SeeBeyond ICAN Suite

X12 OTD Library User's Guide

Release 5.0.3

SEEBEYOND®

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter introduces you to the X12 OTD Library User's Guide.

1.1 Overview

Each of the eGate Object Type Definition (OTD) libraries contains sets of pre-built structures for industry-standard formats. The ASC X12 OTD Library is one of the products within the SeeBeyond ICAN Suite. The OTD library contains message definitions for X12 messages.

This document gives a brief overview of X12 and the X12 message structures provided, and provides information on installing and using the ASC X12 OTD Library.

1.2 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the SeeBeyond ICAN Suite (such as eGate Integrator and eXchange Integrator), to have familiarity with Windows operations and administration, and to be thoroughly familiar with Microsoft Windows graphical user interfaces.

1.3 Compatible Systems

The ASC X12 OTD Library is available on the following platforms:

- Microsoft Windows 2000, Windows XP, and Windows 2003
- Sun Solaris 8 and Solaris 9
- IBM AIX 5L Version 5.1 and AIX 5L Version 5.2
- HP-UX 11.0 and HP-UX 11i (PA-RISC)
- HP Tru64 UNIX Version 5.1A
- Red Hat Linux 8 (Intel *x*86) and Linux Advanced Server 2.1 (Intel *x*86)

1.4 Document Organization

This document is organized topically as follows:

- Chapter 1 **"Introduction"** gives a general preview of this document, its purpose, scope, and organization.
- Chapter 2 **"X12 Overview"** provides an overview of X12.
- Chapter 3 **"Installation"** explains how to install the X12 OTD Library files and where to find them after installation.
- Chapter 4 **"Working With the X12 OTDs"** provides instructions and examples on how to load, view, and test X12 OTDs.
- Chapter 5 **"Java Methods for X12 OTDs"** lists and explains the bean nodes and Java methods that can be used to extend the functionality of the OTDs in the library.

1.5 Writing Conventions

The following writing conventions are observed throughout this document.

Table 1 Writing Conventions

Text	Convention	Example
Names of buttons, files, menus and menu items, icons, parameters, variables, methods, and objects	**Bold** text	▪ Click **OK** to save and close. ▪ Select the **logicalhost.exe** file. ▪ On the **File** menu, click **Exit**. ▪ Enter the **timeout** value. ▪ Use the **getClassName()** method.
Command-line arguments, code samples	`Fixed` font. Variables are shown in ***bold italic***.	`bootstrap -f -p` ***`password`***
Hypertext links	**Blue** text	▪ For more information, see **"Writing Conventions" on page 9**. ▪ **http://www.seebeyond.com**

Additional Conventions

Windows Systems

For the purposes of this guide, references to "Windows" will apply to Microsoft Windows Server 2003, Windows XP, and Windows 2000.

Path Name Separator

This guide uses the backslash ("\") as the separator within path names. If you are working on a UNIX or HP NonStop system, please make the appropriate substitutions.

1.6 Supporting Documents

The following SeeBeyond documents provide additional information about eGate and the ICAN system:

- *SeeBeyond ICAN Suite Installation Guide*
- *SeeBeyond ICAN Suite Primer*
- *SeeBeyond ICAN Suite Deployment Guide*
- *eGate Integrator User's Guide*
- *eGate Integrator Tutorial*
- *eGate Integrator System Administration Guide*
- *eXchange Integrator User's Guide*
- *HIPAA OTD Library User's Guide*
- *UN/EDIFACT OTD Library User's Guide*

You can also refer to the appropriate Microsoft Windows or UNIX documents, if necessary.

1.7 SeeBeyond Web Site

The SeeBeyond Web site is your best source for up-to-the-minute product news and technical support information. The site's URL is:

http://www.seebeyond.com

Chapter 2

X12 Overview

This chapter provides the following information:

- An overview of X12, including the structure of an X12 envelope, data elements, and syntax.
- An explanation of how to use the generic message structures provided as an add-on to eGate to help you quickly create the structures you need for X12 transactions.
- An example of how X12 is used in payment processing.

2.1 Introduction to X12

The following sections provide an introduction to X12 and to the message structures that constitute the X12 OTD Library.

2.1.1. What Is X12?

X12 is an EDI (electronic data interchange) standard, developed for the electronic exchange of machine-readable information between businesses.

The Accredited Standards Committee (ASC) X12 was chartered by the American National Standards Institute (ANSI) in 1979 to develop uniform standards for interindustry electronic interchange of business transactions—electronic data interchange (EDI). The result was the X12 standard.

The X12 body develops, maintains, interprets, and promotes the proper use of the ASC standard. Data Interchange Standards Association (DISA) publishes the X12 standard and the UN/EDIFACT standard. The X12 body comes together three times a year to develop and maintain EDI standards. Its main objective is to develop standards to facilitate electronic interchange relating to business transactions such as order placement and processing, shipping and receiving information, invoicing, and payment information.

The X12 EDI standard is used for EDI within the United States. UN/EDIFACT is broadly used in Europe and other parts of the world.

X12 was originally intended to handle large batches of transactions. However, it has been extended to encompass real-time processing (transactions sent individually as they are ready to send, rather than held for batching) for some healthcare transactions to accommodate the healthcare industry.

2.1.2. What Is a Message Structure?

The term *message structure* (also called a transaction set structure) refers to the way in which data elements are organized and related to each other for a particular EDI transaction.

In eGate, a message structure is called an Object Type Definition (OTD). Each message structure (OTD) consists of the following:

- Physical hierarchy

  The predefined way in which envelopes, segments, and data elements are organized to describe a particular X12 EDI transaction.

- Delimiters

  The specific predefined characters that are used to mark the beginning and end of envelopes, segments, and data elements.

- Properties

  The characteristics of a data element, such as the length of each element, default values, and indicators that specify attributes of a data element—for example, whether it is required, optional, or repeating.

The transaction set structure of an invoice that is sent from one trading partner to another defines the header, trailer, segments, and data elements required by invoice transactions. The X12 OTD Library for a specific version includes transaction set structures for each of the transactions available in that version. You can use these structures as provided, or customize them to suit your business needs.

eGate Integrator uses Object Type Definitions based on the X12 message structures to verify that the data in the messages coming in or going out is in the correct format. There is a message structure for each X12 transaction.

The list of transactions provided is different for each version of X12. This book uses Versions 4010 and 4021 as examples for illustrating how to install and work with X12 OTDs.

2.2 Components of an X12 Envelope

X12 messages are all ASCII text, with the exception of the BIN segment which is binary.

Each X12 message is made up of a combination of the following elements:

- Data elements
- Segments
- Loops

Elements are separated by delimiters.

More information on each of these is provided below.

2.2.1. Data Elements

The data element is the smallest named unit of information in the X12 standard. Data elements can be broken down into two types. The distinction between the two is strictly a matter of how they are used. The two types are:

- Simple

  If a data element occurs in a segment outside the defined boundaries of a composite data structure it is called a simple data element.

- Composite

  If a data element occurs as an ordinally positioned member of a composite data structure it is called a composite data element.

Each data element has a unique reference number; it also has a name, description, data type, and minimum and maximum length.

2.2.2. Segments

A segment is a logical grouping of data elements. In X12, the same segment can be used for different purposes. This means that a field's meaning can change based on the segment. For example:

- The NM1 segment is for *any* name (patient, provider, organization, doctor)
- The DTP segment is for *any* date (date of birth, discharge date, coverage period)

For more information on the X12 enveloping segments, refer to **"Structure of an X12 Envelope" on page 14**.

2.2.3. Loops

Loops are sets of repeating ordered segments. In X12 you can locate elements by specifying:

- The transaction set (for example, 270)
- The loop (for example, "loop 1000" or "info. receiver loop")
- The occurrence of the loop
- The segment (for example, BGN)
- The field number (for example, 01)
- The occurrence of the segment (if it is a repeating segment)

2.2.4. Delimiters

In an X12 message, the various delimiters act as syntax, dividing up the different elements of a message. The delimiters used in the message are defined in the interchange control header, the outermost layer enveloping the message. For this reason, there is flexibility in the delimiters that are used.

No suggested delimiters are recommended as part of the X12 standards, but the industry-specific implementation guides do have recommended delimiters.

The default delimiters used by the SeeBeyond X12 OTD Library are the same as those recommended by the industry-specific implementation guides. These delimiters are shown in Table 2.

Table 2 Default Delimiters in X12 OTD Library

Type of Delimiter	Default Value
Segment terminator	~ (tilde)
Data element separator	* (asterisk)
Subelement (component) separator	: (colon)
Repetition separator (version 4020 and later)	+ (plus sign)

Within eXchange Integrator, delimiters are specified at the B2B protocol level. The delimiters you define are applied to all transaction types.

If you do not specify delimiters, eXchange expects the default delimiters as shown in Table 2.

Note: *It is important to note that errors could result if the transmitted data itself includes any of the characters that have been defined as delimiters. Specifically, the existence of asterisks within transmitted application data is a known issue in X12, and can cause problems with translation.*

2.3 Structure of an X12 Envelope

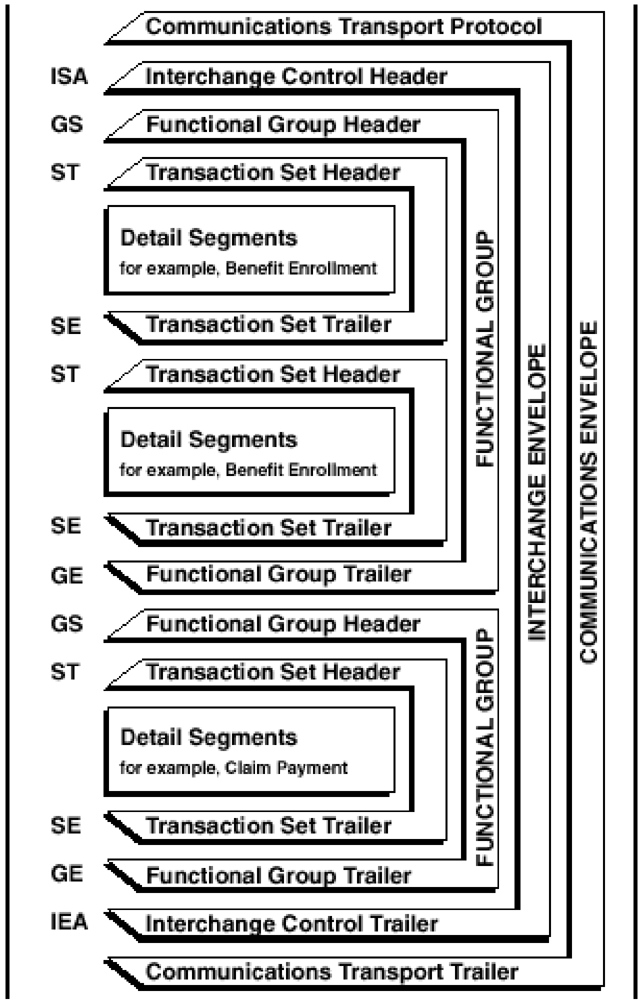
The rules applying to the structure of an X12 envelope are very strict, to ensure the integrity of the data and the efficiency of the information exchange.

The actual X12 message structure has three main levels. From the highest to the lowest they are:

- Interchange Envelope
- Functional Group
- Transaction Set

A schematic of X12 envelopes is shown in Figure 1. Each of these levels is explained in more detail in the following sections.

Figure 1 X12 Envelope Schematic



Note: *The above schematic is from Appendix B of an X12 Implementation Guide.*

Figure 2 shows the standard segment table for an X12 997 (Functional Acknowledgment) as it appears in the X12 standard and in most industry-specific implementation guides.

