/EDI internal document type to use for polling for

documents. The $PARTNER_SELECT variable can be left blank to indicate that only the Partner ID is valid. This causes the Mapper to accept all documents for the application that also match the criteria defined by the $DOCTYPE_SELECT variable.

If the selection criteria match several mapping sets, the Mapper polls for documents matching each set.

$DOCTYPE

This is the Mapper document type for the current document. The value is the Mapper document type (Object Name) as defined in the Mapping Table. It is *not* a Digital DEC/EDI document type or the document type of the standard. Usage depends on direction.

In the Incoming direction, the Mapper fills in the value from the Object Name variable relating to the document it has received. It gets this value by using the Digital DEC/EDI internal document type and the partner identification for the document as look-up criteria into the Mapping Table to choose the mappings for processing the document. The value for the $DOCTYPE variable is obtained from the chosen mapping set. The value becomes available as information for the mapping expressions.

In the Outgoing direction, the Mapper fills in this variable in the following order of precedence:

1. First from the value you assign to the $DOCTYPE variable in the **Initializations** screen.
2. Overridden when you assign a value to the $DOCTYPE variable in the **Record Sequence** popup screen. (The screen is displayed when you double-click on any line in the Record Sequence.) The value you assign is used to get the document type from information in the data as it is loaded into the source tree. With this mechanism, each document in the application file can be of a different document type.
3. Overridden when you assign a value to the Object Name on the **Defaults** screen for the Mapping Table attributes.
4. Overridden at run-time when you specify an `-object -name` in the Application Client call.

In the Outgoing direction, the Mapper uses the $DOCTYPE and $PARTNER variables as look-up criteria into the Mapping Table to determine which mapping set to use for processing the document. Both the

$DOCTYPE and $PARTNER variables must be defined by the time the Mapper is ready to select the mapping set. From the mapping set that is selected, the Mapper obtains the Digital DEC/EDI internal document type to use with the document and the instructions for mapping the document.

During the mappings, the $DOCTYPE variable is available to the mapping expressions.

$PARTNER

This is the partner identification for the document being processed. The usage depends on the direction.

In the Incoming direction, the value is obtained from the incoming document data. It is available for use in the mapping expressions.

In the Outgoing direction, the value is used to select the mapping set as well as to identify the partner for a document. The value is obtained in the following precedence order:

1. First from the value you assign to the $PARTNER variable in the **Initializations** screen. If not overridden, this value determines the partner to whom all documents processed by this map will be sent.
2. Overridden when you assign a value to the $PARTNER variable in the **Record Sequence** screen. This value is used to obtain the partner identification from the document data as it is being loaded into the source tree. With this mechanism, each document in the application file can be directed to a different partner.
3. Overridden when you assign a value to the **Partner ID** field on the **Defaults** screen for the Mapping Table attributes.
4. Overridden at run-time when you specify a `-partner -name` qualifier in the Application Client call.

In the outgoing direction, the Mapper uses the $PARTNER and $DOCTYPE variables as look-up criteria into the Mapping Table to determine which mapping set to use for processing the document. Both the $PARTNER and $DOCTYPE variables must be defined by the time the Mapper is ready to select the mapping set. The value in the $PARTNER variable will be passed through Digital DEC/EDI along with the document. During the mappings, the $PARTNER variable is available to the mapping expressions.

$USERREF

This is the user reference value for the current document. The value can be anything you want to assign, to identify the document. Usage is in the Outgoing direction only.

The value is obtained in the following order of precedence:

1. First from the value you assign to the $USERREF variable on the **Initializations** screen.
2. Overridden when you assign a value to the **User Reference** field on the **Defaults** screen for the Mapping Table attributes.
3. Overridden at run-time when you specify a `-user -reference` qualifier in the Application Client call.

The value is passed through Digital DEC/EDI along with the document.

$PRIORITY

This is the priority for the current document. Its value can be “NORMAL”, or it can be “IMMEDIATE”. The value is used only in the Outgoing direction.

The value of this qualifier decides the priority that will be requested when the document is passed through Digital DEC/EDI. The value is obtained in the following order of precedence:

1. First from the value you assign to the $PRIORITY variable on the **Initializations** screen.
2. Overridden when you assign a value to the /PRIORITY= qualifier on the **Defaults** screen for the Mapping Table attributes.
3. Overridden at run-time when you specify a /PRIORITY qualifier in the Application Client call.

$RUN_ID

The unique mapper Run ID associated with the current run. This is a string containing a six digit number.

The Run ID value is stored in the document audit trail to enable a document to be reconciled with a particular mapper run that either created or fetched the document.

This variable is not available through the Mapper development interface.

$TESTIND

This is the value of Test Indicator flag. Usage depends on direction.

In the Incoming direction, the Mapper fills in the $TESTIND variable from the value of the Test Indicator received with with the document. At this point, the Mapper verifies that this is one of the values it is expecting.

In the Outgoing direction, the Test Indicator flag can take on one of the following four values:

- `Mapper_test`
- `Translation_test`
- `Partner_test`
- `Live`

The Test Indicator flag, when applied to a particular document, is defined or defaulted as a parameter to the `post` command, or overridden during the mapping phase. When this indicator is applied to an Outgoing document, its value is determined in the following order of precedence:

1. The default is **Live**.
2. Any value specified as a Mapping Table Attribute overrides the default.
3. Any value specified to the `post` command overrides any value applied.
4. Any explicit value applied during the mapping process overrides all other values.

For more detailed information about both Incoming and Outgoing data, refer to the *Digital DEC/EDI: User's Guide — Testing the Configuration*.

After mapping for the document is complete, the value of the $TESTIND variable is checked against the test mode of the Mapping Table. The value is then passed through Digital DEC/EDI along with the document. The value must be consistent with the test indicator setting of the Digital DEC/EDI tables, or the document will be rejected.

This is the value of the current application file record. This variable is used for passing the record value to customization routines at hook points. In the Incoming direction, the $RECORD variable is filled by the Mapper, just before it calls the record hook, with the contents of the next record to be written to the application file. A customization routine associated with the record hook point can be passed the $RECORD value. It can process it in

any way it wants and then return the modified record as the value of the function. This modified record is then written to the application file. If there is no customization routine associated with the record hook point, the value of the $RECORD variable is written to the applications file.

In the Outgoing direction, the $RECORD variable is filled in by the Mapper just before it calls the customization routine at the RECORD hook point. The variable contains the next record read from the application file. It can be used as an argument to pass to the customization routine. The routine can modify the record and return it as the value of the function.

$FILENAME

This is the file name for the current application file. In the Incoming direction, the Mapper fills in the value of $FILENAME from the application file parameter in the command line at runtime. It then makes the $FILENAME variable available to the Preprocess Hook. A customization routine called at the preprocess hook can modify it and return it as the function value. This will be the name of the file created as the application file for subsequent processing, where all records for the Mapper run will be written. When the file is opened, its value will be replaced with the full file name, including the version number, and will become available for the mapping expressions. It can also be passed to a customization routine at the post processing hook point.

In the Incoming direction, the Mapper fills in the value of the $FILENAME variable from the application file name in the command line at runtime. It is then made available as a value that can be passed to the customization routine of the preprocess hook. The routine can modify it and return it as the value of the function. the Mapper will then use the modified value of the $FILENAME variable to open the application file. The file name can contain wildcards, which specify a set of files. When each file is opened, its full name, including extension, is placed in the $FILENAME variable and is available for mapping expressions and other hook points.

This is the error code. When the Mapper detects an error, it fills in the variable with the binary value of the error code. The error code can be passed into a customization routine at the soft error hook point or the hard error hook point.

This is the identifier for posting messages to audit. Within a customization routine, you can call this function for posting audit messages. One of the

arguments that will be needed is the identifier for this process. This is available in the $AUDIT_ID variable and can be passed to any customization routine at a hook point or in a mapping expression.

$RECOVERY

This is the recovery mode flag; it specifies the number of documents to skip in the application file. The Mapper does this by completely processing all documents starting with the first, but not sending them on through Digital DEC/EDI. When the Mapper is in this mode, the $RECOVERY variable has a value of 1. Otherwise, its value is 0. You can pass this variable to customization routines that have side effects to inform the routines that the current document is not really going to be sent.

$APPLICATION_ARG

This is the application type that comes from the command line. This value is filled in by the Mapper from values entered at runtime in the command line. If the value was not entered at runtime, the value of this variable will be $UNDEFINED and the Mapper uses the value in the $APPLICATION variable instead. The $APPLICATION_ARG variable can be referenced in any of the expressions.

$DOCTYPE_ARG

This is the document type that comes from the command line.

This value is filled in by the Mapper from values entered at runtime in the command line. If the value was not entered at runtime, the value of this variable will be $UNDEFINED and the Mapper uses the value in the $DOCTYPE variable instead. The $DOCTYPE_ARG variable can be referenced in any of the expressions.

$PARTNER_ARG

This is the partner identification that comes from the command line.

This value is filled in by the Mapper from values entered at runtime in the command line. If the value was not entered at runtime, the value of this variable will be $UNDEFINED and the Mapper uses the value in the $PARTNER variable instead. The $PARTNER_ARG variable can be referenced in any of the expressions.

$USERREF_ARG

This is the user reference that comes from the command line.

This value is filled in by the Mapper from values entered at runtime in the command line. If the value was not entered at runtime, the value of this variable will be $UNDEFINED and the Mapper uses the value in the $USERREF variable instead. The $USERREF_ARG variable can be referenced in any of the expressions.

$TESTIND_ARG

This is the test indicator that comes from the command line.

This value is filled in by the Mapper from values entered at runtime in the command line. If the value was not entered at runtime, the value of this variable will be $UNDEFINED and the Mapper uses the value in the $TESTIND variable instead. The $TESTIND_ARG variable can be referenced in any of the expressions.

$PRIORITY_ARG

This is the priority that comes from the command line.

This value is filled in by the Mapper from values entered at runtime in the command line. If the value was not entered at runtime, the value of this variable will be $UNDEFINED and the Mapper uses the value in the $PRIORITY variable instead. The $PRIORITY_ARG variable can be referenced in any of the expressions.

The variables you create and initialize here are executed once by the Mapper just before execution of the routine you might have assigned at the preprocess hook point at runtime.

The initial value assigned to a global variable can be any expression, just as in the mapping itself. The exception is that the expression cannot refer to fields on records or data labels on segments because at the time the initializations are performed, there is no data loaded into the source tree.

Document Audit Global Variables

The following global variables are available in the incoming direction only, and are populated with the values retrieved from the document audit trail for the document currently being processed.

These variables are blank, if **test_indicator=mapper_test**. These variables are not available through the Mapper development interface.

$INT_DOCTYPE

The internal document name as defined in the Trading Partner Agreement which uniquely identifies the type of document being processed.

$EXT_STANDARD

The external standard used when translating the document.

$EXT_VERSION

The external version of the external standard

$EXT_DOCTYPE

The external document type synonym defined by the external standard to describe the type of document (for example, INVOIC).

$DOC_CONTROL_NUM

The document's external control number.

$GRP_TYPE

The type of functionality group to which this document belongs.

$GRP_CONTROL_NUM

The control number for the functional group that this document belongs to in the transmission file.

$APP_INT_QUAL

The Application's qualifier in the Interchange header to which this document belongs.

$APP_INT_ID

The Application's identifier in the Interchange header to which this document belongs.

$PAR_INT_QUAL

The Partner's qualifier in the Interchange header to which this document belongs.

