SeeBeyond™ eBusiness Integration Suite

X12 ETD Library User's Guide

Release 4.5.3

Version 20021101152218.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

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

Note: *If you are upgrading from version 4.5.1 or earlier and are using the Java ETDs, you must recompile all Collaborations that use these ETDs after installing the new version.*

1.1 Overview

Each of the e*Gate Event Type Definition (ETD) libraries contains sets of pre-built structures for industry-standard formats. e*Gate ETD files are message format definitions in two formats:

- Java
- Monk

The X12 ETD library is a feature of the SeeBeyond eBusiness Integration Suite, and contains message definitions for X12 messages. This document gives a brief overview of X12 and the X12 message structures provided with e*Gate, and provides information on installing and using the X12 ETD libraries.

1.2 Intended Reader

The reader of this guide is presumed to be a developer or system administrator with responsibility for maintaining the e*Gate™ system or the SeeBeyond™ eBusiness Integration Suite, to have familiarity with Windows 2000 and Windows NT operations and administration, and to be thoroughly familiar with Microsoft Windows graphical user interfaces.

1.3 Compatible Systems

The e*Gate X12 ETD Library is available on the following platforms:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4.0 SP61
- Compaq Tru64 Version 4.0F and 5.0A
- Solaris 2.6, 7, and 8
- HP-UX 11.0 and 11i
- AIX 4.3.3
- Red Hat Linux 6.2 (Intel version)

It also supports the following operating systems in Japanese:

- Windows 2000, Windows 2000 SP1, and Windows 2000 SP2
- Windows NT 4.0 SP6a
- Solaris 2.6, 7, and 8
- HP-UX 11.0

Note: ***Windows Systems****—The SeeBeyond eBusiness Integration Suite is fully compliant with Windows 2000 and Windows NT 4.0 platforms. When this document references Windows, such statements apply to both Windows platforms.*

Note: ***UNIX Systems****—This guide uses the backslash ("\") as the separator within path names. If you are working on a UNIX system, make the appropriate substitutions.*

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 e*Gate to help you quickly create the structures you need for various 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 the message structures that comprise the X12 ETD 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 e*Gate, a message structure is called an Event Type Definition (ETD). Each message structure (ETD) 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. Installation of X12 templates 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.

e*Xchange Partner Manager uses e*Gate Event Type Definitions, which are 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 Version 4010 as an example in illustrating how to install and work with X12 templates.

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 13**.

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 ETD Library are the same as those recommended by the industry-specific implementation guides. These delimiters are shown in Table 1.

Table 1 Default Delimiters in X12 ETD 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 e*Xchange Partner Manager, delimiters are specified at the B2B protocol level. The delimiters you define are applied to all transaction types.

If you do not specify delimiters, e*Xchange expects the default delimiters as shown in Table 1.

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

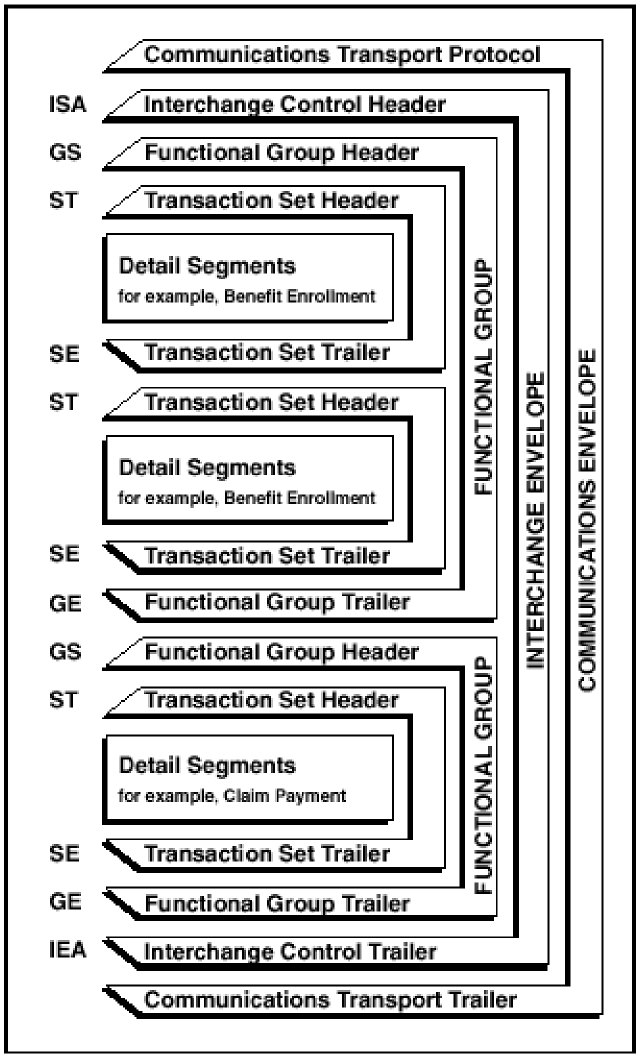
The rules applying to the structure of an X12 envelope are very strict, to ensure the integrity of the data and the efficiency of the information exchange.