Figure 2 X12 997 (Functional Acknowledgment) Segment Table

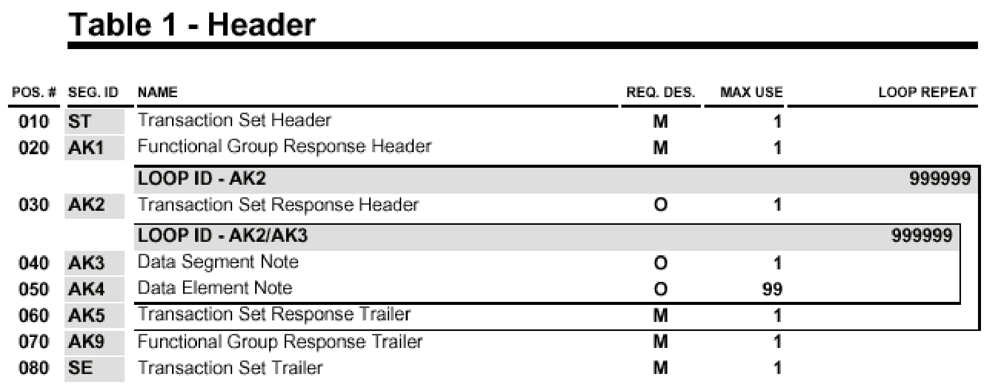
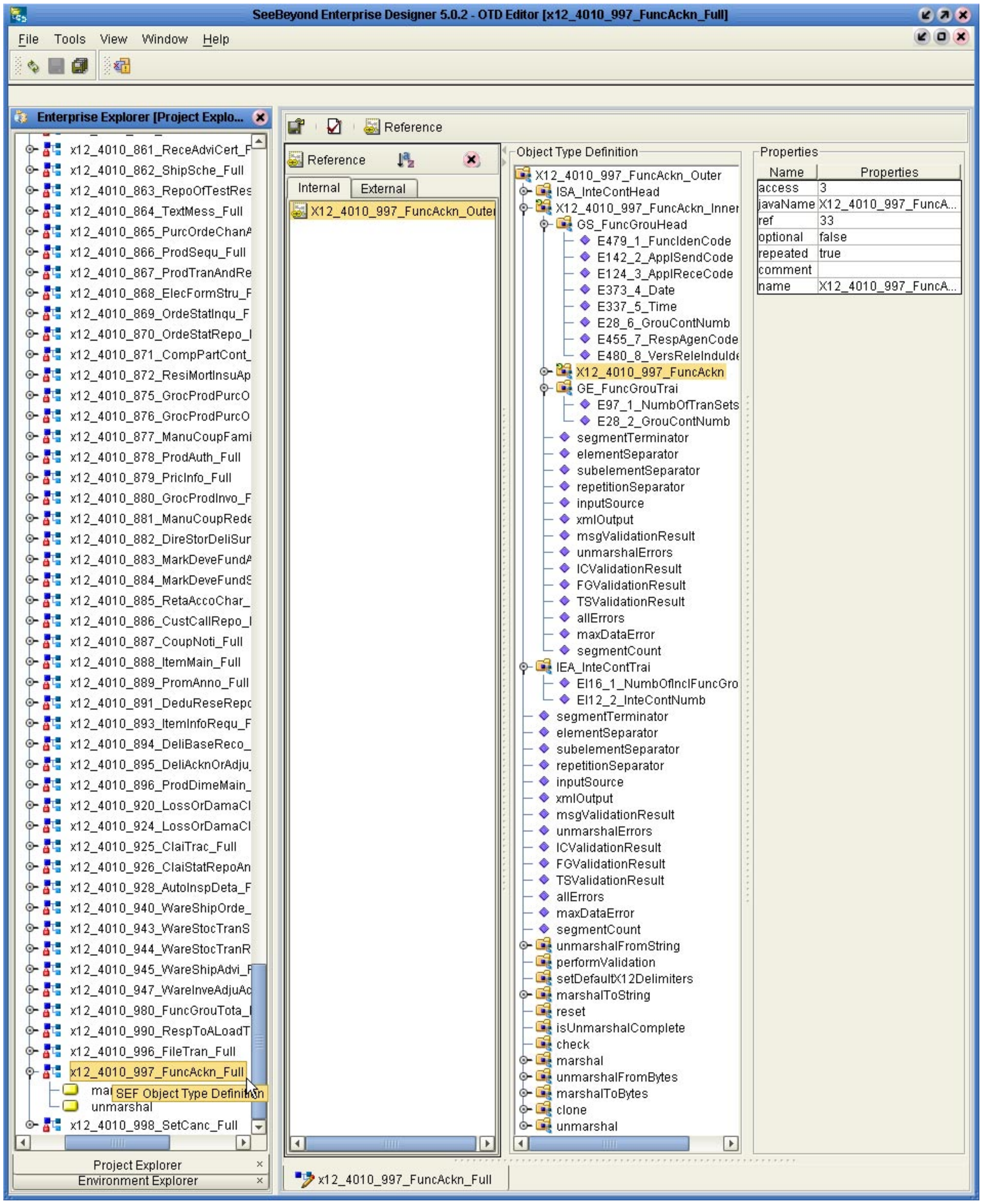


Table 1 - Header

POS. #	SEG. ID	NAME	REQ. DES.	MAX USE	LOOP REPEAT
010	ST	Transaction Set Header	M	1	
020	AK1	Functional Group Response Header	M	1	
		LOOP ID - AK2			999999
030	AK2	Transaction Set Response Header	O	1	
		LOOP ID - AK2/AK3			999999
040	AK3	Data Segment Note	O	1	
050	AK4	Data Element Note	O	99	
060	AK5	Transaction Set Response Trailer	M	1	
070	AK9	Functional Group Response Trailer	M	1	
080	SE	Transaction Set Trailer	M	1	

Figure 3 shows the same transaction as viewed in the OTD Editor.

Figure 3 X12 997 (Functional Acknowledgment) Viewed in OTD Editor


SeeBeyond Enterprise Designer 5.0.2 - OTD Editor [x12_4010_997_FuncAckn_Full]
File
Tools
View
Window
Help
Enterprise Explorer [Project Explo...
x12_4010_861_ReceAdviCert_F
x12_4010_862_ShipSche_Full
x12_4010_863_RepoOfTestRes
x12_4010_864_TextMess_Full
x12_4010_865_PurcOrdeChanA
x12_4010_866_ProdSequ_Full
x12_4010_867_ProdTranAndRe
x12_4010_868_ElecFormStru_F
x12_4010_869_OrdeStatInqu_F
x12_4010_870_OrdeStatRepo_I
x12_4010_871_CompPartCont_
x12_4010_872_ResiMortInsuAp
x12_4010_875_GrocProdPurcO
x12_4010_876_GrocProdPurcO
x12_4010_877_ManuCoupFami
x12_4010_878_ProdAuth_Full
x12_4010_879_PricInfo_Full
x12_4010_880_GrocProdInvo_F
x12_4010_881_ManuCoupRede
x12_4010_882_DireStorDeliSur
x12_4010_883_MarkDeveFundA
x12_4010_884_MarkDeveFundS
x12_4010_885_RetaAccoChar_
x12_4010_886_CustCallRepo_I
x12_4010_887_CoupNoti_Full
x12_4010_888_ItemMain_Full
x12_4010_889_PromAnno_Full
x12_4010_891_DeduReseRepo
x12_4010_893_ItemInfoRequ_F
x12_4010_894_DeliBaseReco_
x12_4010_895_DeliAcknOrAdju_
x12_4010_896_ProdDimeMain_
x12_4010_920_LossOrDamaCl
x12_4010_924_LossOrDamaCl
x12_4010_925_ClaiTrac_Full
x12_4010_926_ClaiStatRepoAn
x12_4010_928_AutoInspDeta_F
x12_4010_940_WareShipOrde_
x12_4010_943_WareStocTranS
x12_4010_944_WareStocTranR
x12_4010_945_WareShipAdvi_F
x12_4010_947_WareInveAdjuAc
x12_4010_980_FuncGrouTota_I
x12_4010_990_RespToALoadT
x12_4010_996_FileTran_Full
x12_4010_997_FuncAckn_Full
mai
SEF Object Type Definition
unmarshal
x12_4010_998_SetCanc_Full
Project Explorer
Environment Explorer
Reference
Reference
Internal
External
X12_4010_997_FuncAckn_Outer
Object Type Definition
X12_4010_997_FuncAckn_Outer
ISA_InteContHead
X12_4010_997_FuncAckn_Inner
GS_FuncGrouHead
E479_1_FuncIdenCode
E142_2_ApplSendCode
E124_3_ApplReceCode
E373_4_Date
E337_5_Time
E28_6_GrouContNumb
E455_7_RespAgenCode
E480_8_VersReleIndulde
X12_4010_997_FuncAckn
GE_FuncGrouTrai
E97_1_NumbOfTranSets
E28_2_GrouContNumb
segmentTerminator
elementSeparator
subelementSeparator
repetitionSeparator
inputSource
xmlOutput
msgValidationResult
unmarshalErrors
ICValidationResult
FGValidationResult
TSValidationResult
allErrors
maxDataError
segmentCount
IEA_InteContTrai
EI16_1_NumbOfInclFuncGro
EI12_2_InteContNumb
segmentTerminator
elementSeparator
subelementSeparator
repetitionSeparator
inputSource
xmlOutput
msgValidationResult
unmarshalErrors
ICValidationResult
FGValidationResult
TSValidationResult
allErrors
maxDataError
segmentCount
unmarshalFromString
performValidation
setDefaultX12Delimiters
marshalToString
reset
isUnmarshalComplete
check
marshal
unmarshalFromBytes
marshalToBytes
clone
unmarshal
Properties
x12_4010_997_FuncAckn_Full

Name	Properties
access	3
javaName	X12_4010_997_FuncA...
ref	33
optional	false
repeated	true
comment	
name	X12_4010_997_FuncA...

2.3.1. Transaction Set (ST/SE)

Each transaction set (also called a transaction) contains three things:

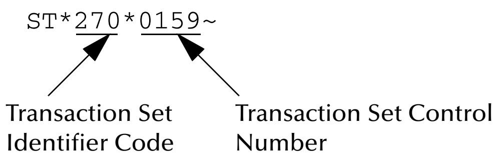
- A transaction set header
- A transaction set trailer
- A single message, enveloped within the header and footer

The transaction has a three-digit code, a text title, and a two-letter code; for example, **997, Functional Acknowledgment (FA)**.

The transaction is composed of logically related pieces of information, grouped into units called segments. For example, one segment used in the transaction set might convey the address: city, state, postal code, and other geographical information. A transaction set can contain multiple segments. For example, the address segment could be used repeatedly to convey multiple sets of address information.

The X12 standard defines the sequence of segments in the transaction set and also the sequence of elements within each segment. The relationship between segments and elements could be compared to the relationship between records and fields in a database environment.

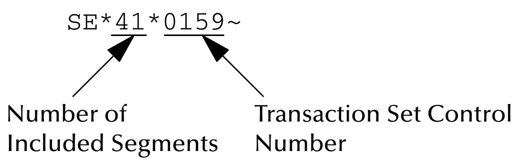
Figure 4 Example of a Transaction Set Header (ST)



```
ST*270*0159~
```

Transaction Set Identifier Code
Transaction Set Control Number

Figure 5 Example of a Transaction Set Trailer (SE)



```
SE*41*0159~
```

2.3.2. Functional Group (GS/GE)

A functional group is composed of one or more transaction sets, all of the same type, that can be batched together in one transmission. The functional group is defined by the header and trailer; the Functional Group Header (GS) appears at the beginning, and the Functional Group Trailer (GE) appears at the end. Many transaction sets can be included in the functional group, but all transactions must be of the same type.

Within the functional group, each transaction set is assigned a functional identifier code, which is the first data element of the header segment. The transaction sets that constitute a specific functional group are identified by this functional ID code.

The functional group header (GS) segment contains the following information:

- Functional ID code (the two-letter transaction code; for example, PO for an 850 Purchase Order, HS for a 270 Eligibility, Coverage or Benefit Inquiry) to indicate the type of transaction in the functional group
- Identification of sender and receiver
- Control information (the functional group control numbers in the header and trailer segments must be identical)
- Date and time

The functional group trailer (GE) segment contains the following information:

- Number of transaction sets included
- Group control number (originated and maintained by the sender)

Figure 6 Example of a Functional Group Header (GS)

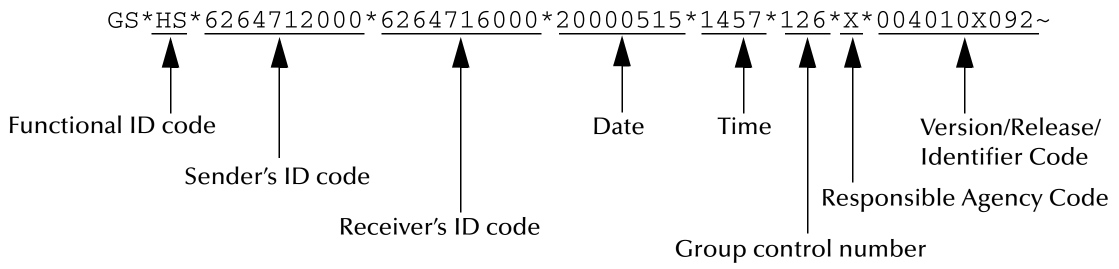
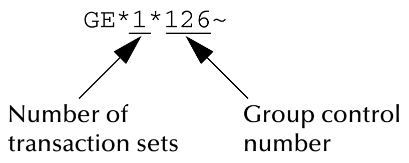

```
GS*HS*6264712000*6264716000*20000515*1457*126*X*004010X092~
```

Figure 7 Example of a Functional Group Trailer (GE)


```
GE*1*126~
```

2.3.3. Interchange Envelope (ISA/IEA)

The interchange envelope is the wrapper for all the data to be sent in one batch. It can contain multiple functional groups. This means that transactions of different types can be included in the interchange envelope, with each type of transaction stored in a separate functional group.

The interchange envelope is defined by the header and trailer; the Interchange Control Header (ISA) appears at the beginning, and the Interchange Control Trailer (IEA) appears at the end.

As well as enveloping one or more functional groups, the interchange header and trailer segments include the following information:

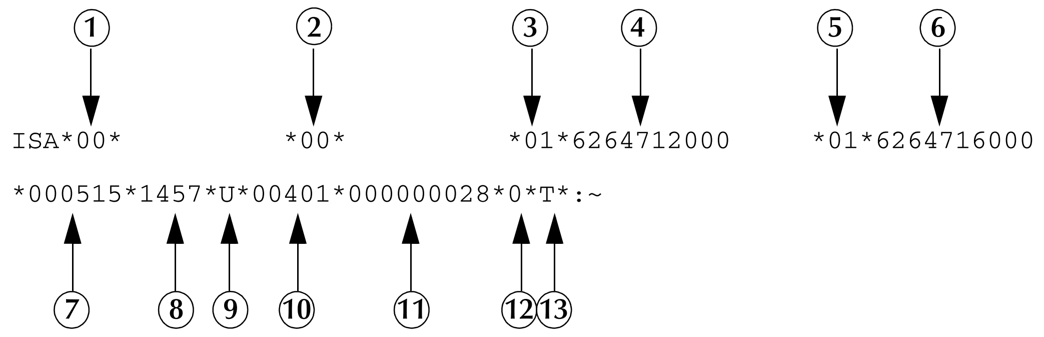
- Data element separators and data segment terminator

- Identification of sender and receiver
- Control information (used to verify that the message was correctly received)
- Authorization and security information, if applicable

The sequence of information that is transmitted is as follows:

- Interchange header
- Optional interchange-related control segments
- Actual message information, grouped by transaction type into functional groups
- Interchange trailer

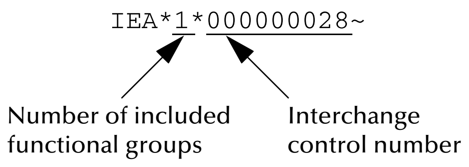
Figure 8 Example of an Interchange Header (ISA)


1
2
3
4
5
6
ISA*00*   *00*   *01*6264712000   *01*6264716000
*000515*1457*U*00401*000000028*0*T*:~
7
8
9
10
11
12
13

Interchange Header Segments from Figure 8:

1 Authorization Information Qualifier	8 Time
2 Security Information Qualifier	9 Repetition Separator
3 Interchange ID Qualifier	10 Interchange Control Version Number
4 Interchange Sender ID	11 Interchange Control Number
5 Interchange ID Qualifier	12 Acknowledgment Requested
6 Interchange Receiver ID	13 Usage Indicator
7 Date	

Figure 9 Example of an Interchange Trailer (IEA)


IEA*1*000000028~
Number of included functional groups
Interchange control number

2.3.4. Control Numbers

The X12 standard includes a control number for each enveloping layer:

- ISA13—Interchange Control Number
- GS06—Functional Group Control Number
- ST02—Transaction Set Control Number

The control numbers act as identifiers, useful in message identification and tracking. eXchange Integrator includes a flag for each control number, so you can choose not to assign control numbers to outgoing messages and not to store control numbers on incoming messages.