$PAR_INT_ID

The Partner's identifier in the Interchange header to which this document belongs.

$INT_CONTROL_NUM

The control number of the interchange to which this document belongs.

$FA_APP_ID

The Application ID used for the functional acknowledgement associated with the document.

$FA_DIR_IND

The direction indicator used for the functional acknowledgement associated with the document.

$FA_DOCCOUNT

The Internal Document Count used for the functional acknowledgement associated with the document.

$INT_DATE

The Interchange Date of Preparation.

$INT_TIME

The Interchange Time of Preparation

$INT_STANDARD

The Interchange Standards ID

$INT_VERSION

The Interchange Version ID

$INT_ACK_REQ

A flag indicating whether an Interchange Acknowledgement was requested.

$INT_SENDER_ID

Interchange sender address for reverse routing

$INT_RECEIVER_ID

Interchange recipient routing address

$INT_PRIORITY

Processing Priority Code

$APP_REFERENCE

Application Reference

$NUM_AREAS

Number of Areas/Message

$GRP_SENDER_ID

Group Sender ID

$GRP_RECEIVER_ID

Group Receiver ID

$GRP_SENDER_QUAL

Group Sender Qualifier

$GRP_RECEIVER_QUAL

Group Receiver Qualifier

$AGENCY_CODE

Responsible Agency code, used by GS07

$GRP_VERSION

Group Version/Release ID code, used by GS08

$TRACK_DOCCOUNT

Related Document ID for site-to-site (application to application) documents

Numeric and String Values

A numeric value is a value that represents a valid numeric quantity. It may be an optionally signed integer or decimal value. For example, the following are all numeric values:

```
1           +5          -9
22.         +22.        -22.
3.456       +3.456      -3.456
0.45        +0.45       -0.45
```

```
.678          +.678         -.678
```

A string value can consist of any sequence of characters. For example, the following are all string values:

```
"ABC"         "  abc   "    "Abc"
"*******"     "12.34  "     "!@&(*$)$)"
```

A string the can be converted to a numeric value can be used in any context where a numeric value is required.

An Expression Using Operators

Two types of operators can be used in an equation: unary and binary. The unary operators are listed in the following table:.

Table 14-2 Unary Operators

Operator	Type	Description
+	Arithmetic	Positive. In an expression it is ignored. In an instance qualifier, it indicates a relative offset from the current instance.
-	Arithmetic	Negative. Negates the value of the expression that follows. In an instance qualifier, it indicates a relative offset from the current instance.
NOT	Logical	Inverts the boolean logic. If the following expression is non-zero, it returns 0. If the expression is zero, it returns 1.

Binary operators operate on specific data types. Before an operation is performed, the value of the operands will be converted to an appropriate type. The binary operators are listed in the following table.

Table 14-3 Binary Operators

Operator	Type	Description
+	Arithmetic	Addition of two numbers.
-	Arithmetic	Subtraction of two numbers.
*	Arithmetic	Multiplication of two numbers.
/	Arithmetic	Division of two numbers.
|	String	Concatenation of two strings.
AND	Logical	Both non-zero, return 1, else 0.
OR	Logical	Boolean OR. Either non-zero, return 1, else 0.
> or GT	Relational	Greater Than compare, Number or string.
< or LT	Relational	Less Than compare, Number or string.
>=	Relational	Greater or Equal compare, Number or string.
<= or LE	Relational	Less than or Equal compare, Number or string.
= or EQ	Relational	Equal to compare, Number or string.
<> or NE	Relational	Not Equal compare, Number or string.

Note: At least one space is required between a subtraction operator (-) and variables used as operands so that the operator can be distinguished from a hyphen in a variable name.

Arithmetic Operators

Arithmetic operators require numeric valued operands. If an operand is a string but contains a numeric value, the value is converted to numeric form.

If an operand is a string that cannot be converted to a numeric value, a runtime error occurs.

String Operators

String operators can have numeric or string operands. Numeric operands are converted to string values.

Relational Operators

Relational operators can have numeric or string operands. In addition, the equality operators (EQ and NE) can have the special value $UNDEFINED as an operand.

Depending on the operands, a numeric comparison or a string comparison is performed, according to the following rules:

1. If both operands are numeric, a numeric comparison is performed.
2. If one operand is numeric and the other is string, the string operand is converted to a numeric value. If it can not be converted, a runtime error occurs and the document is aborted.
3. If both operands are strings, a string comparison is performed.

Since some constants can become strings and since global variables change type according to the value assigned, the type of an operand is sometimes difficult to predict.

Recommendation: It is recommended that comparisons using constants and global variables be written using the $ROUND function or the $STR function. If a numeric comparison is intended, use the $ROUND function. If a string comparison is intended, use the $STR function.

Example

Consider the following sequence of assignments:

```
b = 123.4;
     VALUE SET TO: "123.4"
c= "34.56";
     VALUE SET TO: "34.56"
e = $ROUND(c,2);
     VALUE SET TO: "34.56"
f = if  e > b then 1 else 2;
     VALUE SET TO: "2"
g = if  c > b then 1 else 2;
     VALUE SET TO: "1"
```

The global variables b and c contain string values. The global variable e contains a numeric value because the $ROUND function was used to force it to that type.

In the assignment to the global variable f, the numeric operand e forces a numeric comparison. The string operand b is forced to a number and the result of the comparison is false because e (34.56) is not greater than b (123.4).

In the assignment to the global variable g, both c and b are strings. A warning message is issued and a string comparison occurs. The result of that comparison is TRUE. A string comparison looks at the operands on a character by character basis and finds that the character 3 is greater than the character 1.

To ensure a numeric comparison, the assignment to g could be written as follows:

```
g = if  $ROUND(c,2)  > $ROUND(b,2) then 1 else 2;
```

Logical Operators

Logical operators require numeric-valued operands. A numeric value of zero is considered FALSE All other numeric values are considered TRUE.

A Conditional Statement Using the IF Expression

The IF expression provides a way to perform a conditional assignment. The IF expression in the Mapper is **NOT** the same as a conditional statement found in almost all programming languages. In most programming languages, the IF statement causes branching. In the Mapper, the IF expression simply determines a single value to be returned as the value of the expression.

The syntax is:

```
IF conditional-expression THEN value-if-true
                                    [ELSE value-if-false][ENDIF]
```

The conditional-expression is evaluated. If the result is TRUE or a non-zero numeric value, the value assigned to the IF expression is the value of the THEN clause.

If the conditional-expression evaluates to FALSE or zero, the value assigned to the IF expression is the value of the ELSE clause. If no ELSE clause is present, the value assigned to the IF expression is $UNDEFINED.

If the relational value evaluates to $ENDMAP, $THROWAWAY, or $ERROR, it is an illegal value.

If the IF expressions are nested, the ENDIF keyword is required at the end.

The following examples show possible syntactic combinations for the IF expression:

```
ABC = IF A THEN B;
ABC = IF A > 1 THEN B ELSE C;
ABC = IF $EXIST(record1) THEN "We have it";
ABC = IF record1{1}:field = "A" THEN 26 ELSE $UNDEFINED;
```

The following examples show possible syntactic combinations of nested IF expressions. When expressions are nested, each must be terminated with the ENDIF keyword.

```
ABC = IF A=1 THEN 25
         ELSE IF A=2 THEN 28
              ELSE IF A=3 THEN 32
                   ELSE 0
                   ENDIF
              ENDIF
         ENDIF;

ABC = IF A=1 THEN
             IF B = 1 THEN C ELSE D ENDIF
          ELSE
             IF B = 1 THEN D ELSE C ENDIF
       ENDIF
```

All the parts of a conditional expression are evaluated even though the result of the first part determines the result of the expression. If a dependency exists among the logical expressions that make up the conditional expression, the dependency must be made explicit by the use of the IF expression.

As an example of an incorrect statement, consider the following:

```
a = IF ( n <> 0 AND $SUBSTR( 1, n, PRICE ) <> "" )  THEN PRICE;
```

The $SUBSTR function causes a runtime error if a negative length is specified. The above statement tests for a negative length. However, the second operand of the AND is evaluated even though the first operand is FALSE and thus a runtime error occurs.

In the Mapper, the correct way to perform this test is to write it as follows:

```
a = IF ( n <> 0 ) THEN
            IF ( $SUBSTR( 1, n, PRICE ) <> ""  THEN PRICE;
```

In this case, the subordinate IF statement is executed only if the first IF statement is TRUE.

Operator Precedence

The precedence of operators in an expression is as follows from the weakest (meaning last operated upon) to the strongest (first to be operated on). Within each precedence level the operators are applied left to right in a statement.

```
7) +, - (unary)
6) *, /, |
5) +, -
4) <, >, <>, =, >=, <=
3) NOT
2) AND OR
1) IF THEN ELSE
```

Assume the following example:

```
1 + 2 * 3
```

The answer is 7, not 9, because the multiplication has a higher precedence level. Multiplication operation occurs before addition. Therefore, it would be the same if written as follows:

```
1 + (2*3)
```

You can change the order of precedence by putting in parenthesis to force the software to evaluate the operations within the parenthesis before other operations. In the previous example, to cause the addition operation to occur before the multiplication, you would have to write it as follows:

```
(1+2) * 3
```

The operators AND and OR are at the same precedence level. Therefore, when using combinations of AND and OR in IF statements or expressions, be sure to use parenthesis to explicitly identify the precedence.

Math Precision

The Mapper maintains up to 31 digits of precision with a scale factor between -127 and +127 in most cases.

In rare circumstances, the Mapper shows more precision than this in the map log. These large values are not normally encountered in business processing.

When fields, data labels, globals, and literals are combined in an expression using one of the math operators, the math is done using the scaled decimal string math.

The numeric data types and strings are converted to a Left Separate Numeric String. Precision is 31 significant digits and an exponent ranges from -127 to +127. In all math operations, the result is rounded to 31 significant digits.

In some cases, the result of the math operation will have more digits to the right of the decimal point than can be used by in record field or data label, depending on the field or data label size and scale factor. The assignment will round the fractional part without giving an error.

Functions

The Mapper provides a comprehensive set of data manipulation routines for modifying data in cases where standard data type conversions are not sufficient. The routines are listed in the following table. If this set of functions is not sufficient, you can implement your own manipulation routines using the customization functions. See *Chapter 17 Using Hooks to Customize the Mapper*

.

Table 14-4 Functions

Operator	Description
$BEGINDOC()	Returns a string containing the name of the node that contains the BEGIN DOCUMENT clause for the current document data set.
$DATE(format,date)	Date and Time Conversion. The following formats may be used: 1. OpenVMS binary to MM/DD/YY. 2. OpenVMS binary to DD-MMM-YYYY HH:MM:SS. 3. OpenVMS binary to MM-DD-YYYY HH:MM:SS. 4. OpenVMS binary to UNIX binary date format. 5. OpenVMS delta binary to HHMM (DT format). 6. OpenVMS binary to YYMMDD HHMM. 7. MM/DD/YY [,HHMM] to OpenVMS binary. 8. DD-MMM-YYYY HH:MM:SS to OpenVMS. 9. MM-DD-YYYY [,HHMM] to OpenVMS binary. 10. UNIX binary date to OpenVMS binary. 11. YYMMDD [,HHMM(DT format)] to OpenVMS binary. 12. HHMM to OpenVMS delta binary.