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 Segment Table

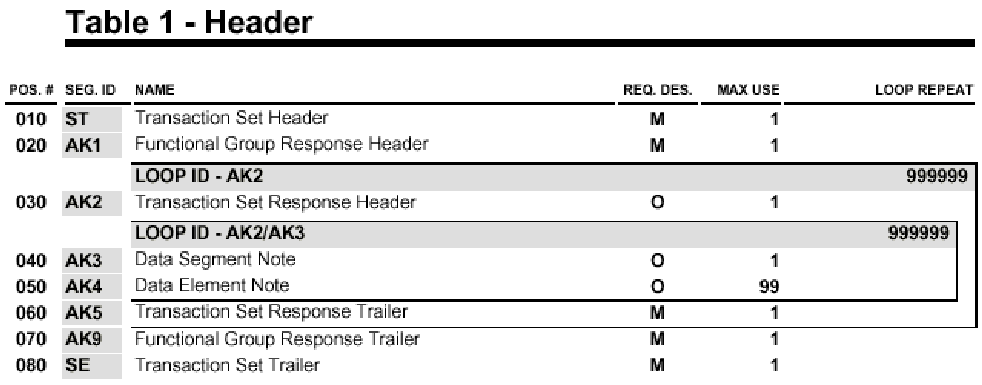


Table 1 - Header

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 Monk ETD Editor in e*Gate.

Figure 3 X12 997 Viewed in Monk ETD Editor

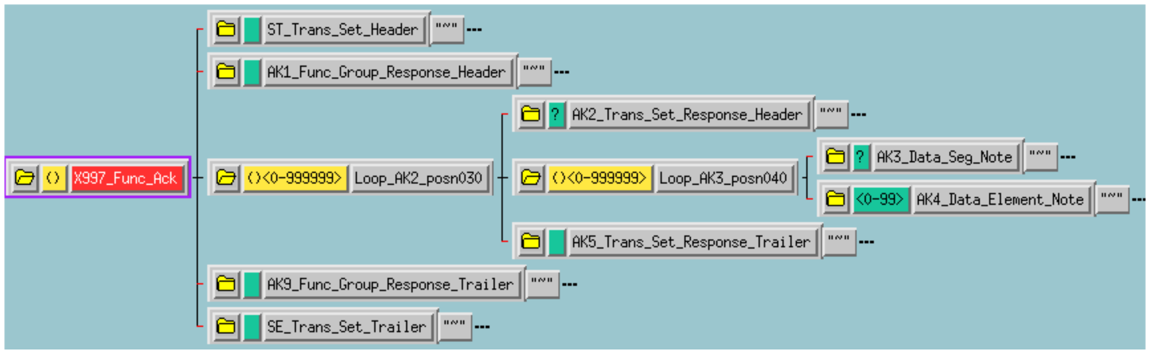
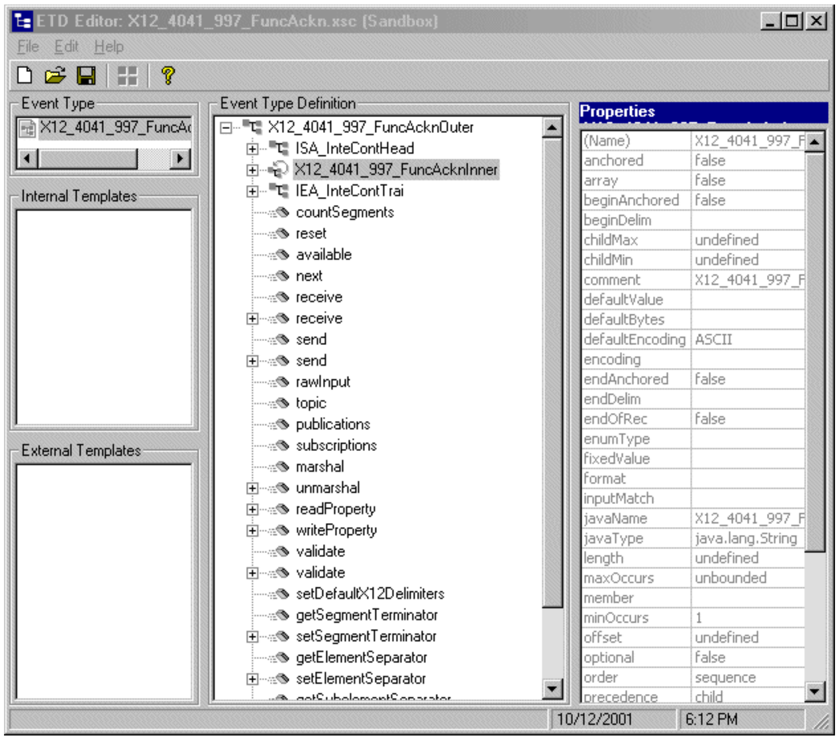