ISA13 (Interchange Control Number)

The ISA13 is assigned by the message sender. It must be unique for each interchange. This is the primary means used by eXchange Integrator to identify an individual interchange.

GS06 (Functional Group Control Number)

The GS06 is assigned by the sender. It must be unique within the Functional Group assigned by the originator for a transaction set.

Note: *The Functional Group control number GS06 in the header must be identical to the same data element in the associated Functional Group trailer, GE02.*

ST02 (Transaction Set Control Number)

The ST02 is assigned by the sender, and is stored in the transaction set header. It must be unique within the Functional Group.

Note: *The control number in ST02 must be identical with the SE02 element in the transaction set trailer, and must be unique within a Functional Group (GS-GE). Once you have defined a value for SE02, eXchange Integrator uses the same value for SE02.*

2.4 Backward Compatibility

Each version of X12 is slightly different. Each new version has some new transactions; in addition, existing transactions might have changed.

New versions of X12 are usually backward compatible; however, this is not a requirement of the X12 rules. You should not expect different versions of X12 to be backward compatible, but you can expect that when you analyze the differences only a few changes are required in the message structures.

Note: *In this context backward compatible means that software that parses one version might not be able to parse the next version, even if the software ignores any unexpected new segments, data elements at the end of segments, and sub-elements at the end of composite data elements. Not backward compatible means that required segments can disappear entirely, data elements can change format and usage, and required data elements can become optional.*

2.5 Messages

An example of an X12 version is shown in Table 3. This table lists the transactions of X12 version 4010. Compare this table with **Figure 11 on page 38**.

Table 3 Transactions Included in X12 Version 4010

Number	Title
101	Name and Address Lists
104	Air Shipment Information
105	Business Entity Filings
106	Motor Carrier Rate Proposal
107	Request for Motor Carrier Rate Proposal
108	Response to a Motor Carrier Rate Proposal
109	Vessel Content Details
110	Air Freight Details and Invoice
112	Property Damage Report
120	Vehicle Shipping Order
121	Vehicle Service
124	Vehicle Damage
125	Multilevel Railcar Load Details
126	Vehicle Application Advice
127	Vehicle Baying Order
128	Dealer Information
129	Vehicle Carrier Rate Update
130	Student Educational Record (Transcript)
131	Student Educational Record (Transcript) Acknowledgment
135	Student Loan Application
138	Testing Results Request and Report
139	Student Loan Guarantee Result
140	Product Registration
141	Product Service Claim Response
142	Product Service Claim
143	Product Service Notification
144	Student Loan Transfer and Status Verification
146	Request for Student Educational Record (Transcript)
147	Response to Request for Student Educational Record (Transcript)
148	Report of Injury, Illness or Incident
149	Notice of Tax Adjustment or Assessment

Table 3 Transactions Included in X12 Version 4010 (Continued)

Number	Title
150	Tax Rate Notification
151	Electronic Filing of Tax Return Data Acknowledgment
152	Statistical Government Information
153	Unemployment Insurance Tax Claim or Charge Information
154	Uniform Commercial Code Filing
155	Business Credit Report
157	Notice of Power of Attorney
159	Motion Picture Booking Confirmation
160	Transportation Automatic Equipment Identification
161	Train Sheet
163	Transportation Appointment Schedule Information
170	Revenue Receipts Statement
175	Court and Law Enforcement Notice
176	Court Submission
180	Return Merchandise Authorization and Notification
185	Royalty Regulatory Report
186	Insurance Underwriting Requirements Reporting
188	Educational Course Inventory
189	Application for Admission to Educational Institutions
190	Student Enrollment Verification
191	Student Loan Pre-Claims and Claims
194	Grant or Assistance Application
195	Federal Communications Commission (FCC) License Application
196	Contractor Cost Data Reporting
197	Real Estate Title Evidence
198	Loan Verification Information
199	Real Estate Settlement Information
200	Mortgage Credit Report
201	Residential Loan Application
202	Secondary Mortgage Market Loan Delivery
203	Secondary Mortgage Market Investor Report
204	Motor Carrier Load Tender
205	Mortgage Note
206	Real Estate Inspection
210	Motor Carrier Freight Details and Invoice

Table 3 Transactions Included in X12 Version 4010 (Continued)

Number	Title
211	Motor Carrier Bill of Lading
212	Motor Carrier Delivery Trailer Manifest
213	Motor Carrier Shipment Status Inquiry
214	Transportation Carrier Shipment Status Message
215	Motor Carrier Pick-up Manifest
216	Motor Carrier Shipment Pick-up Notification
217	Motor Carrier Loading and Route Guide
218	Motor Carrier Tariff Information
219	Logistics Service Request
220	Logistics Service Response
222	Cartage Work Assignment
223	Consolidators Freight Bill and Invoice
224	Motor Carrier Summary Freight Bill Manifest
225	Response to a Cartage Work Assignment
242	Data Status Tracking
244	Product Source Information
248	Account Assignment/Inquiry and Service/Status
249	Animal Toxicological Data
250	Purchase Order Shipment Management Document
251	Pricing Support
252	Insurance Producer Administration
255	Underwriting Information Services
256	Periodic Compensation
260	Application for Mortgage Insurance Benefits
261	Real Estate Information Request
262	Real Estate Information Report
263	Residential Mortgage Insurance Application Response
264	Mortgage Loan Default Status
265	Real Estate Title Insurance Services Order
266	Mortgage or Property Record Change Notification
267	Individual Life, Annuity and Disability Application
268	Annuity Activity
270	Eligibility, Coverage or Benefit Inquiry
271	Eligibility, Coverage or Benefit Information
272	Property and Casualty Loss Notification

Table 3 Transactions Included in X12 Version 4010 (Continued)

Number	Title
273	Insurance/Annuity Application Status
275	Patient Information
276	Health Care Claim Status Request
277	Health Care Claim Status Notification
278	Health Care Services Review Information
280	Voter Registration Information
285	Commercial Vehicle Safety and Credentials Information Exchange
286	Commercial Vehicle Credentials
288	Wage Determination
290	Cooperative Advertising Agreements
300	Reservation (Booking Request) (Ocean)
301	Confirmation (Ocean)
303	Booking Cancellation (Ocean)
304	Shipping Instructions
309	U.S. Customs Manifest
310	Freight Receipt and Invoice (Ocean)
311	Canadian Customs Information
312	Arrival Notice (Ocean)
313	Shipment Status Inquiry (Ocean)
315	Status Details (Ocean)
317	Delivery/Pickup Order
319	Terminal Information
322	Terminal Operations and Intermodal Ramp Activity
323	Vessel Schedule and Itinerary (Ocean)
324	Vessel Stow Plan (Ocean)
325	Consolidation of Goods In Container
326	Consignment Summary List
350	U.S. Customs Status Information
352	U.S. Customs Carrier General Order Status
353	U.S. Customs Events Advisory Details
354	U.S. Customs Automated Manifest Archive Status
355	U.S. Customs Acceptance/Rejection
356	U.S. Customs Permit to Transfer Request
357	U.S. Customs In-Bond Information
358	U.S. Customs Consist Information

Table 3 Transactions Included in X12 Version 4010 (Continued)

Number	Title
361	Carrier Interchange Agreement (Ocean)
362	Cargo Insurance Advice of Shipment
404	Rail Carrier Shipment Information
410	Rail Carrier Freight Details and Invoice
414	Rail Carhire Settlements
417	Rail Carrier Waybill Interchange
418	Rail Advance Interchange Consist
419	Advance Car Disposition
420	Car Handling Information
421	Estimated Time of Arrival and Car Scheduling
422	Shipper's Car Order
423	Rail Industrial Switch List
425	Rail Waybill Request
426	Rail Revenue Waybill
429	Railroad Retirement Activity
431	Railroad Station Master File
432	Rail Deprescription
433	Railroad Reciprocal Switch File
434	Railroad Mark Register Update Activity
435	Standard Transportation Commodity Code Master
436	Locomotive Information
437	Railroad Junctions and Interchanges Activity
440	Shipment Weights
451	Railroad Event Report
452	Railroad Problem Log Inquiry or Advice
453	Railroad Service Commitment Advice
455	Railroad Parameter Trace Registration
456	Railroad Equipment Inquiry or Advice
460	Railroad Price Distribution Request or Response
463	Rail Rate Reply
466	Rate Request
468	Rate Docket Journal Log
470	Railroad Clearance
475	Rail Route File Maintenance
485	Ratemaking Action

Table 3 Transactions Included in X12 Version 4010 (Continued)

Number	Title
486	Rate Docket Expiration
490	Rate Group Definition
492	Miscellaneous Rates
494	Rail Scale Rates
500	Medical Event Reporting
501	Vendor Performance Review
503	Pricing History
504	Clauses and Provisions
511	Requisition
517	Material Obligation Validation
521	Income or Asset Offset
527	Material Due-In and Receipt
536	Logistics Reassignment
540	Notice of Employment Status
561	Contract Abstract
567	Contract Completion Status
568	Contract Payment Management Report
601	U.S. Customs Export Shipment Information
602	Transportation Services Tender
620	Excavation Communication
625	Well Information
650	Maintenance Service Order
715	Intermodal Group Loading Plan
805	Contract Pricing Proposal
806	Project Schedule Reporting
810	Invoice
811	Consolidated Service Invoice/Statement
812	Credit/Debit Adjustment
813	Electronic Filing of Tax Return Data
814	General Request, Response or Confirmation
815	Cryptographic Service Message
816	Organizational Relationships
818	Commission Sales Report
819	Operating Expense Statement
820	Payment Order/Remittance Advice

Table 3 Transactions Included in X12 Version 4010 (Continued)

Number	Title
821	Financial Information Reporting
822	Account Analysis
823	Lockbox
824	Application Advice
826	Tax Information Exchange
827	Financial Return Notice
828	Debit Authorization
829	Payment Cancellation Request
830	Planning Schedule with Release Capability
831	Application Control Totals
832	Price/Sales Catalog
833	Mortgage Credit Report Order
834	Benefit Enrollment and Maintenance
835	Health Care Claim Payment/Advice
836	Procurement Notices
837	Health Care Claim
838	Trading Partner Profile
839	Project Cost Reporting
840	Request for Quotation
841	Specifications/Technical Information
842	Nonconformance Report
843	Response to Request for Quotation
844	Product Transfer Account Adjustment
845	Price Authorization Acknowledgment/Status
846	Inventory Inquiry/Advice
847	Material Claim
848	Material Safety Data Sheet
849	Response to Product Transfer Account Adjustment
850	Purchase Order
851	Asset Schedule
852	Product Activity Data
853	Routing and Carrier Instruction
854	Shipment Delivery Discrepancy Information
855	Purchase Order Acknowledgment
856	Ship Notice/Manifest

Table 3 Transactions Included in X12 Version 4010 (Continued)

Number	Title
857	Shipment and Billing Notice
858	Shipment Information
859	Freight Invoice
860	Purchase Order Change Request - Buyer Initiated
861	Receiving Advice/Acceptance Certificate
862	Shipping Schedule
863	Report of Test Results
864	Text Message
865	Purchase Order Change Acknowledgment/Request - Seller Initiated
866	Production Sequence
867	Product Transfer and Resale Report
868	Electronic Form Structure
869	Order Status Inquiry
870	Order Status Report
871	Component Parts Content
872	Residential Mortgage Insurance Application
875	Grocery Products Purchase Order
876	Grocery Products Purchase Order Change
877	Manufacturer Coupon Family Code Structure
878	Product Authorization/De-authorization
879	Price Information
880	Grocery Products Invoice
881	Manufacturer Coupon Redemption Detail
882	Direct Store Delivery Summary Information
883	Market Development Fund Allocation
884	Market Development Fund Settlement
885	Retail Account Characteristics
886	Customer Call Reporting
887	Coupon Notification
888	Item Maintenance
889	Promotion Announcement
891	Deduction Research Report
893	Item Information Request
894	Delivery/Return Base Record
895	Delivery/Return Acknowledgment or Adjustment

Table 3 Transactions Included in X12 Version 4010 (Continued)

Number	Title
896	Product Dimension Maintenance
920	Loss or Damage Claim - General Commodities
924	Loss or Damage Claim - Motor Vehicle
925	Claim Tracer
926	Claim Status Report and Tracer Reply
928	Automotive Inspection Detail
940	Warehouse Shipping Order
943	Warehouse Stock Transfer Shipment Advice
944	Warehouse Stock Transfer Receipt Advice
945	Warehouse Shipping Advice
947	Warehouse Inventory Adjustment Advice
980	Functional Group Totals
990	Response to a Load Tender
996	File Transfer
997	Functional Acknowledgment (see Figure 2 and Figure 3)
998	Set Cancellation

2.6 Example of EDI Usage

This section provides an overview of the normal processes involved in EDI payment processing.

Note: *This is just a general overview of how electronic payments processing is used. Not everything said here applies to the use of X12 in processing payments.*

2.6.1. Overview of EDI Payments Processing

EDI payments processing encompasses both collection and disbursement transactions. The exchange of funds is accomplished by means of credit and debit transfers. It can also include a related bank balance, as well as transaction and account analysis reporting mechanisms.

Most non-monetary EDI trading partner communications are handled either directly between the parties or indirectly through their respective value added networks (VANs). However, the exchange of funds requires a financial intermediary. This is normally the bank or banks that hold deposit accounts of the two parties.