Table 14-4 Functions (continued)

Operator	Description
$DATE_CONVERT (input_format, output_format, date)	This function converts date into the format specifiedby output_format. The format of date must be specified in input_format. date must be of date type TEXT. input_format and output_format can include any combination of the following format specifiers: %a abbreviated weekday name %A full weekday name %b abbreviated month name %B full month name %c (see note below)) local date and time representation %d day of the month (01-31) %H hour (24-hour clock) (00-23) %I hour (12-hour clock) (01-12) %j day of the year (001-360) %m month (01-12) %M minute (00-59) %p local equivalent of AM or PM %S second (00-59) %U week number of the year (Sunday as 1st day of week) (00-53) %w weekday (0-6, Sunday is 0) %W week number of the year (Monday as 1st day of week) (00-53) %x (see note below) local date representation %X (see note below) local time representation %y year without century %Y year with century %Z time zone name, if any %% % Use %c, %x and %X only in output format string. This function is not available through the Mapper development interface.

Table 14-4 Functions (continued)

Operator	Description
$EXIST(node)	Returns TRUE if specified node's instance exists. Returns FALSE if it does not.
$INSTANCE(node)	Returns the current instance number for node. If there are no elements, return $UNDEFINED. The node argument can be qualified with an instance number.
$INT(expr)	Converts expr to an integer by truncating any fractional part. The argument must be a numeric value.
$LEN(expr)	Returns the current length of the string.
$LOOKUP_SHARED(expr1, expr2)	Look in shared lookup table named expr1 using the value of expr2 as the key, and return its corresponding value. Note that expr1 is a string expression, and must be enclosed within quotes. You may create the table name dynamically by concatenating several strings together within expr1. In the following example, a partner specific lookup table is used. `SEG03 = $LOOKUP_SHARED("LOOKUPS_" | $PARTNER | "_TBL", FIELNAME);` If no value is found, then return $UNDEFINED. This function is not available through the Mapper development interface.
$LOOKUP(tablename, expr)	Look in lookup table using the value of expr as the key and return its corresponding value. If no value is found, return $UNDEFINED.
$PAD(str,len,char)	Pad string str to length of len with the value of char as the pad character. A negative length specifies right justification. The argument len must be a numeric value.

Table 14-4 Functions (continued)

Operator	Description
$ROUND(expr, fractional_digits)	Limit the precision of a fractional numeric string (a string containing a decimal point) by rounding. The argument fraction_digits must be a numeric value. It specifies the maximum number of digits to carry on right of decimal point.)
$STR(expr)	Explicit convert expr to a string.
$STRCHR(str,charset)	Position in str of first character that matches any character in the string charset. Returns 0 if none are found. (First character position is 1).
$STRNCHR(str, charset)	Position in str of first character that DOES NOT match any character in the string charset. Returns 0 if none are found. (First character position is 1).
$STRPOSITION(str, substr,start)	Returns position of substring in str. Returns 0 if sub-string not in str.
$SUBSTR(start, len,expr)	Returns a substring value. Starting with position ‘start’ in the string ‘expr’ and taking ‘len’ characters. First character position is 1. The arguments len and start must be numeric values. If len is 0, returns the null string. If len < 0, produces an error. If start specified a position beyond the end of the string, the null string is returned. If the sum of start and len exceeds the length of the string, the string starting at start and continuing to the end of the string is returned.

Table 14-4 Functions (continued)

Operator	Description
$TIMESTAMP(numeric_format)	Returns the current date/time in specified format. The format may either be one of the following numeric values, or a string specifying formatting characters. The following numeric formats may be used: 1. OpenVMS binary to MM/DD/YY. 2. OpenVMS binary to DD-MMM-YYYY HH:MM:SS. 3. OpenVMS binary to MM-DD-YYYY HH:MM:SS. 4. OpenVMS binary to UNIX binary date format. 5. OpenVMS delta binary to HHMM (DT format). 6. OpenVMS binary to YYMMDD HHMM. 7. MM/DD/YY [,HHMM] to OpenVMS binary. 8. DD-MMM-YYYY HH:MM:SS to OpenVMS. 9. MM-DD-YYYY [,HHMM] to OpenVMS binary. 10. UNIX binary date to OpenVMS binary. 11. YYMMDD [,HHMM(DT format)] to OpenVMS binary. 12. HHMM to OpenVMS delta binary.

Table 14-4 Functions (continued)

Operator	Description

$TIMESTAMP(string_format)	Returns the current date/time in specified format. The format is a string specifying formatting characters:
	`%a abbreviated weekday name`
	`%A full weekday name`
	`%b abbreviated month name`
	`%B full month name`
	`%c local date and time representation`
	`%d day of the month (01-31)`
	`%H hour (24-hour clock) (00-23)`
	`%I hour (12-hour clock) (01-12)`
	`%j day of the year (001-360)`
	`%m month (01-12)`
	`%M minute (00-59)`
	`%p local equivalent of AM or PM`
	`%S second (00-59)`
	`%U week number of the year (Sunday as 1st day of week) (00-53)`
	`%w weekday (0-6, Sunday is 0)`
	`%W week number of the year (Monday as 1st day of week) (00-53)`
	`%x local date representation`
	`%X local time representation`
	`%y year without century`
	`%Y year with century`
	`%Z time zone name, if any`
	`%% %`
	This format of the function is not available through the Mapper development interface.

$TRIM(str)	Trim trailing spaces and tabs from string str.
$LOGICAL(expr)	UNIX environment variable from any table and return its value. Returns $UNDEFINED if there is no translation. The logical name parameter should be placed in quotation marks.

Expression Examples

The following are examples of math operations and conversions:

```
A = 1/2;
      VALUE SET TO: "0.5"
A = 4/2;
      VALUE SET TO: "2"
A = 1/3;
      VALUE SET TO: "0.33333333333333333333333333333333"
A = 1000000/10;
      VALUE SET TO: "100000"
A = 10000000000/0.00000000001;
      VALUE SET TO: "1000000000000000000000"
A = 1/0.000000000000000000001;
      VALUE SET TO: "1000000000000000000000"
A = (3+4)/2;
      VALUE SET TO: "3.5"
A = 3+4/2;
      VALUE SET TO: "5"
A = (3+4)*4/2;  /* 14 */
      VALUE SET TO: "14"
A = 0;
      VALUE SET TO: "0"
A = A+1;
      VALUE SET TO: "1"
A = "1234" + 1;
      VALUE SET TO: "1235"
A = "  1234  " + 1;
      VALUE SET TO: "1235"
A = "  1234.5  " + 1;
      VALUE SET TO: "1235.5"
A = "1234.5" / 3;
      VALUE SET TO: "411.5"
A = "-1234.5" + 1;
      VALUE SET TO: "-1233.5"
A = 1234.5678000000 * 1.5;
      VALUE SET TO: "1851.8517"
A = 1234.5 + 1;
      VALUE SET TO: "1235.5"
A = 1234.5 - 1;
      VALUE SET TO: "1233.5"
A = 1234.5 / 3;
      VALUE SET TO: "411.5"
A = 1234.5 * 3;
      VALUE SET TO: "3703.5"
```

The following are examples of IF expressions:

```
A = IF (B<>$UNDEFINED) THEN B;
      VALUE SET TO: $UNDEFINED

B = 3;
A = IF (B=1) THEN "first"
        ELSE IF (B=2) THEN "second"
           ELSE IF (B=3) THEN "third"
           ENDIF
        ENDIF
    ENDIF;
      VALUE SET TO: "third"
```

The following are examples of Built-in functions:

```
A = 1234;
      VALUE SET TO: "1234"
A = $ROUND(A,2);
      VALUE SET TO: "1234.00"  <== Note precision is extended
A = A + 1;
      VALUE SET TO: "1235"     <== Note addition removed
                                                    unused 0's
A = $ROUND(123.456,6);
      VALUE SET TO: "123.456000"
A = $ROUND(123.456,6) + 1;
      VALUE SET TO: "124.456"
A = $ROUND(A,0);
      VALUE SET TO: "124"      <== Fractional value removed.
A = $ROUND("",4);
      VALUE SET TO: "0.0000"   <== Null or blank string is 0
```

The following are examples of built-in String Manipulation Functions:

```
A = $SUBSTR(5,15,"XXXXshould get thisXXXXX");
      VALUE SET TO: "should get this"
A = $SUBSTR(1,20,A);
      VALUE SET TO: "should get this"
A = $STRCHR("String","i");
      VALUE SET TO: "4"
A = $STRNCHR("String","tSr");
      VALUE SET TO: "4"
A = $STRPOSITION("String","ring",1);
      VALUE SET TO: "3"
A = $TRIM("String        ");
      VALUE SET TO: "String"
A = $PAD("String",20,"*");
      VALUE SET TO: "String**************"
```

```
A = $PAD("String",3," ");
      VALUE SET TO: "String"
A = $LEN("String");
      VALUE SET TO: "6"
A = $INT(1234.5);
      VALUE SET TO: "1235"
A = $STR(1234.5) | "ABC";        <== $STR function is redundant
                                      in this case because of
      VALUE SET TO: "1234.5ABC"       automatic conversions.
A = $TOLOWER("String");
      VALUE SET TO: "string"
A = $TOUPPER("String");
      VALUE SET TO: "STRING"
A = $TIMESTAMP(1);
      VALUE SET TO: "02/08/91"
A = $TIMESTAMP(2);
      VALUE SET TO: " 8-FEB-1991 17:39:02.35"
A = $TIMESTAMP(3);
      VALUE SET TO: "02-08-1991 17:39:02"
A = $TIMESTAMP(4);
      VALUE SET TO: "666034742"
A = $TIMESTAMP(5);
      VALUE SET TO: "1739"
A = $TIMESTAMP(6);
      VALUE SET TO: "910208"
A = $TIMESTAMP(7);
      VALUE SET TO: " 8-FEB-1991 17:39:02.00"
B = $DATE(1,A);
      VALUE SET TO: "02/08/91"
B = $DATE(2,A);
      VALUE SET TO: " 8-FEB-1991 17:39:02"
B = $DATE(3,A);
      VALUE SET TO: "02-08-1991 17:39:02"
B = $DATE(4,A);
      VALUE SET TO: "666034742"
B = $DATE(5,A);
      VALUE SET TO: "1739"
B = $DATE(6,A);
      VALUE SET TO: "910208"
B = $DATE(7,"02/08/91");
      VALUE SET TO: " 8-FEB-1991 00:00:00.00"
B = $DATE(7,"02/08/91,09:56");
      VALUE SET TO: " 8-FEB-1991 09:56:00.00"
B = $DATE(8,"08-FEB-1991 09:56:00");
      VALUE SET TO: " 8-FEB-1991 09:56:00.00"
B = $DATE(9,"02-08-1991,0956");
      VALUE SET TO: " 8-FEB-1991 09:56:00.00"
B = $DATE(10,1234);
```

```
      VALUE SET TO: " 1-JAN-1970 03:30:42.12"
B = $DATE(11,"9102080956");
      VALUE SET TO: " 8-FEB-1991 09:56:00.00"
B = $DATE(12,"0956");
      VALUE SET TO: "   0 09:56:00.00"
```

Mapping Language Keywords

The following table defines the keywords used in the Mapping Language, which have special meaning to the Mapping Table Compiler.

These keywords should *not* be used as identifiers within a Mapping Table because the Mapping Table Compiler will reject them. For example, do not use them as record or field names, or as global variable names.