Figure 4 shows the same transaction as viewed in the Java ETD Editor.

Figure 4 X12 997 Viewed in Java ETD Editor



```
X12_4041_997_FuncAcknOuter
    ISA_InteContHead
    X12_4041_997_FuncAcknInner
    IEA_InteContTrai
    countSegments
    reset
    available
    next
    receive
    receive
    send
    send
    rawInput
    topic
    publications
    subscriptions
    marshal
    unmarshal
    readProperty
    writeProperty
    validate
    validate
    setDefaultX12Delimiters
    getSegmentTerminator
    setSegmentTerminator
    getElementSeparator
    setElementSeparator
```

(Name)	X12_4041_997_F
anchored	false
array	false
beginAnchored	false
beginDelim	
childMax	undefined
childMin	undefined
comment	X12_4041_997_F
defaultValue	
defaultBytes	
defaultEncoding	ASCII
encoding	
endAnchored	false
endDelim	
endOfRec	false
enumType	
fixedValue	
format	
inputMatch	
javaName	X12_4041_997_F
javaType	java.lang.String
length	undefined
maxOccurs	unbounded
member	
minOccurs	1
offset	undefined
optional	false
order	sequence
precedence	child

10/12/2001 6:12 PM

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 comprised 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, ZIP 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 5 Example of a Transaction Set Header (ST)

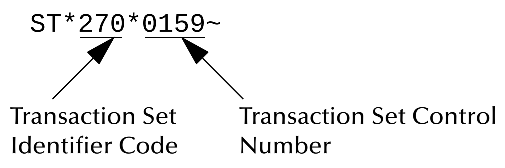
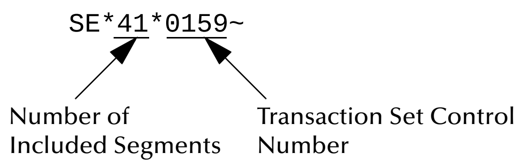

ST*270*0159~
Transaction Set Identifier Code
Transaction Set Control Number

Figure 6 Example of a Transaction Set Trailer (SE)



2.3.2. Functional Group (GS/GE)

A functional group is comprised 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 comprise a specific functional group are identified by this functional ID code.