EDI involves the exchange of remittance information along with the order to pay. In the United States this can become complex as two standards are involved in the transaction. The remittance information, which acts as an electronic check stub, can be sent in any of the following ways:

- Directly between trading partners or through their respective EDI VAN mailboxes
- Through the banking system, with the beneficiary's bank sending notice of payment to the beneficiary
- By the originator to the originator's bank as an order to pay, with the originator's bank notifying the beneficiary

The trading partners and the capabilities of their respective banks determine the following:

- The routing of the electronic check stub
- Which of the following the payment is:
    - a debit authorized by the payor and originated by the beneficiary
    - a credit transfer originated by the payor

Types of Information that Is Exchanged Electronically

There are several types of information that can be exchanged electronically between bank and customer, including:

- Daily reports of balances and transactions
- Reports of lockbox and EFT (electronic funds transfer) remittances received by the bank
- Authorizations issued to the bank to honor debit transfers
- Monthly customer account analysis statements
- Account reconcilement statements
- Statements of the demand deposit account

The electronic payment mechanism, which is a subset of EDI, involves two separate activities:

- The exchange of payment orders, causing value to transfer from one account to another
- The exchange of related remittance information in standardized machine-processable formats.

Types of Electronic Payment

The electronic payment can be either of the following:

- Credit transfer, initiated by the payor
- Debit transfer, initiated by the payee as authorized by the payor

Regardless of how the credit transfer was initiated, the payor sends a payment order to its bank in the form of an X12 Payment Order/Remittance Advice (transaction set 820).

The bank then adds data in a format prescribed in the United States by the National Automated Clearing House Association (NACHA) and originates the payment through the Automated Clearing House (ACH) system.

A corporate-to-corporate payment performs two functions:

- Transfers actual monetary value
- Transfers notification of payment from payor to payee

When a credit transfer occurs, these two functions are sometimes treated as one, and sometimes treated separately. The two functions can travel in either of these two ways:

- Together through the banking system
- Separately and by different routes

X12 820 is a data format for transporting a payment order from the originator to its bank. This payment order might be either of the following:

- An instruction to the originator's bank to originate a credit transfer
- An instruction to the trading partner to originate a debit transfer against the payor's bank account

Once this decision has been made, the 820 transports the remittance information to the beneficiary. The transfer can either be through the banking system or by a route that is separate from the transport of funds.

Note: *Whenever the 820 remittance information is not transferred with the funds, it can be transmitted directly from the originator to the beneficiary. It can also be transmitted through an intermediary, such as a VAN.*

Transfer of Funds

Before funds can be applied against an open accounts receivable account, the beneficiary must reconcile the two streams—the payment advice from the receiving bank and the remittance information received through a separate channel—that were separated during the transfer. If this reconciliation does not take place and if the amount of funds received differs from the amount indicated in the remittance advice, the beneficiary might have problems balancing the accounts receivable ledger.

The value transfer begins when the originator issues a payment order to the originator's bank. If a credit transfer is specified, the originator's bank charges the originator's bank account and pays the amount to the beneficiary's bank for credit to the beneficiary's account.

If the payment order specifies a debit transfer, the originator is the beneficiary. In this case, the beneficiary's bank originates the value transfer, and the payor's account is debited (charged) for a set amount, which is credited to the originator's (beneficiary's) bank account. The payor must issue approval to its bank to honor the debit transfer, either before the beneficiary presents the debit transfer or at the same time. This debit authorization or approval can take one of four forms:

- Individual item approval
- Blanket approval of all incoming debits with an upper dollar limit

- Blanket approval for a particular trading partner to originate any debit
- Some combination of the above

2.6.2. Payment-Related EDI Transactions

X12 uses an end-to-end method to route the 820 Payment Order/Remittance Advice from the originator company through the banks to the beneficiary. This means that there might be several relay points between the sender and the receiver.

The 820 is wrapped in an ACH banking transaction for the actual funds transfer between the banks.

2.7 Acknowledgment Types

X12 includes two types of acknowledgment, the TA1 Interchange Acknowledgment and the 997 Functional Acknowledgment.

2.7.1. TA1, Interchange Acknowledgment

The TA1 acknowledgment verifies the interchange envelopes only. The TA1 is a single segment and is unique in the sense that this single segment is transmitted without the GS/GE envelope structures. A TA1 acknowledgment can be included in an interchange with other functional groups and transactions.

2.7.2. 997, Functional Acknowledgment

The 997 includes much more information than the TA1; see **Figure 2 on page 15** and **Figure 3 on page 16**. The 997 was designed to allow trading partners to establish a comprehensive control function as part of the business exchange process.

There is a one-to-one correspondence between a 997 and a functional group. Segments within the 997 identify whether the functional group was accepted or rejected. Data elements that are incorrect can also be identified.

Many EDI implementations have incorporated the acknowledgment process into all of their electronic communications. Typically, the 997 is used as a functional acknowledgment to a functional group that was transmitted previously.

The 997 is the acknowledgment transaction recommended by X12.

The acknowledgment of the receipt of a payment order is an important issue. Most corporate originators want to receive at least a Functional Acknowledgment (997) from the beneficiary of the payment. The 997 is created using the data about the identity and address of the originator found in the ISA and/or GS segments.

Some users argue that the 997 should be used only as a point-to-point acknowledgment and that another transaction set, such as the Application Advice (824) should be used as the end-to-end acknowledgment.

2.7.3. Application Acknowledgments

Application acknowledgments are responses sent from the destination system back to the originating system, acknowledging that the transaction has been successfully or unsuccessfully completed. The application advice (824) is a generic application acknowledgment that can be used in response to any X12 transaction. However, it has to be set up as a response transaction; only TA1 and 997 transactions are sent out automatically.

Other types of responses from the destination system to the originating system, which may also be considered application acknowledgments, are responses to query transactions—for example, the Eligibility Response (271) which is a response to the Eligibility Inquiry (270).Other types of responses from the destination system to the originating system, which may also be considered application acknowledgments, are responses to query transactions—for example, the Eligibility Response (271) which is a response to the Eligibility Inquiry (270).

2.8 Key Parts of EDI Processing Logic

The five key parts of EDI processing logic are listed in Table 4.

Table 4 Key Parts of EDI Processing

Term	Description	Language Analogy	eGate Component
structures	format, segments, loops	syntax rules	OTD elements and fields
validations	data contents “edit” rules	semantic rules	validation methods
translations (also called mappings)	reformatting or conversion	translation	collaborations
enveloping	header and trailer segments	envelope for a written letter	the special “envelope” OTDs: FunctionalGroupEnv and InterchangeEnv
acks	acknowledgments	return receipt	specific acknowledgment elements in the OTD

eGate uses the structures, validations, translations, enveloping, and acknowledgments listed below to support the X12 standard.

2.8.1. Structures

The X12 OTD Library includes pre-built OTDs for all supported X12 versions. These OTDs can be viewed in the OTDEditor, but cannot be modified.

To customize the OTD structure—for example, to add a segment or loop—you must first generate a SEF file (typically using a third-party tool, such as the EDISIM tool from Foresight Corporation). You then use the SEFWizard to generate the OTD.

2.8.2. Validations, Translations, Enveloping, Acknowledgments

Within each OTD are Java methods and Java bean nodes for handling validation; and the marshal and unmarshal methods of the two **envelope** OTDs handle enveloping and de-enveloping. No pre-built translations are supplied with the OTD libraries; these can be built in an eGate GUI called the Java Collaboration Editor (JCE).

Note: *In eGate, X12 translations are called collaborations.*

2.8.3. Trading Partner Agreements

There are three levels of information that guide the final format of a specific transaction. These three levels are:

- The X12 standard

  The Accredited Standards Committee publishes a standard structure for each X12 transaction.

- Industry-specific Implementation Guides

  Specific industries publish Implementation Guides customized for that industry. Normally, these are provided as recommendations only. However, in certain cases, it is extremely important to follow these guidelines. Specifically, since HIPAA regulations are law, it is important to follow the guidelines for these transactions closely.

- Trading Partner Agreements

  It is normal for trading partners to have individual agreements that supplement the standard guides. The specific processing of the transactions in each trading partner's individual system might vary between sites. Because of this, additional documentation that provides information about the differences is helpful to the site's trading partners and simplifies implementation. For example, while a certain code might be valid in an implementation guide, a specific trading partner might not use that code in transactions. It would be important to include that information in a trading partner agreement.

2.9 Additional Information

For more information on the X12 standard, visit the following Web sites:

http://www.disa.org and specifically **http://www.x12.org/x12org/index.cfm**

X12 implementation guides can be obtained from Washington Publishing Company:

http://www.wpc-edi.com; specifically, **http://www.wpc-edi.com/tg4/tg4home.asp**

Note: *This information is correct at the time of going to press; however, SeeBeyond has no control over these sites. If you find the links are no longer correct, use a search engine to search for* ***X12****.*

Chapter 3

Installation

This chapter provides information on installing the X12 OTD library, and shows the resulting Project Explorer tree for the OTDs. It includes both general installation information and step-by-step installation instructions.

The X12 OTD library includes OTDs for the following X12 versions.

Table 5 X12 Versions Supported

▪ 4010	▪ 4020	▪ 4030	▪ 4040	▪ 4050	▪ 4060
▪ 4011	▪ 4021	▪ 4031	▪ 4041	▪ 4051	▪ 4061
▪ 4012	▪ 4022	▪ 4032	▪ 4042	▪ 4052	

Some additional points to note:

- The library OTDs only accept messages with all the envelope segment information. If you need to generate a custom OTD without an envelope segment, use the SEF OTD Wizard.
- Messages can be batched; however, all the messages in one functional group must be of the same message type.

3.1 X12 Libraries

When the X12 OTD Library is installed, the Project Explorer tree adds a new project under the **SeeBeyond > OTD Library > X12** hierarchy. Each version of X12, such as **4010** or **4021**, has a separate folder; for an example, see **Figure 11 on page 38**.

The SeeBeyond > OTD Library > X12 > **envelope** folder contains two additional OTDs for manipulating envelopes as bytes:

- **InterchangeEnv**— parses interchange envelope segments (ISA/IEA and TA1), treating functional groups as bytes.
- **FunctionalGroupEnv**— parses functional group envelope segments (GS/GE), treating transaction sets as bytes.

Note: *The following abbreviations are common: "IC" for "interchange"; "FG" for "functional group"; and "TS" for "Transaction Set."*

3.2 Installation Procedure

The steps for installing the X12 OTD Library are the same as for other products in the ICAN Suite. You can find general product installation instructions in the *ICAN Suite Installation Guide*, which is available on the product media and can also be accessed via Enterprise Manager (Documentation tab).

3.2.1. Uploading to the Repository

Before you begin

- A Repository server must be running on the machine where you will be uploading the product files.
- You must have already uploaded **eGate.sar** (for either eGate 5.0.4 or eGate 5.0.3 with appropriate ESRs), and you must have already uploaded a **license.sar** file that includes a license for the X12 OTD library product.

To upload product files to the Repository

1. On a Windows machine, start a Web browser and point it at the machine and port where the Repository server is running:

   ```
   http://<hostname>:<port>
   ```

   where
   - *<hostname>* is the name of the machine running the Repository server.
   - *<port>* is the starting port number assigned when the Repository was installed.

   For example, the URL you enter might look like either of the following:

   ```
   http://localhost:12001
   http://serv1234.company.com:19876
   ```

2. In the Enterprise Manager **SeeBeyond Customer Login** page, enter your username and password.
3. When Enterprise Manager responds, click the **ADMIN** tab.
4. In the ADMIN page, click **Browse**.
5. In the **Choose file** dialog, click **ProductsManifest.xml**, and then click **Open**.
6. In the ADMIN page, click **Submit**.

   The lower half of the ADMIN page lists the product files you are licensed to upload.
7. In the Products column, find the **ASC X12 OTD Library v40?0** product, and then click the **Browse** button for it.
8. In the **Choose file** dialog, click the corresponding **X12_v40??_OTD.sar** file, and then click **Open**.
9. Repeat the previous two steps for other **.sar** files you want to upload, such as other X12 OTD libraries, eXchange, or SME Web Services.

Note: *SMEWebServices.sar is required for such features as encryption/decryption, signature verification, certificate authentication, and nonrepudiation.*

10. In the ADMIN page, click the | upload now ⁝· | button.

3.2.2. Refreshing Enterprise Designer

Before you begin

- You must have already downloaded and installed Enterprise Designer, and a Repository server must be running on the machine where you uploaded the product files for the X12 OTD Library.

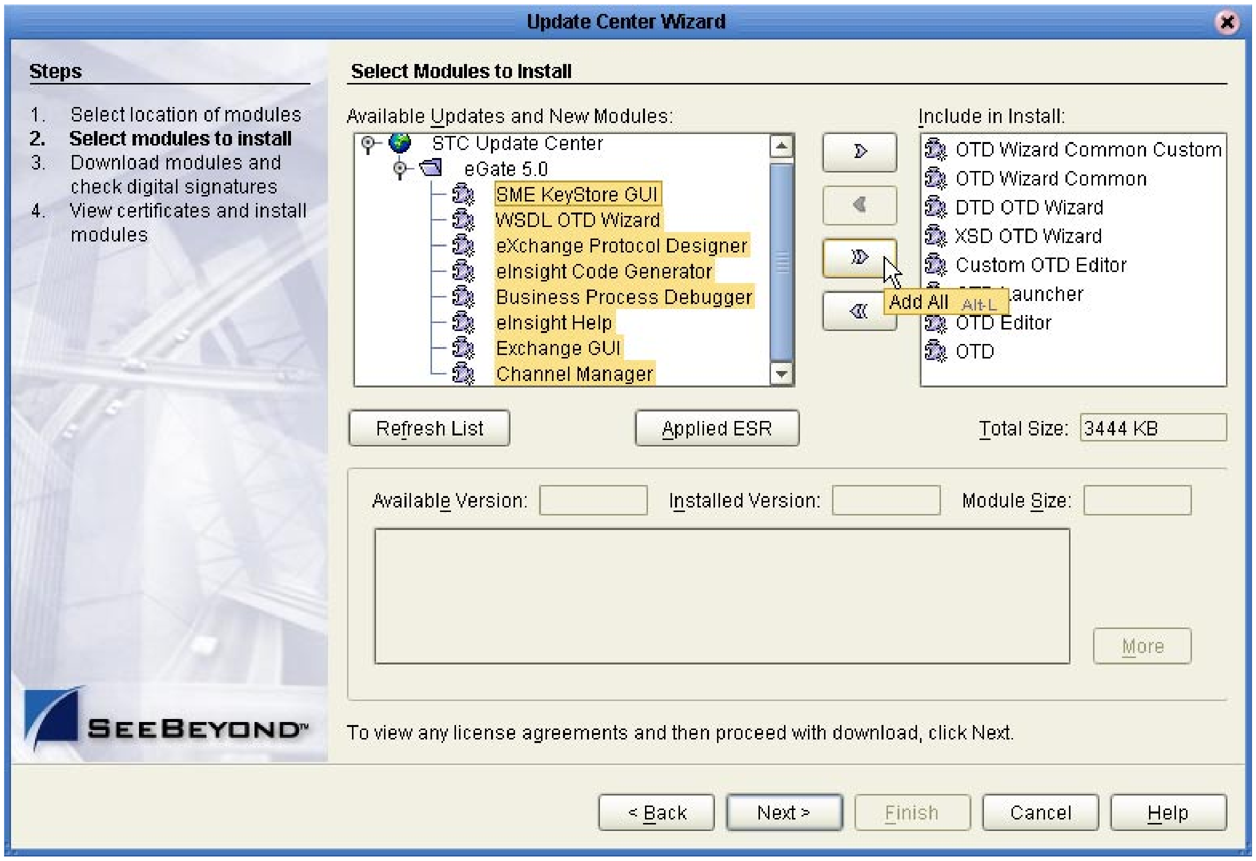
To refresh an existing installation of Enterprise Designer

1 Start Enterprise Designer.

2 On the **Tools** menu, click **Update Center**.

The Update Center shows a list of components ready for updating. See Figure 10.