Table 14-5 Keywords

Keyword	Useage
ALPHABETIC	data type
AND	operator
AN	document data type
ARRAY	array data type
BEGINS	record begins document
BYTE	data type
BY	expression
CHANGES	navigation option
CHARACTERS	data type
CH	document data type
COLUMN_MAJOR	array order
DATATYPE	data type
DATE	data type
DECIMAL	data type
DESCRIPTOR	argument passing mechanism
DIGITS	data type

Table 14-5 Keywords (continued)

Keyword	Useage
DT	document data type
EBCDIC	data type
ELSE	conditional expression
ENDIF	conditional expression
ENDVARIANT	data structure
EQ	operator
FILLER	filler field
FLOATING	data type
FOREACH	navigation option
FOR	expression
GE	operator
GT	operator
HOOK	customization function
ID	document data type
IF	conditional expression
INTEGER	data type
IS	data type
JUSTIFIED	data type justification
LEFT	left justified
LE	operator
LIMIT	error condition
LONGWORD	data type
LT	operator
MANDATORY	requirement
MANY	occurrences

Table 14-5 Keywords (continued)

Keyword	Useage
MAX	occurrences
MAX_EXCEEDED	error condition
MIN	occurrences
NAVIGATION	navigation
NE	operator
NOCHANGE	navigation option
NOT	operator
NUMERIC	data type
N	document data type
OCCURS	array specifier
OPTIONAL	requirement
OR	operator
OUT_OF_DATA	error condition
OVERPUNCHED	sign is overpunched
PACKED	data type
QUADWORD	data type
REAL	data type
RECOGNITION	recognition expression
REFERENCE	argument passing mechanism
REPEAT	repeat pattern
RIGHT	right justified
ROW_MAJOR	array order
SCALE	data type
SEPARATE	sign is separate
SIGNED	signed

Table 14-5 Keywords (continued)

Keyword	Useage
SIGN	data type
SIZE	data type size specifier
STRUCTURE	record structure
TERMINATES	record terminates document
TEXT	data type
THEN	conditional expression
TM	document data type
TO	expression
TYPE	data type
UNSIGNED	unsigned
VALUE	argument passing mechanism
VARIANT	data structure
VARYING_STRING	data type
WORD	data type
ZONED	data type

Chapter 15 **Lookup Tables**

This chapter describes how to define a Lookup Table.

A Lookup Table is a list of codes and corresponding values that you can define. The Mapper refers to a Lookup Table to substitute particular code for its corresponding value.

For example, your company may have a set of codes that correspond to the sizes and types of packages that you use in selling a particular line of products. In this case, 1KB might correspond to a One-Kilo Box: 1_5LB might correspond to a 1.5 Litre Bottle.

Lookup Tables can be stored in two ways:

- Privately within a Mapping Table
- As a Shared Lookup Table in the Digital DEC/EDI server

In creating a Lookup Table, you can create its “mirror image” quickly by copying it, and selecting a menu option to reverse its codes and corresponding values.

Private and Shared Lookup Tables

Private Lookup Tables are stored within a Mapping Table. All translations are private to the Mapping Table; they can be used by any of the mappings in the Mapping Table.

Shared Lookup Tables are stored in the Digital DEC/EDI database on the server. There are advantages in using Shared Lookup Tables over Private Lookup Tables.

The advantages are:

- One Shared Lookup Table can be referenced from all Mapping Tables on the server. This reduces the development effort required per Mapping Table.
- When Lookup values need to be updated, only the Lookup Table on the server database will need to be updated, instead of individual Mapping Tables.
- A Shared Lookup Table is loaded automatically into a memory cache when first used, so successive calls to translations within that table will be very fast.

The name of a Private Lookup Table can be used in the mappings as the first argument of a $LOOKUP function. The name of a Shared Lookup Table can be used in the mappings as the first argument of a $LOOKUP_SHARED function

You can define any number of tables, privately and shared.

Shared lookups are not available through the Mapper development interface.

Codes and Values

Lookup Table translations are global; they can be used by any of the mappings in a Mapping Table.

You can define any number of tables. The values of the table are used in a mapping definition to specify a type of data manipulation.

Creating and Editing Lookup Tables

To begin, select the **Lookup Tables** option from the **Map Navigator**. Alternatively, you can select the **Lookups...** option from the Mapping menu in the Mapping Table Editor. When you use either alternative, the Lookups menu option is displayed on the Mapping Table Editor's Menu Bar. The Lookup Table Window is also displayed.

The Lookups menu has the following options:

- Create
- Modify
- Delete
- Sort
- Reverse
- Load
- Store
- Recache

Before you can define any Lookups, you need to create a Lookup Table and give it a name. You can then populate this with lookup codes and their corresponding values.

You can edit an existing Lookup Table (or populate a new one) either by double-clicking on it, or by selecting the table you want to edit. You then select the **Modify** option and the **Edit Lookup** dialog is displayed.

The fields for the **Edit Lookup** dialog are:

From: This is the search value for the lookup. During a mapping, the second argument of the $LOOKUP function is compared with each of the values in this column using a string compare. When a match is found, the function returns the corresponding value in the “to value” column.

If a value of spaces is desired, it must be enclosed in double quotes. Shared Lookup Tables do not support a value of spaces.

To: This is the value returned by a $LOOKUP function when a match is made. If a value of spaces is desired, it must be enclosed in double quotes. Shared Lookup Tables do not support a value of spaces.

Creating, Editing, and Storing Shared Lookup Tables

Before you can create a Shared Lookup Table, you need to Load the Server Lookups. This must be done *regardless* of the existence of any Lookup Tables on the Server.

After editing Shared Lookup Tables, they must be stored on the Server. This is done by selecting the **Store** option on the **Lookups** menu.

It is recommended you use the Mapping Table Editor to create, modify and update the Mapper Shared Lookups table. However, you may also update the Shared Lookups Table directly on the Server node.

Tru64 UNIX It is possible to update the Shared Lookup Tables by writing a script and executing it on the server. An example of such a script can be found in:

```
/usr/examples/decedi/mapper/decedi_msl_insert_ora.sql
```

OpenVMS It is possible to update the Shared Lookup Tables by creating or editing a file that conforms to the RMS FDL defintion found in:

```
DECEDI$DATA:DECEDI$MAPPER_SHARED_LOOKUPS.FDL
```

Once you have updated the file, use the `CONVERT/FDL` command to ensure that the file structure conforms to the above RMS FDL definition. You need to shut down the Digital DEC/EDI Server and copy the updated file to:

```
DECEDI$DATA:DECEDI$MAPPER_SHARED_LOOKUPS.DAT
```

After copying the updated file, restart the Digital DEC/EDI Server.

Mapper Shared Lookups Table

Tru64 UNIX The Digital DEC/EDI Database Name is `decedi_db`. The Shared Lookups Table is used by the Mapper, and defines a set of simple one-to-one mappings for the specified name.

Physical Implementation	SQL table MSL
Structure Name	msl

OpenVMS

The Shared Lookups Table is implemented as an RMS file, whos eattributes are defined by the FDL definition:

```
DECEDI$DATA:DECEDI$MAPPER_SHARED_LOOKUPS.FDL
```

The Shared Lookups table defines a set of simple one-to-one mappings for the specified name.

File Name	DECEDI$DATA: DECEDI$MAPPER_SHARED_LOOKUPS.DAT
FDL	DECEDI$DATA: DECEDI$MAPPER_SHARED_LOOKUPS.FDL

Table 15-1 Mapper Shared Lookups Table

Field	Tag	Type	Description
Lookup Table Name	looktab_s	CHAR*62	The Lookup Table name.
From Value	from_s	CHAR*255	The From value within the Lookup Table which is used to obtain the value to be returned
To Value	to_s	CHAR*255	The value corresponding to the specified From value
Keys			
Primary (Unique)	= looktab_s + from_s		
Secondary	= looktab_s		

Recaching Shared Lookup Tables

Once the Shared Lookup table has been modified, it must be recached by the Mapper in order for the lookups to become effective. This may be done, either by using the **Recache** option from the **Lookups** menu in the Mapping Table Editor.

Tru64 UNIX You may also recache the Shared Lookups by using the following command on the Server:

```
# /usr/sbin/decedi_recache_lookups
```

Sorting and Reversing Lookup Tables

It is possible to sort translations with a Lookup table. The sort order can be either ascending or descending alphabetical order.

To sort the translations in a Lookup table, select the **Sort option** on the **Lookups** menu.

You can reverse the codes and values of all translations in a Lookup table. By doing this, you can create a Lookup table that can be used by a Mapping table for documents sent in the opposite direction. This can be particularly useful with File to File processing.

To reverse the codes and values in a Lookup table:

1. First create a Shared Lookup Table for one direction; outgoing for example.
2. Create a copy of that table, and rename it.
3. Select the **Reverse** option on the **Lookups** menu to reverse the values. The **From Value** becomes the **To Value** and vice versa.

Once stored, this new Shared Lookup Table can be used for a corresponding incoming map.

Chapter 16 Supported Mapping Constructs

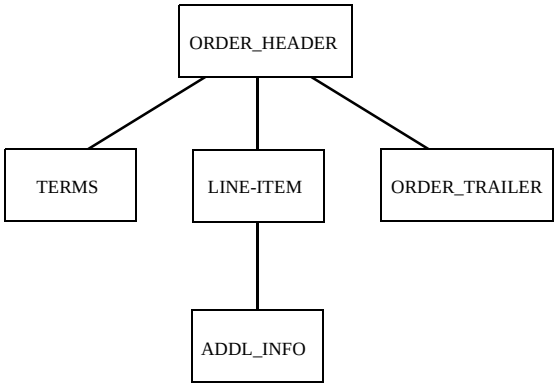
You can use the Mapper for specifying many possible mapping combinations. This chapter describes the Mapper language constructs which are supported, and the rules for combining constructs.

Make sure that any mapping table you develop conforms to the rules listed in this chapter, and uses only the constructs given here.

Application File, Records, and Tree Structure

In the Mapping Table Editor, the application file is defined as a hierarchy or tree structure. Each of the record types is assigned to a place on the tree. Similarly, the EDI Document is defined as a hierarchy or tree structure, and each of the segment types is assigned to a place on the tree.

The Mapping Table Editor processes one document at a time so the file is defined in terms of the records that make up a document. The following diagram shows the representation of the source tree for a purchase order:



Suppose you had an application file of purchase orders with such a tree structure. You would describe it to the Mapping Table Editor as follows:

Level Number	Record Name
01	ORDER_HEADER
02	TERMS
02	LINE_ITEM
02	ORDER_TRAILER

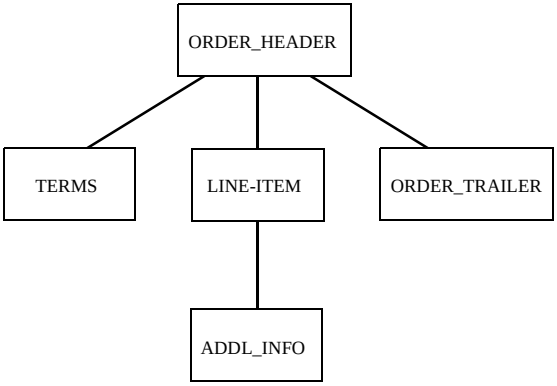
In an application file description, you would normally have only one record at Level 1 which defines the start of a document or batch of documents. All other records are subordinate to that, and would therefore have higher level numbers.

Relationships Between Records

Normally, when a record type is shown as a child of another record type, you can think of the child record type as being owned by its parent.

For example, the line items make up the purchase order, and all of the information in the purchase order header, such as order date and purchase order number, apply to all of the line items.