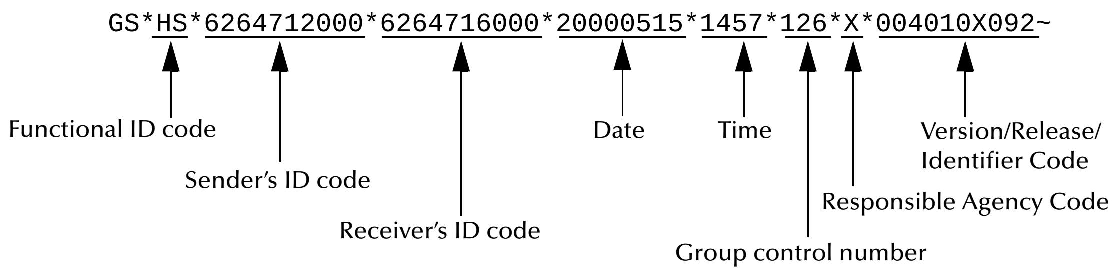
The functional group header (GS) segment contains the following information:

- 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:

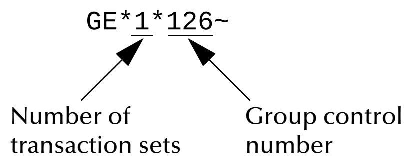
- Number of transaction sets included
- Group control number (originated and maintained by the sender)

Figure 7 Example of a Functional Group Header (GS)



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

Figure 8 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.

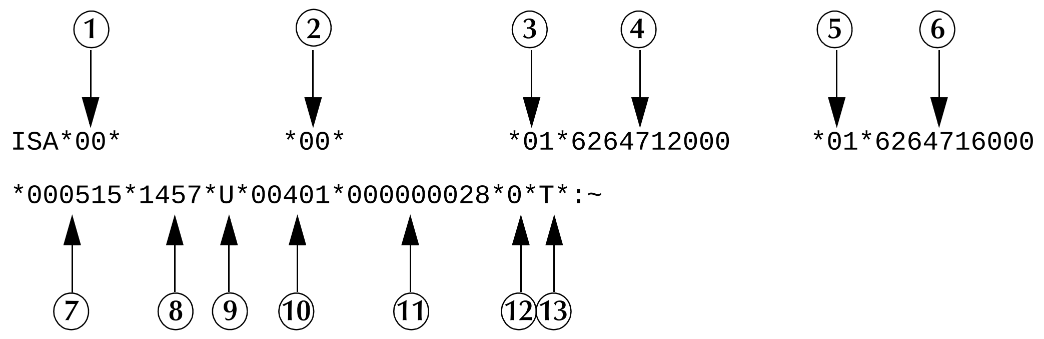
As well as enveloping one or more functional groups, the interchange header and trailer segments include the following information:

- 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

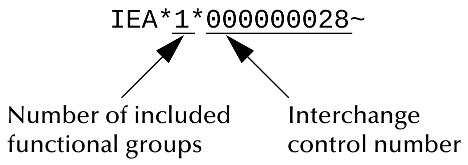
Figure 9 Example of an Interchange Header (ISA)


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 9:

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 10 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. The e*Xchange Partner Manager 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 e*Xchange Partner Manager 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, e*Xchange Partner Manager 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 2. This table lists the transactions that comprise X12 version 4010.

Table 2 Transactions Included in X12 Version 4010

Number	Title
101	Name and Address Lists
104	Air Shipment Information

Table 2 Transactions Included in X12 Version 4010 (Continued)

Number	Title
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
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

Table 2 Transactions Included in X12 Version 4010 (Continued)

Number	Title
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
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

Table 2 Transactions Included in X12 Version 4010 (Continued)

Number	Title
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
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

Table 2 Transactions Included in X12 Version 4010 (Continued)

Number	Title
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
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

Table 2 Transactions Included in X12 Version 4010 (Continued)

Number	Title
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
486	Rate Docket Expiration
490	Rate Group Definition
492	Miscellaneous Rates
494	Rail Scale Rates
500	Medical Event Reporting
501	Vendor Performance Review

Table 2 Transactions Included in X12 Version 4010 (Continued)

Number	Title
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
821	Financial Information Reporting
822	Account Analysis
823	Lockbox
824	Application Advice
826	Tax Information Exchange
827	Financial Return Notice

Table 2 Transactions Included in X12 Version 4010 (Continued)

Number	Title
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
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

Table 2 Transactions Included in X12 Version 4010 (Continued)

Number	Title
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
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

Table 2 Transactions Included in X12 Version 4010 (Continued)

Number	Title
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
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. 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 3. The table describes each term, and lists its language analogy along with its associated e*Gate Collaboration scripts.