Figure 10 Update Center Wizard: Select Modules to Install



To view any license agreements and then proceed with download, click Next.

3 Click **Add All** (the button with a doubled chevron pointing to the right).

All modules move from the Available/New pane to the **Include in Install** pane.

4 Click **Next** and, in the next window, click **Accept** to accept the license agreement.

5 When the progress bars indicate the download has ended, click **Next**. Review the certificates and installed modules, and then click **Finish**. When prompted to restart Enterprise Designer, click **OK**.

When Enterprise Designer restarts, installation of the X12 OTD Library is complete.

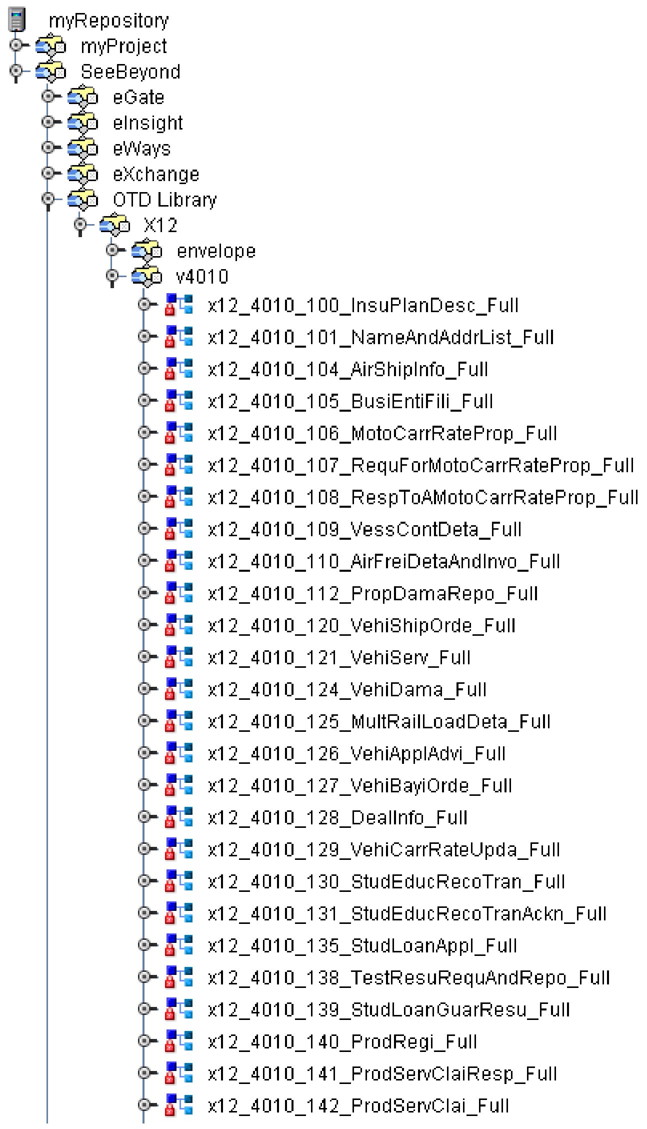
If you need help on details of product installation, see the *SeeBeyond ICAN Suite Installation Guide*.

3.3 X12 OTD Libraries

3.3.1. X12 OTDs

Since there is an OTD for every X12 transaction, installation of each version of X12 includes a large number of OTDs. **Figure 3 on page 16** and Figure 11 below show some of the OTDs installed for a specific version of X12 (in this case, version 4010); compare with **Table 3 on page 21**.

Figure 11 Some of the Transaction Set Structures for X12 Version 4010



```
myRepository
  myProject
  SeeBeyond
    eGate
    eInsight
    eWays
    eXchange
    OTD Library
      X12
        envelope
        v4010
          x12_4010_100_InsuPlanDesc_Full
          x12_4010_101_NameAndAddrList_Full
          x12_4010_104_AirShipInfo_Full
          x12_4010_105_BusiEntiFili_Full
          x12_4010_106_MotoCarrRateProp_Full
          x12_4010_107_RequForMotoCarrRateProp_Full
          x12_4010_108_RespToAMotoCarrRateProp_Full
          x12_4010_109_VessContDeta_Full
          x12_4010_110_AirFreiDetaAndInvo_Full
          x12_4010_112_PropDamaRepo_Full
          x12_4010_120_VehiShipOrde_Full
          x12_4010_121_VehiServ_Full
          x12_4010_124_VehiDama_Full
          x12_4010_125_MultRailLoadDeta_Full
          x12_4010_126_VehiApplAdvi_Full
          x12_4010_127_VehiBayiOrde_Full
          x12_4010_128_DealInfo_Full
          x12_4010_129_VehiCarrRateUpda_Full
          x12_4010_130_StudEducRecoTran_Full
          x12_4010_131_StudEducRecoTranAckn_Full
          x12_4010_135_StudLoanAppl_Full
          x12_4010_138_TestResuRequAndRepo_Full
          x12_4010_139_StudLoanGuarResu_Full
          x12_4010_140_ProdRegi_Full
          x12_4010_141_ProdServClaiResp_Full
          x12_4010_142_ProdServClai_Full
```

3.3.2. Transaction Names

The names for the X12 OTDs are designed to assist you in quickly locating the file you want. The name for each transaction OTD is composed of the same set of elements in the same sequence. The names are constructed as follows (using a v4021 997, Functional Acknowledgment, as an example):

- **x12_** (name of standard followed by underscore)
- **4021_** (name of version followed by underscore)
- **997_FuncAckn** (transaction code and abbreviation for the transaction name)
- **_Full**

Examples:

- The name for a 270, Eligibility Coverage or Benefit Inquiry, for version 4010 is **x12_4010_270_EligCoveOrBeneInqu_Full**
- The name for an 855, Purchase Order Acknowledgment, in version 4032 is **x12_4032_855_PurcOrdeAckn_Full**

Chapter 4

Working With the X12 OTDs

This chapter provides information on additional features built into the X12 OTDs, and instructions on working with the OTDs and on testing them.

- See **"Viewing an X12 OTD in the OTD Editor" on page 41**.

It also provides information on using the Java methods provided within the OTDs, and other general information about using the X12 OTD Library.

- See **"Setting the Delimiters" on page 43** and **"Methods for Getting and Setting" on page 43**. (Further details are provided in **Chapter 5** "Java Methods for X12 OTDs" on page 52.)

To test that your data is being mapped correctly by the OTD, and that the data is valid based on definitions and business rules, you can run **performValidation()** within the Java Collaboration Editor.

- See **"Using Validation in the Java Collaboration Editor" on page 45**.

Information on limitations you should know about the X12 OTD Library is provided in **"Limitations of X12 OTDs" on page 49**.

4.1 Importing .jar Files

If your project contains one or more Java collaborations that access bean nodes for reporting errors and exceptions of X12 OTDs (see **"Bean Nodes for Reporting Errors and Exceptions" on page 53**), then you must import a **.jar** file as described below.

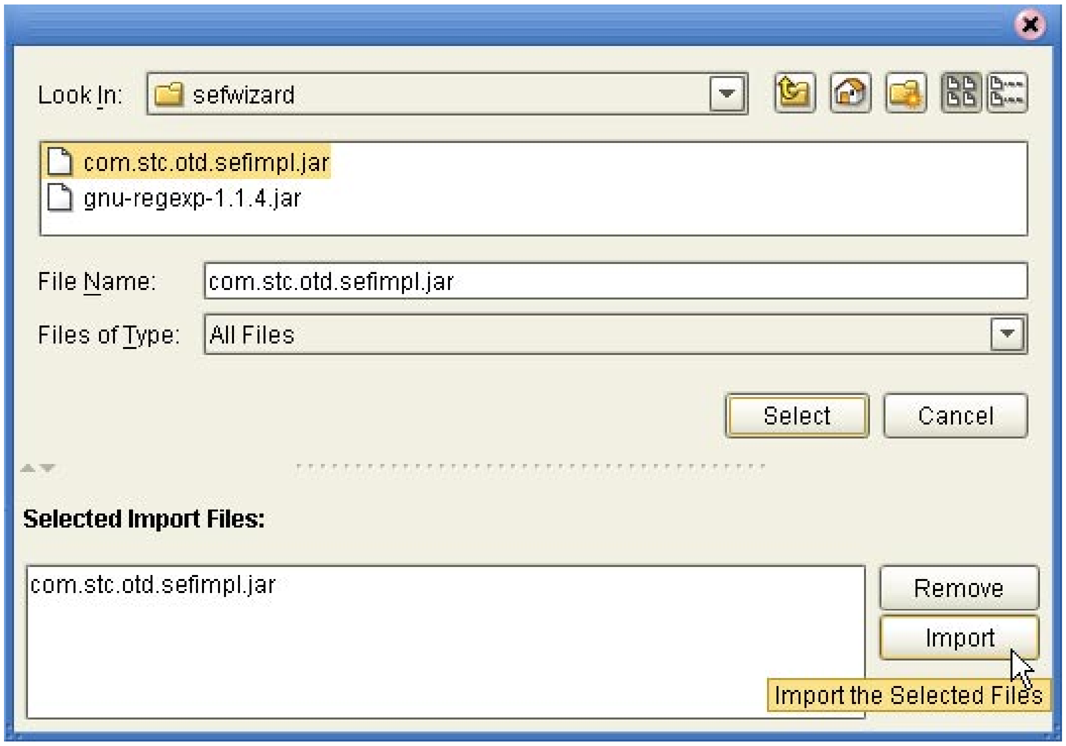
Before you begin

- You must have completed all the other installation steps.
- A Repository server must be running on the machine where you uploaded the X12 OTD Library product files.
- You must have already created a project.

To import com.stc.otd.sefimpl.jar

1. Start Enterprise Designer and, if necessary, create a project.
2. Right-click the project and, on the popup context menu, point at **New** and click **File**.
3. Navigate to the folder <*ican50*>\edesigner\usrdir\modules\ext\sefwizard\, select **com.stc.otd.sefimpl.jar**, and click **Import**. See Figure 12.

Figure 12 Importing sefimpl.jar



4 Later, in the Java Collaboration Editor, you will use the **Import JAR file** button on the tool palette to add **sefimpl.jar** to a collaboration that uses the X12 OTD.

4.2 Viewing an X12 OTD in the OTD Editor

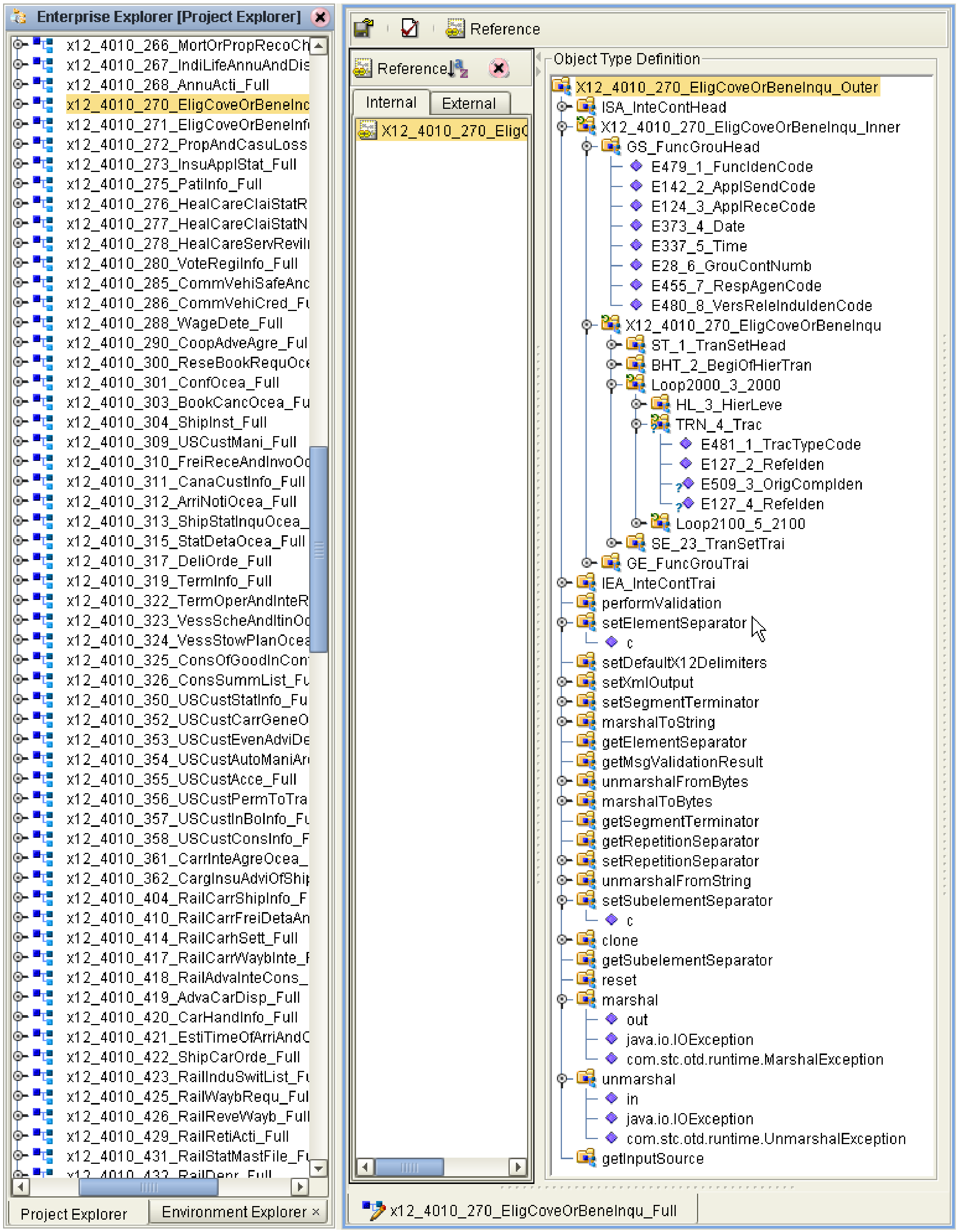
To view an X12 OTD (or any other OTD), simply double-click the name in the Project Explorer tree. The OTD Editor automatically opens to display it. Within the OTD Editor, you can expand or contract a parent node by single-clicking the icon to its left, or by double-clicking the node name. For some of the items, help is available by hovering your cursor over the item.