The following terminology is useful when discussing the tree structure, an example of which is repeated here:



- ORDER_HEADER is the **parent** of LINE_ITEM.
- LINE_ITEM is the **child** of ORDER_HEADER and ADDL_INFO is the **child** of LINE_ITEM.
- TERMS and LINE_ITEM and ORDER_TRAILER are **siblings**.
- The **ancestors** of ADDL_INFO are:

  LINE_ITEM

  ORDER_HEADER
- The ancestor of TERMS is:

  ORDER_HEADER
- The descendants of ORDER_HEADER are:

  TERMS

  LINE_ITEM

  ADDL_INFO

  ORDER_TRAILER
- A line of descent starts with the top record and follows a path to one of the bottom records. This example has three lines of descent:

  ORDER_HEADER * TERMS

  ORDER_HEADER * LINE_ITEM * ADDL_INFO

  ORDER_HEADER * ORDER_TRAILER

- TERMS is on the ORDER_HEADER * TERMS line of descent.
- LINE_ITEM is not on the ORDER_HEADER * TERMS line of descent.

Definition of Mapping

Mapping is the process of taking data from the source and constructing the output. The output is called the destination.

A mapping creates one destination record or segment at a time. (In an outgoing table segments are created; in an incoming table, records are created.) The unit that is used to create a segment or record is called a map. A set of maps that is used to create a complete document is called a mapping set.

Each mapping set is associated with one application file format and one EDI document definition; that is, a particular standard, version and message.

Source and Destination

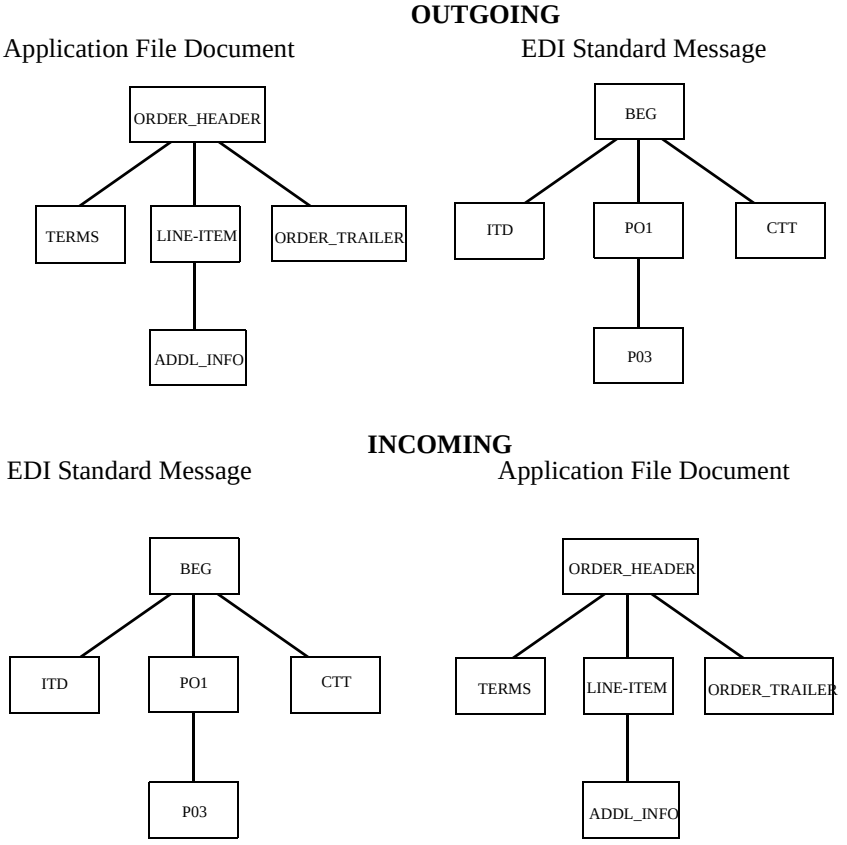
The information in the source and destination trees depends on the whether the document direction is outgoing or incoming:

- In an *outgoing* direction, documents are read from the application file one at a time and used to produce the information that Digital DEC/EDI uses to create EDI standard messages. The source tree is a representation of the document in the application file. The destination tree is a representation of the information in the EDI Standard Message.

  The source tree therefore consists of *records* from the Application File, and the destination tree consists of *segments* destined for the EDI standard message.
- In the *incoming* direction, the source tree is a representation of the EDI standard message, while the destination tree is a representation of the document in the application file.

  The source tree therefore consists of *segments* from the EDI standard message, and the destination tree is made up of *records* destined for the Application File.

This is illustrated in the following diagram.



As you read this chapter, bear in mind that the content of source and destination trees can be records or segments, according to the direction. Sometimes the term **record/segment** or **segment/record** is used, where the direction is not important.

The Mapping Table

A mapping table is used to define how to process one application file format. Each mapping table is either an INCOMING table or an OUTGOING table.

One-to-One Mapping

This is the simplest case where:

- For an outgoing document, all of the data used to create a destination segment comes from one single source record, and the source record is not used in another Set Context line or FOR EACH Repeat Pattern, anywhere else in the mapping set.
- For an incoming document, all of the data used to create a destination record comes from one single source segment, and the source segment is not used in another Set Context line or FOR EACH Repeat Pattern, anywhere else in the mapping set.

An instance of one-to-one mapping is where all of the data for a destination segment/record comes from a single source record/segment. For example, in the diagram in *Source and Destination* on page 16-4, the P01 segment data maps entirely onto the LINE_ITEM record.

The following syntax is needed for the one-to-one case:

Set Context	Source record/segment
Repeat pattern	Blank
Map part set default	May be used

If a particular source record/segment will be referenced on multiple Set Context lines, see: *Splitting* on page 16-6.

Splitting

A source record may be split into multiple destination segments, or a source segment may be split into multiple records, in the following ways:

- *Splitting Into Independent Segments/Records* on page 16-7 describes the case where the source record/segment is split into two or more

destination segments/records, and the destination segments/records are on different lines of descent.

- *Splitting into a Parent and Children* on page 16-7 describes the case where the first destination segment/record is the parent of all the other destination segments/records that share the same context.

Splitting Into Independent Segments/Records

In the outgoing direction, a source record may be split into multiple destination segments, where the destination segments are on different lines of descent. This includes the case where the destination segments are siblings.

Likewise, in the incoming direction, a source segment may be split into multiple destination records, where the destination records are on different lines of descent. This includes the case where the destination records are siblings.

An instance of this is where the data for two or more destination segments/records comes from a single source record/segment. For example each ADDL_INFO record might provide the data for both the LIN segment and the PAC segment. LIN and PAC are siblings under the PO1 segment in an EDIFACT INVOICE. Because all LIN segments are generated before the first PAC segment, it is necessary to begin again with the first ADDL_INFO record in order to generate the PAC segments.

The second and succeeding Set Context lines which reference the same source record/segment must use the explicit instance 1 notation, record/segment B{1}, to cause the pointers to be reset to the first instance. This makes the Mapping Table Editor examine all the records/segments.

Set Context	Source record/segment B{1}
Repeat pattern	Blank
Map part set default	May be used

Splitting into a Parent and Children

In the outgoing direction, a source record may be split into multiple contiguous destination segments, where the first destination segment is the parent of the other destination segments.

Likewise, in the incoming direction, a source segment may be split into multiple contiguous destination records, where the first destination record is the parent of the other destination records.

An instance of this is where the data for a several adjacent destination segments/records (for example the N1, N3 and N4 segments) comes from a single source record/segment (for example an ADDRESS record). The syntax that may be used for this is:

First Map

Set Context	Source record/segment B. In the case where record/segment B is referenced in the Set Context of another map (as in *Splitting Into Independent Segments/Records* on page 16-7, record/segment B{1} must be used in the Set Context line.
Repeat pattern	Blank
Map part set default	May be used

Additional Maps

Set Context	Source krs B
Repeat pattern	No change

Combining

In the outgoing direction, multiple source records may be combined to form one destination segment.

Likewise, in the incoming direction, multiple source segments may be combined to form one destination record.

An instance of this is where the data for a destination segment/record must be gathered from several source records/segments. For example the ITD

segment might be constructed from data from both the ORDER_HEADER record and the TERMS record.

Set Context	Source record/segment B
Repeat pattern	Blank
Map part set default	Source record/segment C

Additional parts, if needed

Map part set default	Source record/segment D

Skipping a Level

Sometimes there is a situation where a source record/segment B is not needed, but there is a requirement to generate a destination segment/record, for each of the children C of each instance of segment/record B.

An example of this case is where there is a record between the ORDER_HEADER and LINE_ITEM records. This could be a PLANT record for the PLANT that wants the part, and where the PLANT record is not used in the output, but each LINE_ITEM record under each PLANT record is used to form a PO1 segment.

The syntax for this is:

Set Context	Source record/segment C. If segment/record C is referenced in the Set Context line for any map on a different line of descent which precedes the map with the FOR EACH, the record/segment C{1} notation must be used.
Repeat pattern	FOR EACH source record/segment B.
Map part set default	May be used.

The FOR EACH clause may only be used within the following restrictions. Only one FOR EACH clause may be used in the creation of any destination line of descent.

The record/segment referenced in the FOR EACH clause (that is, B) may not be referenced in the Set Context line for any other map on the destination line of descent.

Avoiding Cross-Over Patterns

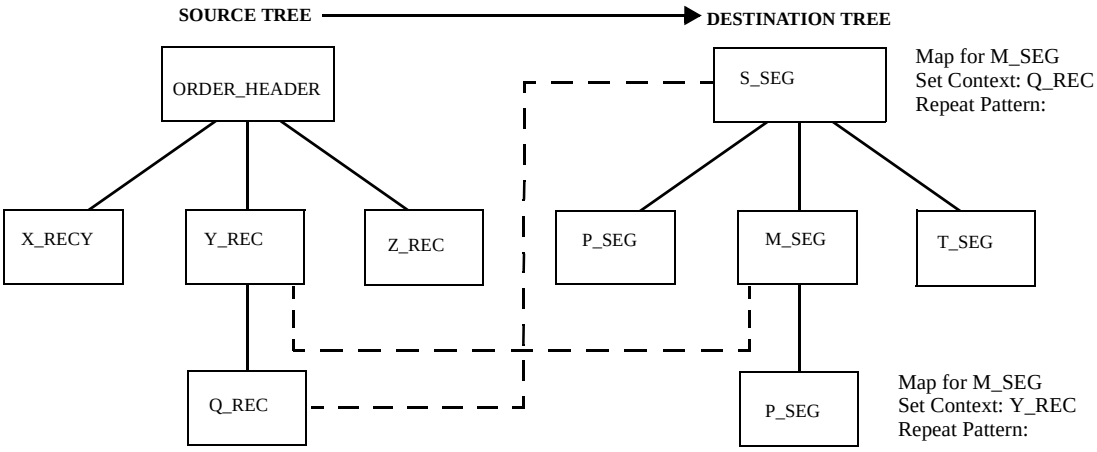
In the outgoing direction, if a map for a destination segment S_SEG references a source record Y_REC, none of the maps of the descendants of S_SEG may reference source record Y_REC or any of its ancestors, unless the NO CHANGE Repeat Pattern is used by the descendant's maps.

In other words, within a destination line of descent, the references to a line of descent in the source tree can not cross. Likewise for the incoming direction.

The Mapping Table Editor maintains a single set of pointers defining the current instance for each record/segment in the source tree. When maps are executed these pointers are changed when the Repeat Pattern line is left blank or when the FOR EACH Repeat Pattern is used. The NO CHANGE Repeat Pattern causes the pointers to be left unchanged.

Cross-overs without NO CHANGE are restricted because they cause the child's map to change the pointers that its parent map is also changing. This may cause unpredictable results or infinite loops.

Example of a Crossover - Not Supported



Set Context Line Must Be Used

The Set Context line may not be left blank.

Use of Explicit Qualification and Explicit Instances

When specifying record names on the Set Context line and the Repeat Pattern FOR EACH line, the record name may be specified as record or record{1}. Likewise for segments.

In other words, on a Set Context line only the name of the segment or record should appear, possibly with {1} after it. Constructs such as S_SEG.M_SEG are not supported. Similarly, on a FOR EACH Repeat Pattern, only the name of the segment or record should appear.

Mapping Assignments

In mapping assignments, explicit qualification and explicit instances should only be used where necessary.

Default Qualification

The Set Context and Set Default lines set the default qualification. As long as the default record/segment and current instance are to be used, no qualification (segment or record names) or instances is needed or desirable.

Explicit Qualification

If it is necessary to reference a different record/segment than the default, all names up to level 01 must be specified in the case of record names, or all names up to level 02 in the case of segment names.

For example if the current default is the PO1 segment, and it is necessary to reference the ITD segment, the reference would be specified as BEG.ITD:data_label.

If the current default record/segment is the LINE_ITEM record and it is necessary to reference the TERMS record, the reference would be specified as ORDER_HEADER.TERMS:field_name.

Explicit Instance

If it is necessary to specify an instance number, include the instance number in curly brackets and include the full qualification. For example to refer to the jth ADDL_INFO record under the kth LINE_ITEM under the current ORDER_HEADER, specify ORDER_HEADER.LINE_ITEM{k}.ADDL_INFO{j}:field_name. Where an instance number is not given the current instance number will be used.

Children Are Created After Parents

When the Mapping Table Editor creates destination segments/records, maps for children are only executed when the parent map has succeeded in producing a destination segment/record. In other words, it is not possible to create a child segment/record if the parent has not been created.

For example, if there were no PO1 segments created, the Mapping Table Editor would never try to create a PO3 segment.

Unused Records/Segments Permitted

It is not required to use all source records/segments or all destination segments/records.

For example, it is often the case that only a small subset of the possible destination segments are used. Also, there will sometimes be source records that are not needed for the document sent to a particular trading partner.

Chapter 17 Using Hooks to Customize the Mapper

This chapter describes how customize the Mapper by attaching user-written routines to either the predefined hooks provided in the Mapper, or hooks that you may define.

Customization Routines

When you customize the Mapper, you are extending it to include additional features not provided in the basic software. To customize the Mapper, you code a routine in the computer language of your choice that accepts data from the Mapper and returns a status and a value. A customization routine can be called at any point where a mapping expression can be evaluated. You declare the routine to the Mapper so that it knows what to look for. Then, at specific hook points in its processing, the Mapper calls the routine.

The values passed to the customization routine are determined by the values passed in a mapping expression or as part of one of the hook points. The customization routine cannot call back into the Mapper or see any of its global symbols.

Of the arguments passed to the customization routine, only one can be the output. All others are for input only. The output value is returned as the value of the function in the mapping expression. The return status of the customization routine determines whether or not to use the returned value. If you have more than one value to return, and these values can be computed before the mappings are executed, use the RECORD and SWITCH hook points, described in a later section of this chapter, to get the data into the input stream. This puts the data into the source tree where it can be accessible by the mappings.

The routine's return status must be one of the following:

- SUCCESS

  Use the return value.
- UNDEFINED

  Ignore, do not use the return value. This returns the special value $UNDEFINED.
- Anything else

  A Tru64 UNIX or OpenVMS error condition indicating a termination.

You declare a customization routine by specifying to the Mapper the characteristics of the routine and its calling arguments. You must specify:

- Function name.
- Shared image name or logical giving the name of the file containing the function.
- Return code for SUCCESS.
- Return code for UNDEFINED.
- Index of return value argument.
- Flag specifying Declare-Audit-Event-When-Called.
- Flag specifying Record-History-of-all-calls.
- Data type of each argument.
- Flag indicating whether the argument is optional.
- Type of argument passing mechanism for each argument, that is, pass-by-value, pass-by-reference, pass-by-descriptor.

To be called by the Mapper, your routine must be compiled and installed as a shareable object library. The Mapper performs a dynamic link at runtime.

The Mapper runs as a server process, so any shareable object libraries that are referenced within Mapping Tables must be accessible from the Mapper process.

To ensure this, it is advisable to place all hook shareable images in `/usr/bin`.

For a local Application Client, the Mapper runs under the account that issued the POST or FETCH request.

For a remote Application Client, the Mapper runs under the Digital DEC/EDI account.

Note that a customization routine runs in the Digital DEC/EDI Server environment, not in the user's process environment.

Creating and Customizing Hooks

You use the Mapping Table Editor to create, edit and delete hooks. It allows you to declare a custom subroutine to the Mapper. It also allows you to declare a subroutine for any (or all) of the predefined hook points.

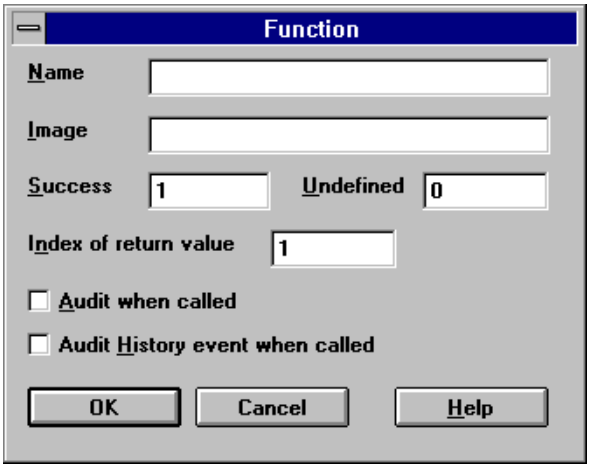
To begin, select the **Hooks** option from either the **Map Navigator** or the **Mapping** menu, when you are editing a Mapping Table. The **Hooks** screen is then displayed, and the **Hooks** menu is added to the Menu Bar.

In creating a customized hook point, you have to declare a function and one or more arguments for that function. You have to create the function before you can create its arguments. The order in which you attach arguments to a function dictates their order of precedence.

Function and **Argument** are the two options on the **Hooks | Create** sub-menu. Each option provides a pop-up screen dialog.

The **Hooks Function** screen is shown in Figure 17-1 *The Hooks Function Screen*.

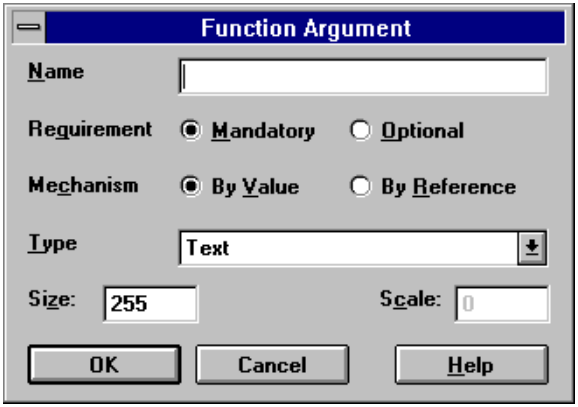
Figure 17-1 The Hooks Function Screen



For detailed information about what you need to enter in each field on the screen, refer to the following table or read the online help. The online help displays a copy of the original dialog screen. You obtain help by pointing the mouse arrow at the field about which you want the information, and clicking once.

The **Hooks Function Argument** screen is shown in Figure 17-2 *The Hooks Function Argument Screen*.

Figure 17-2 The Hooks Function Argument Screen



For detailed information about what you need to enter in each field, refer to the following table or read the online help.

Function and Argument Field Entries

The possible field entries and choices on the **Function** and **Function Argument** dialog screens are described in the following table.

Table 17-1 Function and Argument Field Entries

Field	Type	Description
Name	Function	This is the name of the customization routine, as it will be called within the Mapper and of the function that will be used in mapping expressions. This name must also correspond to the symbol defined for the routine's entry point within the shared image file. After you enter the name, any other details, and select **OK**, you are returned to the **Function** screen.
Image	Function	This is the file name of the shared image that contains the routine. A shared image can contain more than one customization routine. If you omit the device and directory specification from the location of the shareable object library, it is assumed to reside in `/usr/bin`, and have file extension of `.so`. On OpenVMS, This is the file name of the shareable image that contains the routine. A shareable image can contain more than one customization routine. If you omit the device and directory specification from the location of the shareable image, then it is assumed to reside in SYS$SHARE, and have file extension of .EXE If you update a hook shareable image file, you must use the INSTALL command REPLACE to ensure that the Mapper uses the new version of the image.

Table 17-1 Function and Argument Field Entries

Field	Type	Description
Success	Function	This is the longword value that will be returned by the routine to indicate successful execution. A value of 1 is suggested. Successful execution means that the returned value is to be used. Any return value besides SUCCESS or UNDEFINED will be an error code indicating a termination.
Undefined	Function	This is the longword value that will be returned by the routine to indicate the result of execution is undefined. A value of 0 is suggested. An undefined result means that the returned value is to be ignored. This returns the special value $UNDEFINED. Any return value besides SUCCESS or UNDEFINED will be an error code indicating a termination.
Index of return value	Function	This is the number for the argument to be used as the return value. The first argument is number 1. Using the first argument is suggested. When using the function in a mapping expression or at a hook point, this argument will not be included. For example, if the customization routine ABC contains four arguments and argument 1 is identified as the return value, then when using the function in a mapping expression or a hook point, give only three values: `ABC(1,2,3)` The value 1 is passed in argument 2, the value 2 is passed in argument 3, and the value 3 is passed in argument 4. The value of the function is taken from the value returned in argument 1.

Table 17-1 Function and Argument Field Entries

Field	Type	Description
Audit when called	Function	Enter an X in the box to set this flag. If this flag is set, each time the function is called the Mapper makes an entry in the Mapper audit log.
Audit history event when called	Function	Enter an X in the box to set this flag. If set, the Mapper records in a history file the values of all calling arguments and the return value and status each time the function is called. The enable history flag must also be set before any recording can take place.
Name	Argument	Refer to the online help for examples.
Requirement	Argument	Use this flag to indicate whether this argument is mandatory or optional. Optional arguments do not have to be provided with values when used — mandatory arguments do. The Mapper passes a zero by value for all unused optional arguments. Any argument following an optional argument must also be optional.

Table 17-1 Function and Argument Field Entries

Field	Type	Description
Mechanism	Argument	Use this flag to indicate whether the argument is to be passed by value, or by reference. By value, the entire value is copied into the argument list prior to the call (data type must be 4 bytes or less). The pass-by-value method cannot be used to return a value from the function. This item is mutually exclusive with pass-by-reference and pass-by-descriptor flags. By reference, the argument is passed by putting a pointer to the value (actually, a copy of it) in the argument list. The called function will get a pointer to the value. For input arguments — those passed to the customization routine, the Mapper passes a pointer to a copy of the value specified in the mapping or hook expression. The value has been converted to match the data type. If the argument is a return value, the Mapper passes a pointer to a buffer of the type specified. The function fills in the buffer and the Mapper uses it as the return value of the function. This is mutually exclusive with pass-by-value and pass-by-descriptor flags.
Type	Argument	This is the data type of the value to be passed on each argument. The acceptable types are the same as the data type attributes used to define fields on records. Refer to the online help for more information.
Size	Argument	Enter the size of the argument field.
Scale	Argument	Enter the size of the scaling factor to use with a numeric field. Note: This field is disabled for textual types.