For elements other than bean nodes, the following naming conventions apply:

- Each element name begins with **E**
- Each segment loop name begins with **Loop**

An example of an X12 270 transaction in the OTD Editor is shown in Table 13 on page 42. The OTD shown in Figure 13 is **x12_4010_270_EligCoveOrBeneInqu_Full**. Some of its parent nodes are fully expanded, some partly so, and some full collapsed. In this example, the root node is **X12_4010_270_EligCoveOrBeneInqu_Outer**. This pattern holds for all the X12 OTDs: the root node name is the same as the OTD name, but with the first letter in upper case (**X** instead of x) and the string **_Outer** replacing the string **_Full**. Under this root node, the first node is the ISA header node, and then comes the node **X12_4010_270_EligCoveOrBeneInqu_Inner**, which references the enveloping information.

Figure 13 X12 270 Transaction in the OTD Editor



```
Enterprise Explorer [Project Explorer]
x12_4010_266_MortOrPropRecoCh
x12_4010_267_IndiLifeAnnuAndDis
x12_4010_268_AnnuActi_Full
x12_4010_270_EligCoveOrBeneInc
x12_4010_271_EligCoveOrBeneInf
x12_4010_272_PropAndCasuLoss
x12_4010_273_InsuApplStat_Full
x12_4010_275_PatiInfo_Full
x12_4010_276_HealCareClaiStatR
x12_4010_277_HealCareClaiStatN
x12_4010_278_HealCareServRevil
x12_4010_280_VoteRegiInfo_Full
x12_4010_285_CommVehiSafeAnc
x12_4010_286_CommVehiCred_Fu
x12_4010_288_WageDete_Full
x12_4010_290_CoopAdveAgre_Ful
x12_4010_300_ReseBookRequOce
x12_4010_301_ConfOcea_Full
x12_4010_303_BookCancOcea_Fu
x12_4010_304_ShipInst_Full
x12_4010_309_USCustMani_Full
x12_4010_310_FreiReceAndInvoOc
x12_4010_311_CanaCustInfo_Full
x12_4010_312_ArriNotiOcea_Full
x12_4010_313_ShipStatInquOcea_
x12_4010_315_StatDetaOcea_Full
x12_4010_317_DeliOrde_Full
x12_4010_319_TermInfo_Full
x12_4010_322_TermOperAndInteR
x12_4010_323_VessScheAndItinOc
x12_4010_324_VessStowPlanOcea
x12_4010_325_ConsOfGoodInCon
x12_4010_326_ConsSummList_Fu
x12_4010_350_USCustStatInfo_Fu
x12_4010_352_USCustCarrGeneO
x12_4010_353_USCustEvenAdviDe
x12_4010_354_USCustAutoManiAr
x12_4010_355_USCustAcce_Full
x12_4010_356_USCustPermToTra
x12_4010_357_USCustInBoInfo_Fu
x12_4010_358_USCustConsInfo_F
x12_4010_361_CarrInteAgreOcea_
x12_4010_362_CargInsuAdviOfShip
x12_4010_404_RailCarrShipInfo_F
x12_4010_410_RailCarrFreiDetaAn
x12_4010_414_RailCarhSett_Full
x12_4010_417_RailCarrWaybInte_F
x12_4010_418_RailAdvaInteCons_
x12_4010_419_AdvaCarDisp_Full
x12_4010_420_CarHandInfo_Full
x12_4010_421_EstiTimeOfArriAndC
x12_4010_422_ShipCarOrde_Full
x12_4010_423_RailInduSwitList_Fu
x12_4010_425_RailWaybRequ_Full
x12_4010_426_RailReveWayb_Full
x12_4010_429_RailRetiActi_Full
x12_4010_431_RailStatMastFile_Fu
x12_4010_432_RailDepr_Full
Project Explorer | Environment Explorer
```

```
Reference
Reference
Internal | External
X12_4010_270_EligC

Object Type Definition
X12_4010_270_EligCoveOrBeneInqu_Outer
  ISA_InteContHead
  X12_4010_270_EligCoveOrBeneInqu_Inner
    GS_FuncGrouHead
      E479_1_FuncIdenCode
      E142_2_ApplSendCode
      E124_3_ApplReceCode
      E373_4_Date
      E337_5_Time
      E28_6_GrouContNumb
      E455_7_RespAgenCode
      E480_8_VersReleIndulIdenCode
    X12_4010_270_EligCoveOrBeneInqu
      ST_1_TranSetHead
      BHT_2_BegiOfHierTran
      Loop2000_3_2000
        HL_3_HierLeve
        TRN_4_Trac
          E481_1_TracTypeCode
          E127_2_RefeIden
          E509_3_OrigCompIden
          E127_4_RefeIden
        Loop2100_5_2100
      SE_23_TranSetTrai
    GE_FuncGrouTrai
  IEA_InteContTrai
  performValidation
  setElementSeparator
    c
  setDefaultX12Delimiters
  setXmlOutput
  setSegmentTerminator
  marshalToString
  getElementSeparator
  getMsgValidationResult
  unmarshalFromBytes
  marshalToBytes
  getSegmentTerminator
  getRepetitionSeparator
  setRepetitionSeparator
  unmarshalFromString
  setSubelementSeparator
    c
  clone
  getSubelementSeparator
  reset
  marshal
    out
    java.io.IOException
    com.stc.otd.runtime.MarshalException
  unmarshal
    in
    java.io.IOException
    com.stc.otd.runtime.UnmarshalException
  getInputSource
x12_4010_270_EligCoveOrBeneInqu_Full
```

4.3 Setting the Delimiters

The X12 OTDs must include some way for delimiters to be defined so that they can be mapped successfully from one OTD to another. The X12 delimiters are as follows:

- Data Element Separator (default is an asterisk)
- Subelement Separator/Component Element Separator (default is a colon)
- Repetition Separator (version 4020 and later) (default is a plus sign)
- Segment Terminator (default is a tilde)

Two delimiters—Repetition Separator and Subelement Separator—are explicitly specified in the interchange header segment (ISA). The other two delimiters are implicitly defined within the structure of the ISA, by their first usage. For example, after the fourth character defines the Data Element Separator, the same character is used subsequently to delimit all data elements; and after the 107th character defines the Segment Terminator, the same character is used subsequently to delimit all segments.

Because the OTD automatically detects delimiters while unmarshaling, you need not (and should not) specify delimiters for an incoming message; any delimiters that are set before unmarshaling are ignored, and the unmarshal() function picks up the delimiter being used in the ISA segment of the incoming message.

You can specify delimiters in two ways:

- You can set the Subelement Separator and Repetition Separator from the corresponding elements within the ISA segment.
- You can set the delimiters in the Java Collaboration Editor using bean nodes that are provided in the OTDs. Specific information on using bean nodes to get and set these delimiter values is provided in **Chapter 5**:
    - elementSeparator (see **getElementSeparator** on page 58)
    - subelementSeparator (see **getSubelementSeparator** on page 62)
    - repetitionSeparator (see **getRepetitionSeparator** on page 60)
    - segmentTerminator (see **getSegmentTerminator** on page 61)

If the input data is already in X12 format, you can use the "get" methods to get the delimiters from the input data. If the collaboration is putting the data into X12 format, you can use the "set" methods to set the delimiters in the output OTD. See **"Methods for Getting and Setting" on page 43**.

4.4 Methods for Getting and Setting

Bean nodes automatically have **get** and **set** methods associated with them; in other words, a bean node named *theBeanNode* has a method **get*TheBeanNode*()** to read the current value and another method **set*TheBeanNode*()** to write a value. Therefore, do not assume that a node is read/write merely because it has a **set*Node*()** method.

4.4.1. Bean Nodes for Getting and Setting Data

The following bean nodes are available under the root node and at the *xxx*_Outer, *xxx*_Inner, and *xxx* (transaction set) levels:

- **elementSeparator**(char)— to get or set the element separator.
- **inputSource**(byte[])— to get the byte array of original input data source.
- **repetitionSeparator**(char)— to get or set the repetition separator.
- **segmentCount**(int)— to get the segment count at the current level. This node is also available for segment loops.
- **segmentTerminator**(char)— to get or set the segment terminator.
- **subelementSeparator**(char)— to get or set the subelement separator.
- **xmlOutput**(boolean)—to set whether the output should be in XML format.

The following bean node is available from the Loop elements:

- **segmentCount**(int)— to get the segment count at the current level. This node is also available under the root node and at the *xxx*_Outer, *xxx*_Inner, and *xxx* (transaction set) levels.

4.4.2. Bean Nodes for Getting Errors and Results

The following bean nodes are available under the root node and at the *xxx*_Outer, *xxx*_Inner, and *xxx* (transaction set) levels.

- **allErrors**(String[])— to get errors during unmarshaling from the input data and validation results on message and envelopes, in the format of a String array that combines (without duplication) the results from ICValidationResult(), FGValidationResult(), TSValidationResult(), and msgValidationResult().
- **ICValidationResult**(com.stc.otd.runtime.check.sef.ICError[])— to get the interchange envelope validation result, in the format of an array of com.stc.otd.runtime.check.sef.**ICError** objects.
- **FGValidationResult**(com.stc.otd.runtime.check.sef.FGError[])— to get the functional group envelope validation result in the format of an array of com.stc.otd.runtime.check.sef.**FGError** objects.
- **TSValidationResult**(com.stc.otd.runtime.check.sef.TSError[])— to get the transaction set envelope validation result in the format of an array of com.stc.otd.runtime.check.sef.**TSError** objects.
- **maxDataError**(int)— to get or set the maximum number of validation errors to be reported, where **-1** means “no limit.”
- **msgValidationResult**(com.stc.otd.runtime.check.sef.DataError[])— to get validation errors, in the format of com.stc.otd.runtime.check.sef.**DataError** objects.
- **unmarshalErrors**(com.stc.otd.runtime.check.sef.DataError[])— to get errors that occurred during unmarshaling from the input data, in the format of

com.stc.otd.runtime.check.sef.**DataError** objects. The presence of any objects in this array implies that **isUnmarshalComplete()** is false.

4.5 Using Validation in the Java Collaboration Editor

Each of the OTDs in the X12 OTD library includes a Java method for the purpose of validating your data:

- **performValidation***()*

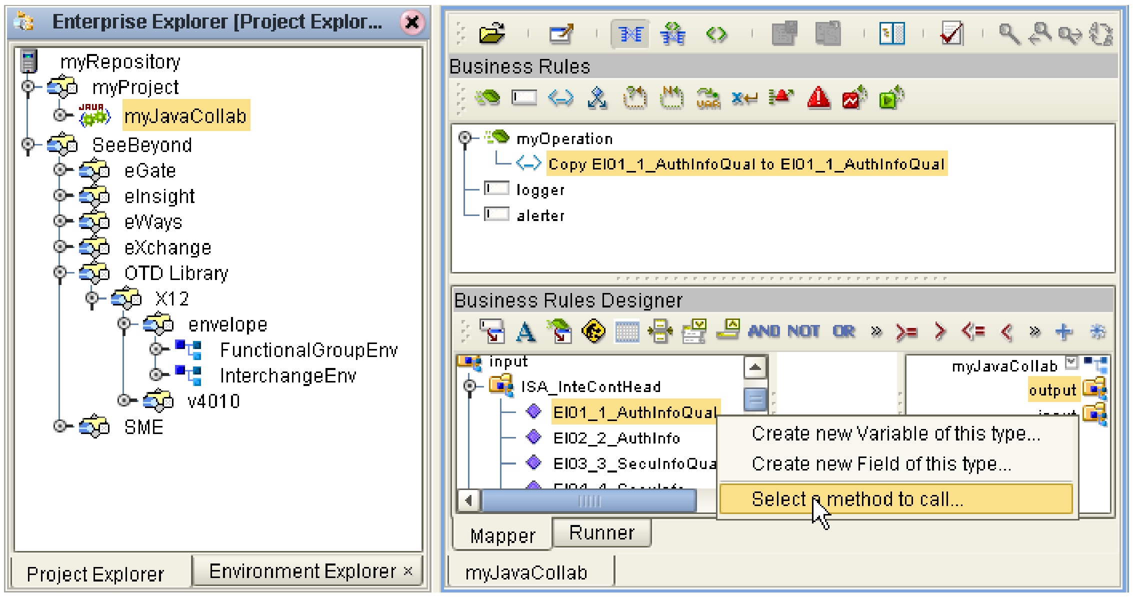
Information on using this method from within Java Collaboration Editor (JCE) GUI is provided below. Technical information on the Java methods is provided in **"Java Methods for X12 OTDs" on page 52**.