The customization routines must conform to the Tru64 UNIX calling standards. For more information regarding data types, descriptors, and calling standards, refer to your Tru64 UNIX documentation.

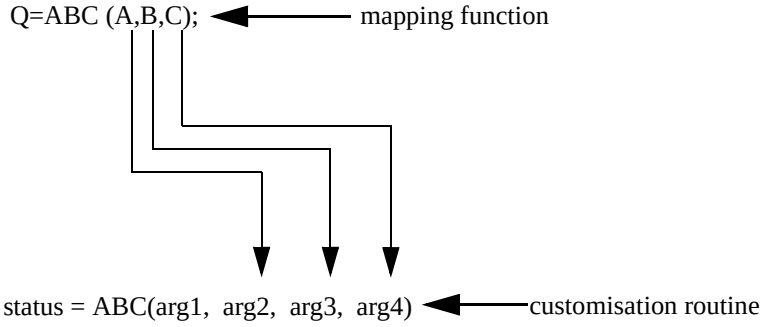
To use the customization facilities, you write a custom coded subroutine. You can write it in any programming language that accepts data from the Mapper and returns a status and a value. The language must use the Tru64 UNIX or OpenVMS calling standard, as defined by the Digital DEC/EDI Server platform. The values passed to the custom coded subroutine are determined by the values passed in a mapping expression or else are passed as part of one of the predefined hook points.

For example, suppose you have defined a function name ABC in the **Function** screen. Also, suppose that the function ABC is used in the following mapping expression or maybe at a hook point:

```
Q=ABC(A,B,C);
```

When the Mapper encounters this assignment, it checks the declaration, converts the parameters to the appropriate data type, and then passes them using the appropriate “pass-by” mechanism to the customization routine. The customization routine processes them. If it is successful, (status=SUCCESS) then the value of the argument specified as the return value argument is returned in place of the function in the expression. If it is unsuccessful, (status=UNDEFINED) then the $UNDEFINED special constant is returned in place of the function in the expression. The following diagram illustrates this concept.

Figure 17-3



If status=SUCCESS then arg1 is returned as the value of the mapping function.

If status=UNDEFINED then the $UNDEFINED special constant is returned as the value of the mapping function.

The following procedures describe how to compile and link your customization routine so that the Mapper can find it.

1. Write the customization function as you would any other subroutine using any language. The function name must be global so that the linker can find it. You can have several functions in the Module.

   The function name you specify in the hook declaration must be identical to the function name declared in the hook shareable image. This includes case sensitivity.

   The hook may be written in any conventional programming language. However, some languages may modify the names of externally visible functions, so you must refer to the hook routine by its external name.

   The following can be used as a sample C customization routine:

```
/* HELLO.C - Example Customization Routine in C */

#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include "ssdef.h"

#define OUTPUT_FILE_SPEC "/var/adm/decedi/temp/hello.txt"
#define ARG_SIZE 80
```

```
/*****************************************
 * Announce that the hook was called, pass the value back.
 *    HELLO(text)
 *      1 return_val  Pass by reference  TEXT 80;
 *      2 text        Pass by reference  TEXT 80;
 *****************************************/
int hello(char *return_val, char *text)
{
    int stat;
    int len;
    char *ptr;
    FILE *fout;

    /* trim trailing spaces off text and print it */
    len = ARG_SIZE-1;
    ptr = text;
    while (len>=0 && isspace(ptr[len])) len--;
    len++;
    /*
    **  Determine if file spec was given, if so then this is
    **  a new preprocess event, so open a new output file.
    */
    if ((strchr (ptr, '.') != (char *)NULL) &&
        (strchr (ptr, '/') != (char *)NULL))
    {
        fout = fopen(OUTPUT_FILE_SPEC, "w");
    }
    else
    {
        fout = fopen(OUTPUT_FILE_SPEC, "a");
    }

    if (fout != (FILE *)NULL)
    {
        fprintf(fout,"hello: \"%.*s\"\n",len,ptr);
    }
    else
    {
        printf ("hello: \"%.*s\"\n",len,ptr);
    }
    /* Send back what we got */
    sprintf (return_val, "%.*s",len,ptr);

    if (fout != (FILE *)NULL)
    {
        fclose (fout);
    }
```

```
    stat = 1; /* success */
    return(stat);
}
```

2. On OpenVMS, Global variables are shared between functions called by a single invocation of the Mapper, but must not be shared between other Mapper processes.To make sure that global variables are not shared, be sure to declare your variables as non-shareable (NOSHARE in C) in your customization routine. If your language has no facilities for doing this, you can force it in the linker with options that can be added to the option file as follows:

   ```
   PSECT_ATTR=<psect_name>,NOSHR,LCL
   ```

3. Compile the module containing the customization routines.
4. Link the module as a shared image as follows:

Tru64 UNIX

Enter the following UNIX commands to create shared object library of the above routine and copy the output ".so" file to where the Mapper expects to find it in `/usr/bin`:

```
prompt> cc -c -o hello.o hello.c  # Compilation
prompt> ar -r hello.a hello.o     # Create library
prompt> ld -shared -o hello.so \
    -all hello.a -none -lc        # link object (shared)
prompt> cp hello.so /usr/bin      # copy to where mapper
                                  # expects to find it.
```

OpenVMS

Link the module as a shared image and specify the function names that are to be accessed by the Mapper as UNIVERSAL. The UNIVERSAL option is specified in a separate options file. For example, suppose the module ABC.C has two functions, ABC1 and ABC2. The LINK command would be as follows:

```
$ LINK/SHARE ABC,ABC.OPT/OPTION
```

The contents of the ABC.OPT file would be as follows, in an OpenVMS Alpha environment:

```
SYMBOL_VECTOR=(ABC1=PROCEDURE,ABC2=PROCEDURE)
```

In the Customization Routine Declaration, specify the full file specification; if you do not then the Mapper will look for the hook shareable image in SYS$SHARE.

Enter the following OpenVMS commands to create a shareable image of the above routine and copy the output ".EXE" file to where the Mapper expects to find it in SYS$SHARE:

The shareable image may be installed to improve activation performance, however this is not a requirement. If the image is to be installed, you should LINK it specifying /NOTRACEBACK, and INSTALL it as a privileged image by specifying /PRIVILEGED. The image file must be accessible to the Digital DEC/EDI Server in order to dynamically link the image at run-time.

Accessing SQL Databases from Hooks on OpenVMS

When compiling hook routines that access SQL databases, you must use the `/NOCONNECT` option on the compilation command.

You also need to specify the following PSECTs in the link options file, as well as PSECTs for any SQL aliases that are declared in the SQL program.

```
PSECT_ATTR=RDB$DBHANDLE,NOSHR,LCL
PSECT_ATTR=RDB$MESSAGE_VECTOR,NOSHR,LCL
PSECT_ATTR=RDB$TRANSACTION_HANDLE,NOSHR,LCL
PSECT_ATTR=SQLCA,NOSHR,LCL
PSECT_ATTR=SQLDA,NOSHR,LCL
```

Declaring Routines at Predefined Hook Locations

A customization routine can be called at any point where a mapping expression can be evaluated. In addition, a customization routine can be called at any of the predefined hook locations within Mapper processing.

The following is a list of the predefined hook locations:

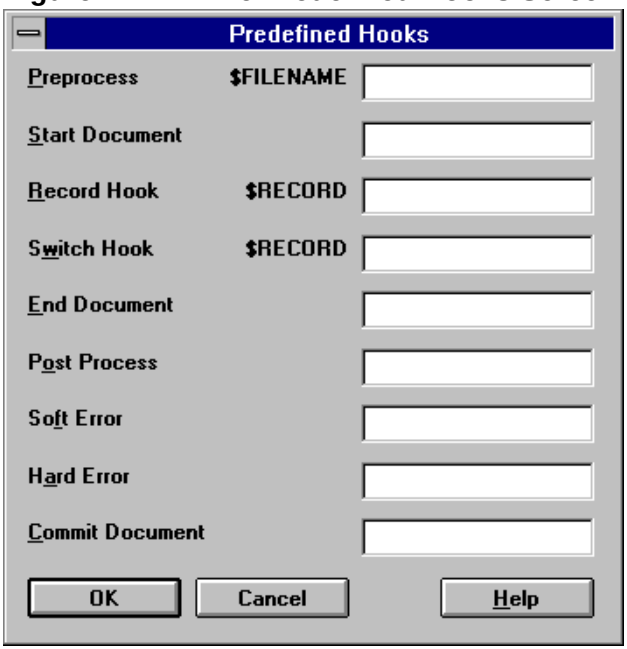
- Preprocess
- Start Document
- Record Hook
- Switch Hook
- End Document
- Post Process
- Soft Error
- Hard Error
- Commit Document

These hook points are described in detail later in this section.

You declare a hook subroutine at any (or several) of the predefined hook points by using the **Predefined Hook**s dialog screen.

The **Predefined Hooks** screen is shown in Figure 17-4 *The Predefined Hooks Screen*.

Figure 17-4 The Predefined Hooks Screen



You access the screen by selecting the **Predefined...** option on the **Hooks** menu. The menu is added to the Mapping Table Editor Menu Bar when you select the **Hooks** option from either the **Map Navigator** or the **Mapping** menu.

- **Preprocess**

  The customization routine associated with this hook point is called just before the Mapper opens the application data file for an Outgoing document or creates the output application file for an Incoming document.

  Just prior to this call, the Mapper places the application file name given in the command, into the global variable $FILENAME. You can pass this global variable to the customization routine at this hook point.

  This feature allows you to screen the file names and make substitutions. In the Outgoing direction, it allows the custom coded subroutine assigned to this hook to open the file, preprocess it, close it, and then allow the Mapper to process it.

  In the Incoming direction, the file name can be converted into a unique file name that matches the conventions of the application by the customization routine.

  The returned value is used by the Mapper to open or create application files. In the Outgoing direction, the returned file name can contain wildcards, which tells the Mapper to open and read all of them. After a file is opened or created, its full name with extension is placed in the global variable $FILENAME.

  If the return status is $UNDEFINED, or there is no customization routine specified, the Mapper uses the value currently in $FILENAME.

- **Start Document**

  For an Incoming document, the customization routine is called just after the start of a document, just before the first output record is generated.

  For an Outgoing document, the routine is called as soon as the Mapper detects the beginning of a new document in the record stream. This happens after the first record of the document has been read and processed, but before the rest of the document has been read.

  The return status is ignored unless it is an error. There is no return value.

  The START_DOCUMENT HOOK is only called if the BEGINS DOC indicator is set.

- **Record**

  At this hook point, the customization routine is called whenever the Mapper has read a logical record from the input stream (Outgoing direction), or is about to write one (Incoming direction).

  Custom coded routines assigned to this hook can be used to reformat or preprocess a record before it is passed to the Mapper for processing. If the pre-processing results in the record being broken into several records, the routine can assign the new records to a queue maintained by a customization routine assigned to the SWITCH hook and then return the first new record.

  Just before calling the customization routine at this hook point, the Mapper fills in the global variable $RECORD with the record just read from the Application File (Outgoing direction) or the record just obtained from the mappings (Incoming direction).

  If the return status is SUCCESS, it means the Mapper can use the returned value for the record. If the return status is UNDEFINED, it means that the Mapper should not process the record passed in. That is, in the Outgoing direction it should ignore the record and read another one; in the Incoming direction it should discard this record and not write it to the output file.

  A data type of VARYING_STRING is recommended for passing $RECORD, because the original record length is maintained. Define the VARYING_STRING to be a size long enough to hold the largest possible record that might be returned from the function.

- **Switch**

  The customization routine at this hook point is called by the Mapper after it has processed a record and before getting another one to process.

  The routine assigned to the SWITCH hook can return a record to be inserted in the record stream at this point by returning a value and a status of SUCCESS. One implementation of such a routine would be to operate a record queue that is used by other customization routines to provide records for processing.

  If processing is for the Outgoing direction, the Mapper continues calling this hook to get new records, rather than reading the Application Data File. As long as the routine assigned to the SWITCH hook has records to process, it should return a status of SUCCESS. When it has no more

records, it should return a status of UNDEFINED. The Mapper would then resume reading from the input file to get the next record.

If the processing is for Incoming direction, the Mapper calls the customization routine on the SWITCH hook point just prior to obtaining the next record from the mapping. If it gets a record from this hook point, it writes this record to the application file instead of obtaining one from the mappings.

The Mapper calls the SWITCH hook to get records to write to the output Application File that might have been queued to it by other customization routines.

The Mapper should continue getting records from this routine and writing them to the application file as long as the routine returns SUCCESS. If it returns UNDEFINED, then the Mapper resumes getting the next record from the mapping.

The SWITCH hook can be used by RECORD hook to break up a record into a series of smaller records.

The Mapper assumes the return value is a record to be processed if the status is SUCCESS.

A data type of VARYING_STRING is recommended for passing $RECORD, because the original record length is maintained.

- **End Document**

  This hook is called when the Mapper detects the end of a document. Customization routines assigned to this hook can perform document level processing. The entire document data cannot be accessed at this point, unless it was captured by the RECORD hook.

  Among the predefined global variables are the following that can be used as values passed into the routine:

- $APPLICATION
- $DOCTYPE
- $PARTNER
- $USERREF
- $TESTIND
- $FILENAME

The return status is ignored unless it is an error. Success means a Digital DEC/EDI End_Send or End_Fetch is to be performed. Undefined means abandon the document with an ABORT status to Digital DEC/EDI.

- **Post Process**

  At this hook point, the customization routine is called just after the Mapper closes the application data file and just before it terminates.

  In the Incoming direction, the routine assigned to this hook can open the application file just created and perform any post processing that might be needed. Remember that if mapping statements do not generate any Incoming documents, the file will not be created.

  In the Outgoing direction, the routine can signal the application that the contents of the file have been processed.

  The return status is ignored unless it is an error. No return value is expected.

- **Soft Error**

  At this hook point, the customization routine is called by the Mapper when it wants to abandon a single document (with an ABORT status to Digital DEC/EDI) and issue a warning message, but can continue processing the remaining documents. Using the return status values, this routine can turn the soft error into a termination or direct the Mapper to ignore the error.

  Prior to calling the customization routine at the SOFT ERROR hook point, the Mapper fills in the global variable $ERROR_CODE with the Tru64 UNIX error code. This can be passed to the customization routine to identify the type of error.

  If the return status is SUCCESS the processing continues with the next document. If the return status is UNDEFINED or an error, the processing stops and it is converted to a hard error.

- **Hard Error**

  At this hook point, the customization routine is called just prior to termination. It passes the error code in its first input argument.

  Prior to calling the customization routine at the HARD_ERROR hook point, the Mapper fills in the global variable, $ERROR_CODE, with the Tru64 UNIX error code. This can be passed to the customization routine to identify the type of error.

  The return status is ignored. Any outstanding documents are abandoned with a QUIT status to Digital DEC/EDI so that the outstanding documents can be retrieved in the next session.

- **Commit Document**

  At this hook point, the customization routine is called just after the Mapper has performed the END SEND and Digital DEC/EDI has returned the document identifier, called the DOCCOUNT. For example, if the application id is ORDERS, the unique document id assigned by Digital DEC/EDI will have the form ORDERS_O_xxxxx where xxxxx is the DOCCOUNT.

  The Mapper processing with Digital DEC/EDI proceeds as follows.

  – The Mapper performs a START SEND. An audit event is recorded in the audit database.

  – The Mapper then performs a series of START GROUP, END GROUP and PUT DATA calls.

  – The Mapper then performs an END SEND. An audit event is recorded in the audit database.

  When Digital DEC/EDI returns, it returns the unique identifying status code DOCCOUNT. The Mapper assigns the value to the global variable $DOCCOUNT.

  The Mapper records the ‘commit document’ event in the audit database, including the DOCCOUNT.

  If the COMMIT DOCUMENT hook is defined, the hook routine defined at the commit document point is called.

Using a hook routine at this point, the unique document id that has been returned by Digital DEC/EDI can be written out by using $DOCCOUNT as one of the calling arguments. This information may then be used by the caller if the Mapper was invoked through the callable interface or for user-written reporting.

Hook Example

The following is an example of how a customization routine can be called at a hook point. This example contains the following:

- Parts of a Mapper compilation listing showing the customization routine declaration and its use in mappings and hook points.
- Sample C customization routine for the hello function.
- Instructions on how to compile and link for this example.
- Comments.

Excerpts from Compile Listing

```
  ... declarations ...
------------------------------------------------------------
INITIALIZATION:

      Variable   Expression
 dummy = hello("initialization");    <=== Used the same way in
a map.
------------------------------------------------------------
  ... other declarations ...
------------------------------------------------------------
CUSTOMIZATION ROUTINE DECLARATION:

  Function Name:  hello
  Image Filename: hello.so
  Return Status Value for SUCCESS:   1    <=== This is where we
define SUCCESS
  Return Status Value for UNDEFINED: 0
  Index of Return Value Argument:    1    <=== first argument
is return value
  Declare Audit Event When Called:  [ ]
  Record History of all Calls:      [ ]
  Arguments:
      name                                  Data Type
  1 return_val           BY REFERENCE TEXT;      <=== Out
  2 text                 BY REFERENCE TEXT;      <=== In

------------------------------------------------------------

HOOK ASSIGNMENTS:

  Built-in Hook Location                     Function
    PREPROCESS            $FILENAME = hello($FILENAME);
```

```
    START DOCUMENT                    = hello("start document
hook");

    RECORD                    $RECORD =

    SWITCH                    $RECORD =

    END DOCUMENT                      = hello("End Document
Hook");

    POST PROCESS                      = hello("Post Process
hook");

    SOFT ERROR                       = hello("Soft Error hook");


    HARD ERROR                        = hello("Hard Error hook");
```

Sample C Customization Routine

Tru64 UNIX

The following is the 'C' source code for the above example, for the Tru64 UNIX environment:

```
/* HELLO.C - Example Customization Routine in C */

#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include "ssdef.h"

#define OUTPUT_FILE_SPEC "/var/adm/decedi/temp/hello.txt"
#define ARG_SIZE 80

/*****************************************
 * Announce that the hook was called, pass the value back.
 *    HELLO(text)
 *      1 return_val  Pass by reference  TEXT 80;
 *      2 text        Pass by reference  TEXT 80;
 *****************************************/

int hello(char *return_val, char *text)
{
    int stat;
    int len;
    char *ptr;
    FILE *fout;
```

```
    /* trim trailing spaces off text and print it */
    len = ARG_SIZE-1;
    ptr = text;
    while (len>=0 && isspace(ptr[len])) len--;
    len++;

    /*
    **  Determine if file spec was given, if so then this is
    **  a new preprocess event, so open a new output file.
    */
    if ((strchr (ptr, '.') != (char *)NULL) &&
        (strchr (ptr, '/') != (char *)NULL))
    {
        fout = fopen(OUTPUT_FILE_SPEC, "w");
    }
    else
    {
        fout = fopen(OUTPUT_FILE_SPEC, "a");
    }

    if (fout != (FILE *)NULL)
    {
        fprintf(fout,"hello: \"%.*s\"\n",len,ptr);
    }
    else
    {
        printf ("hello: \"%.*s\"\n",len,ptr);
    }

    /* Send back what we got */
    sprintf (return_val, "%.*s",len,ptr);

    if (fout != (FILE *)NULL)
    {
        fclose (fout);
    }

    stat = 1; /* success */
    return(stat);
}
```

OpenVMS

The following is the ‘C’ source code for the above example, for the OpenVMS environment:

```
/* HELLO.C - Example Customization Routine in 'C' */
#include <ctype.h>
#include <descrip.h>
#include <stdio.h>
```

```
#include <ssdef.h>
#include <string.h>
#define OUTPUT_FILE_SPEC "decedi$temp:hello.txt"
typedef struct dsc$descriptor desc; /* short typedef for
descriptor */
/*****************************************
* Say that the hook was called, pass the value back
* HELLO(text)
* 1 return_val Pass by descriptor TEXT 80;
* 2 my_text Pass by descriptor TEXT 80;
*****************************************/
int hello(desc *return_val, desc *my_text)
{
  int stat;
  int len;
  char *ptr;
  FILE *fout;
  /*
  ** Trim trailing spaces off text and print it
  */
  len = my_text->dsc$w_length-1;
  ptr = my_text->dsc$a_pointer;
  while (len>=0 && isspace(ptr[len])) len--;
  len++;
  /*
  ** Determine if file spec was given, if so then this is
  ** a new preprocess event, so open a new output file.
  */
  if ((strchr (ptr, '.') != (char *)NULL) &&
      (strchr (ptr, '[') != (char *)NULL) &&
      (strchr (ptr, ']') != (char *)NULL))
  {
    fout = fopen(OUTPUT_FILE_SPEC, "w");
  }
  else
  {
    fout = fopen(OUTPUT_FILE_SPEC, "a");
  }
  if (fout != NULL)
  {
    fprintf(fout,"hello: \"%.*s\"\n",len,ptr);
    /*
    ** Send back what we sent it
    */
    stat = lib$scopy_dxdx (my_text, return_val);
    if (stat != SS$_NORMAL)
    {
      fprintf (fout, "status from lib$scopy_dxdx = %d\n",
```

```
        stat);
      stat = 0; /* undefined */
    }
    else
    {
      stat = 1; /* success */
    }
    fclose (fout);
  }
  else
  {
    stat = 0; /* undefined */
  }
  return(stat);
}
```

Linking Instructions

Tru64 UNIX The following commands were used to link the above example, in a Tru64 UNIX environment:

```
prompt> cc -c -o hello.o hello.c  # Compilation
prompt> ar -r hello.a hello.o     # Create library
prompt> ld -shared -o hello.so \
prompt>    -all hello.a -none -lc # link object (shared)
prompt> cp hello.so /usr/bin      # copy to where mapper
prompt>                           # expects to find it.
```

OpenVMS The following commands are used to link the above example, in an OpenVMS Alpha environment:

```
$ CC hello.c
$ LINK/SHARE HELLO,SYS$INPUT/OPT
SYMBOL_VECTOR=(HELLO=PROCEDURE)
```

Comments

1. The $FILENAME global will be loaded up by the Mapper just before the PREPROCESS Hookpoint. It will have the name provided in the MAPPER command line.
2. The $RECORD will also be loaded up by the Mapper just prior to the RECORD hook call if you gave a routine to call at that point. It is best to define the argument type VARYING_STRING so that the record length will be maintained. If you defined it as TEXT, the record will be space padded to the maximum length.

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

Z