4.5.1. Creating a Collaboration Rule to Validate an X12 OTD

The elements that are part of an OTD can be dragged and dropped when two or more OTDs are opened in the Java Collaboration Editor; see the *eGate Integrator User's Guide* for more information. A field on the input (left) side pane can be dragged to a field in the output (right) pane. This action, when highlighted in the Business Rules pane, displays the rule in the Rule Properties pane.

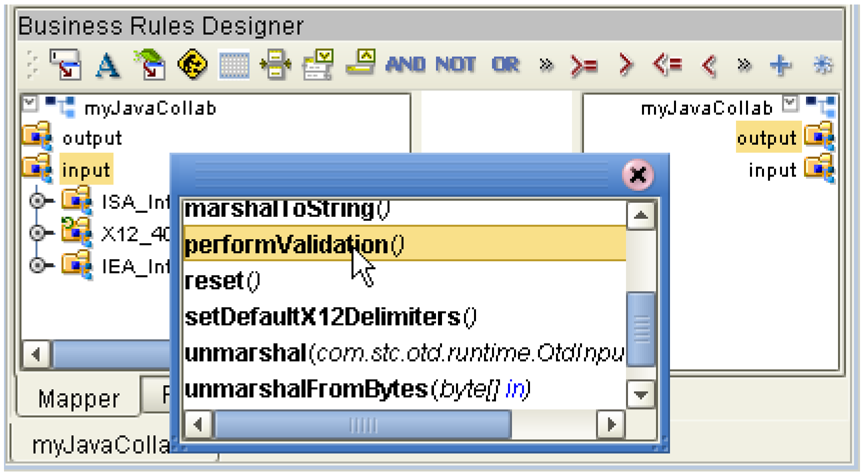
To access the method, right-click the node and, on the context popup menu, click **Select a method to call**. See Figure 14.

Figure 14 Accessing a Method in an X12 OTD



The methods available depend on the node you select. In particular, if you right-click the root node of the OTD, one of the methods available to you is **performValidation()**; see Figure 15.

Figure 15 Accessing the performValidation Method from the Root Node



The **performValidation()** method can be used to validate an X12 message at run time. If the OTD content is found to be invalid, the appropriate error bean nodes are populated (see **"Bean Nodes for Reporting Errors and Exceptions" on page 53**). Therefore, the complete set of bean nodes for reporting errors and exceptions can only be accessed after the call to **performValidation()**.

Note: *Although validation is a useful tool to ensure that data conforms to the definitions and business rules, be aware that it significantly impacts performance.*

4.6 Alternative Formats: ANSI and XML

All the X12 OTDs accept either standard ANSI X12 format or XML format as input, by default; and, by default, output from a collaboration that uses messages from an X12 OTD is in ASC X12 format. However, there is a Java method available for setting the output to XML:

- setXMLOutput (boolean isXML)

If you want to set the collaboration to output XML format, use setXmlOutput(true); in other words, set the xmlOutput bean node to the value **true**.

4.6.1. XML Format for X12

Since there is no de facto XML standard for X12 as yet, the SeeBeyond X12 OTD Library uses Open Business Objects for EDI (OBOE) as the XML format for X12.

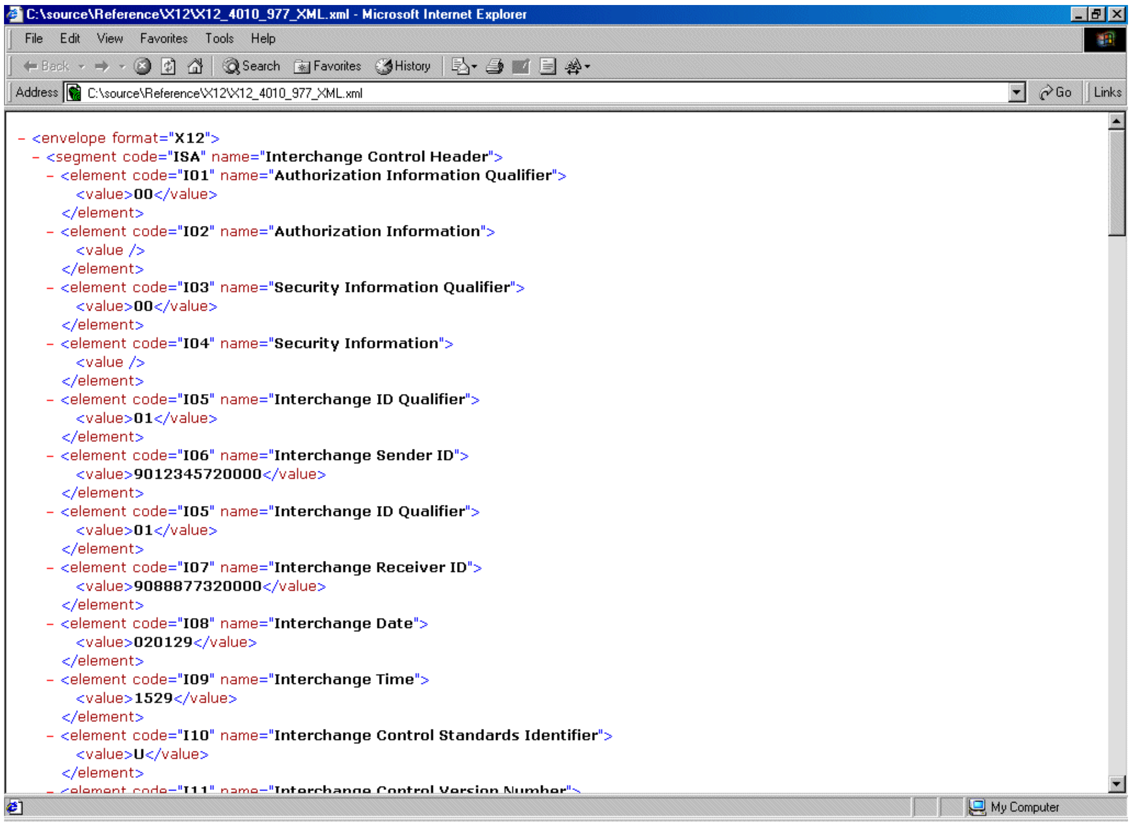
The XML X12 DTD is shown in Figure 16.

Figure 16 XML X12 DTD

```
<!ELEMENT envelope (segment, segment?, functionalgroup+, segment)>
<!ATTLIST envelope format CDATA #IMPLIED>

<!ELEMENT functionalgroup (segment, transactionset+, segment)>

<!ELEMENT transactionset (table+)>
<!ATTLIST transactionset code CDATA #REQUIRED>
<!ATTLIST transactionset name CDATA #IMPLIED>

<!ELEMENT table (segment)+>
<!ATTLIST table section CDATA #IMPLIED>

<!ELEMENT segment ((element | composite)+, segment*)>
<!ATTLIST segment code CDATA #REQUIRED>
<!ATTLIST segment name CDATA #IMPLIED>

<!ELEMENT composite (element)+>
<!ATTLIST composite code CDATA #REQUIRED>
<!ATTLIST composite name CDATA #IMPLIED>

<!ELEMENT element (value)>
<!ATTLIST element code CDATA #REQUIRED>
<!ATTLIST element name CDATA #IMPLIED>

<!ELEMENT value (#PCDATA)>
<!ATTLIST value description CDATA #IMPLIED>
```

Figure 17 shows an X12 997 Functional Acknowledgment, in XML format.

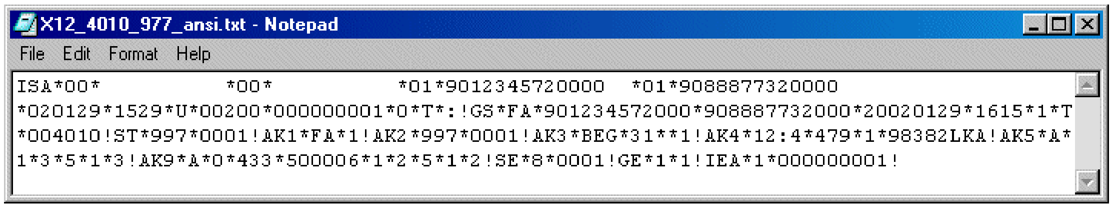
Figure 17 X12 997 Functional Acknowledgment—XML


C:\source\Reference\X12\X12_4010_977_XML.xml - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address C:\source\Reference\X12\X12_4010_977_XML.xml
Go Links
My Computer

```
- <envelope format="X12">
  - <segment code="ISA" name="Interchange Control Header">
    - <element code="I01" name="Authorization Information Qualifier">
        <value>00</value>
      </element>
    - <element code="I02" name="Authorization Information">
        <value />
      </element>
    - <element code="I03" name="Security Information Qualifier">
        <value>00</value>
      </element>
    - <element code="I04" name="Security Information">
        <value />
      </element>
    - <element code="I05" name="Interchange ID Qualifier">
        <value>01</value>
      </element>
    - <element code="I06" name="Interchange Sender ID">
        <value>9012345720000</value>
      </element>
    - <element code="I05" name="Interchange ID Qualifier">
        <value>01</value>
      </element>
    - <element code="I07" name="Interchange Receiver ID">
        <value>9088877320000</value>
      </element>
    - <element code="I08" name="Interchange Date">
        <value>020129</value>
      </element>
    - <element code="I09" name="Interchange Time">
        <value>1529</value>
      </element>
    - <element code="I10" name="Interchange Control Standards Identifier">
        <value>U</value>
      </element>
    - <element code="I11" name="Interchange Control Version Number">
```

An example of the same transaction, an X12 997 Functional Acknowledgment, using standard ANSI format, is shown in Figure 18.

Figure 18 X12 997 Functional Acknowledgment—ANSI Format



```
ISA*00*          *00*          *01*9012345720000  *01*9088877320000
*020129*1529*U*00200*000000001*0*T*:!GS*FA*901234572000*908887732000*20020129*1615*1*T
*004010!ST*997*0001!AK1*FA*1!AK2*997*0001!AK3*BEG*31**1!AK4*12:4*479*1*98382LKA!AK5*A*
1*3*5*1*3!AK9*A*0*433*500006*1*2*5*1*2!SE*8*0001!GE*1*1!IEA*1*000000001!
```

4.7 Possible Differences in Output When Using Pass-Through

If you are using pass-through, the output file contains essentially the same data as the input file.

Certain differences in output, based on variations in acceptable interpretation of the information, are acceptable, provided that the data conforms to the formats specified for the elements. For example:

- If the input file includes a six-digit date, the output file might represent this as an eight-digit value. For example, 040715 in the input file might be represented as 20040705 in the output file.
- The number of trailing zeros after a decimal point might vary. For example, an input value of 10.000 might be represented as 10 in the output file.

The reason these changes occur is that, during pass-through, certain data fields are parsed and stored as Java objects other than strings; for example, Date or Double.

The actual value of all the information must remain the same.

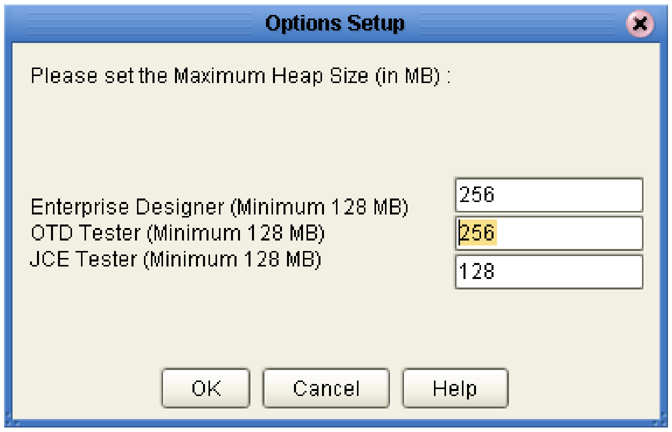
4.8 Limitations of X12 OTDs

4.8.1. Memory Requirements

When using the X12 OTD Library, set the maximum heap size to more than 128MB for Enterprise Designer and the OTD tester; a value of 256MB is recommended. If settings are 128MB or less, and multiple messages are processed simultaneously (such as during FTP batch upload of messages), a NullPointerException can occur.

- To set the heap size in Enterprise Designer: On the **Tools** menu, click **Options**.

Figure 19 Setting the Maximum Heap Size


Options Setup

Please set the Maximum Heap Size (in MB) :

Enterprise Designer (Minimum 128 MB)	256
OTD Tester (Minimum 128 MB)	256
JCE Tester (Minimum 128 MB)	128

4.8.2. Delayed Unmarshaling

For performance reasons, the **unmarshal()** method does not unmarshal the entire message. Instead, it does the following:

- Unmarshals the incoming message at the segment level. In other words, the OTD checks for all relevant segments and reports any missing or extra segments.
- Reports trailing delimiter for elements and composites.

Elements within a segment are not unmarshaled until an element in that segment is accessed in the collaboration using a **get*Xxx*()** method.

4.8.3. Errors and Exceptions

For all X12 OTDs, including the two **envelope** OTDs, if the incoming message cannot be parsed (for example, if the OTD cannot find the ISA segment), then the **unmarshal()** method throws a com.stc.otd.runtime.UnmarshalException.

You can also use the **isUnmarshalComplete()** method to learn whether **unmarshal()** executed without reporting any errors. Successful completion does not guarantee that the OTD instance is free of unmarshal exceptions within segments, however, since elements are not unmarshaled until the first **getElement*Xxxx*()** method of a segment is encountered (see **"Delayed Unmarshaling" on page 50**). Encountering this triggers an automatic background unmarshal of the entire segment, and any problems with the segment will be appended in **allErrors**, **unmarshalErrors**, and **msgValidationResult**. Note that the value returned by **isUnmarshalComplete()** is not influenced by the outcome of the automatic background unmarshal; instead, its value reflects what was set by the explicit invocation of the **unmarshal()** method.

It is an error to use the **set***Xxx* method if bean node *Xxx* is read-only. The system may throw compile exceptions if this is attempted.

If trailing element separators are found inside a segment

In the initial unmarshaling process, the OTD tries to parse the message based on the segment sequence defined in the input metadata (SEF) file. At the same time, however, it also checks for the presence of one or more trailing element separators inside the segment. If found, error arrays such as for unmarshalErrors are populated accordingly. Trailing element separators do not affect data parsing. Immediately after unmarshal() is invoked, therefore, if **isUnmarshalComplete()** returns **true**, then the error arrays for unmarshalErrors either contain no entries or else contain only errors of trailing element separators; if it returns **false**, the error arrays contain errors other than those of trailing element separators.

See **"Bean Nodes for Reporting Errors and Exceptions" on page 53**.

4.8.4. Special Methods for Error Classes

The **toString()** and **marshal()** methods of the following error classes cannot be implemented via the GUI, and have to be hand-coded:

- com.stc.otd.runtime.check.sef.DataError
- com.stc.otd.runtime.check.sef.TSError
- com.stc.otd.runtime.check.sef.FGError
- com.stc.otd.runtime.check.sef.ICError

Also see **"Bean Nodes for Reporting Errors and Exceptions" on page 53**.

Chapter 5

Java Methods for X12 OTDs

Each X12 OTD contains bean nodes and Java methods that extend the functionality of the OTDs. This chapter describes bean nodes and methods, and includes descriptions of the output generated by methods for validation and error message output.

5.1 Bean Nodes

All bean nodes have **get** methods associated with them; in other words, a bean node named *theBeanNode* has a method **get*TheBeanNode*()** to read the current value.

In addition to the **get** methods that all bean nodes have, read/write bean nodes have **set** methods; that is, the method **set*TheBeanNode*()** writes a value.

These methods can be used together or separately. For example, they can allow you to get the X12 delimiters from the input OTD and set them appropriately for the output OTD; or they can also allow you to set the delimiters to the default values.

5.1.1. Read/Write Bean Nodes for Getting and Setting Values

The following bean nodes have both **get** and **set** methods:

- **xmlOutput**—This bean node is of data type **boolean**. The value determines whether the marshal() method generates XML output. The default value is **false**, which causes the marshal() method to generate delimited output. For details, see **"setXmlOutput" on page 63**.
- **maxDataError**—This bean node is of data type **int**. The value defines the number of validation errors to be recorded in the bean node **msgValidationResult**; any errors that occur after the value is reached are not recorded. The default value is **-1**, which causes all errors to be recorded. If the value is set to **0** all errors are suppressed.
- Delimiter bean nodes—The following four bean nodes are of the data type **char**:
  - **segmentTerminator** (See getSegmentTerminator on page 61.)
  - **elementSeparator** (See getElementSeparator **on page 58**.)
  - **subelementSeparator** (See getSubelementSeparator **on page 62**.)
  - **repetitionSeparator** (See getRepetitionSeparator **on page 60**.)

  These four bean nodes are used to set the delimiters for a target X12 message (X12 message generated by the collaboration). The delimiters for an incoming message with interchange envelope are defined in the interchange header; any values that the bean nodes might contain are ignored and overwritten during unmarshal.

5.1.2. Read-Only Bean Nodes for Getting Values

The values of the following bean nodes are set internally by the OTD, usually as a result of some processing. Although these nodes sometimes have corresponding **set** methods, the behavior is not guaranteed and can throw compile exceptions; thus, you should not use the set method for these bean nodes.

- **inputSource**—This bean node is of data type **byte[]**. It holds raw bytes that were unmarshaled to the OTD instance.
- **segmentCount**—This bean node is of data type **int**. It gives the number of segments at the current node. It is available at the following levels:
  - *xxx*_**Outer** (from ISA to IEA segments)
  - *xxx*_**Inner** (from GS to GE segments)
  - *xxx*_<*transactionset*> (from ST to SE segments)

5.1.3. Bean Nodes for Reporting Errors and Exceptions

After **sefimpl.jar** has been imported (see **"To import com.stc.otd.sefimpl.jar" on page 40**), you can use the following bean nodes for reporting errors and exceptions that occur during unmarshal or validation of transactions.

- **allErrors**—This bean node is of data type **java.lang.String[]**. It holds a string version of every exception stored in the other bean nodes listed below: unmarshalErrors, ICValidationResult, FGValidationResult, TSValidationResult, and msgValidationResult.

Note: *A single exception can qualify for inclusion in two or more bean nodes. When this occurs, **allErrors** holds just **one** copy of the exception, and the length of its array is less than the sum of the lengths of the combined individual arrays.*

- **unmarshalErrors**—Data type **com.stc.otd.runtime.check.sef.DataError[]**. This array stores all exceptions that occur when the unmarshal() method is invoked and while accessing elements of a segment for the first time. If this array has any objects in it before a call to **performValidation** has been made, their presence is equivalent to isUnmarshalComplete() returning false.
- **ICValidationResult**—Data type **com.stc.otd.runtime.check.sef.ICError[]**.This array stores all exceptions that occur during the validation of the Interchange envelope.
- **FGValidationResult**—Data type **com.stc.otd.runtime.check.sef.FGError[]**.This array stores all exceptions that occur during validation of the Function Group envelope (in other words, the GS and GE segments).
- **TSValidationResult**—Data type **com.stc.otd.runtime.check.sef.TSError[]**.This array stores all exceptions that occur during validation of the Transaction set (in other words, the ST and SE segments).
- **msgValidationResult**—Data type **com.stc.otd.runtime.check.sef.DataError[]**.This array stores all exceptions that occur during the validation of the message (in other words, the segments between, but not including, the ST and SE segments).

The methods **toString()** and **unmarshal()** are both implemented, allowing the exception to be output in a specific format so that it can be wrapped in an acknowledgment and sent as a response.

Bean node	Data type of the array	Return format
unmarshalErrors	com.stc.otd.runtime.check.sef.DataError	AK3 / AK4 segment of the 997 transaction.
ICValidationResult	com.stc.otd.runtime.check.sef.ICError	TA1 transaction.
FGValidationResult	com.stc.otd.runtime.check.sef.FGError	AK1 / AK9 segment of the 997 transaction.
TSValidationResult	com.stc.otd.runtime.check.sef.TSError	AK2 / AK5 segment of the 997 transaction.
msgValidationResult	com.stc.otd.runtime.check.sef.DataError	AK3 / AK4 segment of the 997 transaction.

5.2 Java Methods

In addition to the bean nodes described in the previous section, and whose methods are explained below, the top node of any x12_40??_[...] OTD in the X12 OTD Library also includes the following Java methods:

- **check** on page 54 (X12 OTDs only, not envelopes)
- **clone** on page 55 (X12 OTDs only, not envelopes)
- **isUnmarshalComplete** on page 55
- **marshalToBytes** on page 56
- **marshalToString** on page 56
- **performValidation** on page 56 (X12 OTDs only, not envelopes)
- **reset** on page 57
- **setDefaultX12Delimiters** on page 58
- **unmarshalFromBytes** on page 65
- **unmarshalFromString** on page 65

check

Description

Performs validation on the OTD unmarshaled from inbound data, and returns message validation result into a String array.

Parameters

None.

Throws

None.

Returns

String[] (the message validation result)

clone

Description

Clones the object itself.

Parameters

None.

Throws

java.lang.CloneNotSupportedException (if clone not implemented)

Returns

java.lang.Object (the cloned object)

isUnmarshalComplete

Description

Flag for whether or not the unmarshaling (parsing) has completed successfully.

Parameters

None.

Throws

None.

Returns

boolean (whether or not the initial explicit call to **unmarshal** completed successfully). For caveats and limitations, see **"Delayed Unmarshaling" on page 50** and **"Errors and Exceptions" on page 50**.

marshal

Description

Marshals (serializes, renders) the internal data tree into an output stream.

Parameters

out - **com.stc.otd.runtime.OtdOutputStream** (the output, as an **OtdOutputStream** object)

Throws

java.io.IOException (for output problems)

com.stc.otd.runtime.MarshalException (for an inconsistent internal tree)

Returns

None.

marshalToBytes

Description

Marshals (serializes, renders) the internal data tree into a byte array.

Parameters

in - **byte[]** (the input, as a byte array)

Throws

java.io.IOException (for output problems)

com.stc.otd.runtime.MarshalException (for an inconsistent internal tree)

Returns

byte[] (serialized byte array)

marshalToString

Description

Marshals (serializes, renders) the internal data tree into a String.

Parameters

None.

Throws

java.io.IOException (for input problems)

com.stc.otd.runtime.MarshalException (for an inconsistent internal tree)

Returns

java.lang.String (the serialized String)

performValidation

Description

Validates the OTD content immediately after unmarshaling.

Syntax

```
public void performValidation()
```

Parameters

None.

Constants

None.

Returns

None. If the OTD content is found to be invalid, the appropriate error bean nodes are populated; see **"Bean Nodes for Reporting Errors and Exceptions" on page 53**.

Throws

None.

Examples

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
myOTD.performValidation();
```

Notes

For an example of using **performValidation()** in a collaboration, see **Creating a Collaboration Rule to Validate an X12 OTD** on page 45.

reset

Description

Clears out any data and resources held by this OTD instance.

Parameters

None.

Throws

None.

Returns

None.

setDefaultX12Delimiters

Description

Sets the default X12 delimiters, such as:

~	segment terminator
*	element separator
:	subelement separator
+	repetition separator

Syntax

```
public void setDefaultX12Delimiters()
```

Parameters

None.

Constants

None.

Returns

void (none).

Throws

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
myOTD.setDefaultX12Delimiters();
```

getElementSeparator

Description

Gets the elementSeparator character.

Syntax

```
public char getElementSeparator()
```

Parameters

None.

Constants

None.

Returns

char

Returns the element separator character.

Throws

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char elmSep=myOTD.getElementSeparator();
```

setElementSeparator

Description

Sets the elementSeparator character.

Syntax

```
public void setElementSeparator(char c);
```

Parameters

Name	Type	Description
c	char	The character to be set as the element separator.

Constants

None.

Returns

void (none).

Throws

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char c='+';
myOTD.setElementSeparator(c);
```

getRepetitionSeparator

Description

Gets the RepetitionSeparator character. Valid only for X12 version 4020 and later.

Syntax

```
public char getRepetitionSeparator()
```

Parameters

None.

Constants

None.

Returns

char
Returns the getRepetitionSeparator character.

Throws

None.

Example

```
x12_4020.x12_4020_850_PurcOrde_Outer myOTD=new x12_4020.x12_4020_850_
PurcOrde_Outer();
......
......
char repSep=myOTD.getRepetitionSeparator();
```

setRepetitionSeparator

Description

Sets the RepetitionSeparator character. Valid only for X12 version 4020 and later.

Syntax

```
public void setRepetitionSeparator(char c)
```

Parameters

Name	Type	Description
c	char	The character to be set as the repetition separator.

Constants

None.

Returns

void (none).

Throws

None.

Example

```
x12_4030.x12_4030_850_PurcOrde_Outer myOTD=new x12_4030.x12_4030_850_
PurcOrde_Outer();
......
......
char c='*';
myOTD.setRepetitionSeparator(c);
```

getSegmentTerminator

Description

Gets the segment terminator character.

Syntax

```
public char getSegmentTerminator()
```

Parameters

None.

Constants

None.

Returns

char (the segment terminator character).

Throws

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char segTerm=myOTD.getSegmentTerminator();
```

setSegmentTerminator

Description

Sets the segmentTerminator character.

Syntax

```
public void setSegmentTerminator(char c)
```

Parameters

Name	Type	Description
c	char	The character to be set as the segment terminator.

Constants

None.

Returns

void (none).

Throws

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char c='~';
myOTD.setSegmentTerminator(c);
```

getSubelementSeparator

Description

Gets the subelementSeparator character.

Syntax

```
public char getSubelementSeparator()
```

Parameters

None.

Constants

None.

Returns

char
Returns the getSubelement character.

Throws

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char subeleSep=myOTD.getSubelementSeparator();
```

setSubelementSeparator

Description

Sets the SubelementSeparator character.

Syntax

```
public void setSubelementSeparator(char c)
```

Parameters

Name	Type	Description
c	char	The character to be set as the subelement separator.

Constants

None.

Returns

void (none).

Throws

None.

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
char c=':';
myOTD.setSubelementSeparator(c);
```

setXmlOutput

Description

When used with the parameter set to **true**, this method causes the X12 OTD involved to output XML.

When used with the parameter set to **false**, this method causes the X12 OTD to output ANSI (which is the default output if this method is not used at all).

Use this method when the X12 OTD is set to automatic output (the default). If the collaboration is set to manual output, use marshal (boolean) to achieve the same result.

Syntax

```
public void setXMLOutput(boolean isXML)
```

Parameters

Name	Type	Description
isXML	boolean	If true, the X12 is output in XML format. If false, output is standard ANSI X12.

Constants

None.

Returns

void (none).

Throws

None

Example

```
x12_4010.x12_4010_850_PurcOrde_Outer myOTD=new x12_4010.x12_4010_850_
PurcOrde_Outer();
......
......
myOTD.setXMLOutput(true);
```

unmarshal

Description

Unmarshals (deserializes, parses) the given input into an internal data tree.

Parameters

in - **com.stc.otd.runtime.OtdInputStream** (the input, as an OtdInputStream object)

Throws

java.io.IOException (for output problems)

com.stc.otd.runtime.UnmarshalException (for a lexical or other mismatch)

Returns

None.

Notes

Caveats and limitations are discussed in **"Delayed Unmarshaling" on page 50**, **"Errors and Exceptions" on page 50**, and **"Special Methods for Error Classes" on page 51**.

unmarshalFromBytes

Description

Unmarshals (deserializes, parses) the given input byte array into an internal data tree.

Parameters

in - **byte[]** (the input, as a byte array)

Throws

java.io.IOException (for input problems)

com.stc.otd.runtime.UnmarshalException (for an inconsistent internal tree)

Returns

None.

unmarshalFromString

Description

Unmarshals (deserializes, parses) the given input string into an internal data tree.

Parameters

in - **java.lang.String** (the input, as a String)

Throws

java.io.IOException (for input problems)

com.stc.otd.runtime.UnmarshalException (for an inconsistent internal tree; typically occurs if the OTD cannot recognize the incoming message as X12)

Returns

None.

Index