Table 3 Key Parts of EDI Processing

Term	Description	Language Analogy	e*Gate Collaboration Scripts
structures	format, segments, loops	syntax	ETD files or structures
validations	data contents “edit” rules	semantics	validation scripts
translations (also called mapping)	reformatting or conversion	translation	translation scripts
enveloping	header and trailer segments	envelopes	part of translation
acks	acknowledgments	return receipt	e*Way scripts

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

2.8.1. Structures

The Event Type Definition library for X12 includes pre-built ETDs for all supported X12 versions.

To customize a message structure, use the e*Gate GUI message structure editor, which is called the ETD (Event Type Definition) Editor.

2.8.2. Validations, Translations, Enveloping, Acknowledgments

e*Gate does not include any pre-built validations, transformations, enveloping, or acknowledgments. These scripts can be built in either the Monk or Java versions of the Collaboration Rules Editor graphical user interface (GUI). These GUIs provide a user-friendly drag-and-drop front end for creating Monk or Java scripts.

However, installation of the e*Xchange Partner Manager includes a set of custom Monk validations for HIPAA transactions.

Note: *In e*Gate, translations are called Collaborations.*

2.8.3. X12 Acknowledgments in e*Xchange Partner Manager

All X12 acknowledgments are automatically handled in e*Xchange Partner Manager. This allows you to configure the transaction set, if any, that is expected as the acknowledgment. e*Xchange Partner Manager can automatically create any type of X12 acknowledgment, including TA1, 997, 824, and transaction-specific acknowledgments.

For more information on X12 acknowledgment types, refer to **"Acknowledgment Types" on page 32**.

2.8.4. 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 X12, visit the following Web sites:

- For X12 standard:

  http://www.disa.org

- For Implementation Guides: Washington Publishing Company at

  http://www.wpc-edi.com

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

X12 Template Installation

This chapter provides information on the installation procedure for the SeeBeyond X12 ETD library template files and shows the resulting directory structure for the templates. It includes general installation information and installation instructions.

Note: *If you are upgrading from version 4.5.1 or earlier and are using the Java ETDs, you must recompile all Collaborations that use these ETDs after installing the new version.*

The X12 ETD library includes templates for the following X12 versions.

Table 4 X12 Versions Supported

▪ 2000 (Java only)	▪ 3042	▪ 4020
▪ 2001	▪ 3050	▪ 4021
▪ 2002	▪ 3051	▪ 4022
▪ 2003	▪ 3052	▪ 4030
▪ 2040	▪ 3060	▪ 4031
▪ 3010	▪ 3061	▪ 4032
▪ 3020	▪ 3062	▪ 4040
▪ 3022	▪ 3070	▪ 4041 (Java only)
▪ 3030	▪ 3071	▪ 4042
▪ 3031	▪ 3072 (Java only)	▪ 4050
▪ 3032	▪ 4010	▪ 4051
▪ 3040	▪ 4011	
▪ 3041	▪ 4012	

Some additional points to note:

- The ETDs only accept messages with all the envelope segment information.
- 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 ETD Library is installed, there is a separate subdirectory for each version of X12; for example, 3020 or 4010. Within that subdirectory all the files that comprise the ETD library for that version are stored.

If the X12 version you are installing includes both Monk and Java files, there is a separate location for each set of files.

For more information on the folder structure for the e*Gate X12 ETD Library, refer to **"X12 Folder Structure Created by Installation" on page 40**.

3.2 Installation Procedure

This section explains how to install the X12 template files.

The X12 ETD Library is available on e*Gate installation CDs #11, 12, and 13. The breakdown is as shown in Table 5.

Table 5 X12 ETD Library CDs

CD #11	CD #12	CD #13
▪ 2000 (Java only)	▪ 3072 (Java only)	▪ 4041 (Java only)
▪ 2001	▪ 3070	▪ 4030
▪ 2002	▪ 3071	▪ 4031
▪ 2003	▪ 4010	▪ 4032
▪ 2040	▪ 4011	▪ 4040
▪ 3010	▪ 4012	▪ 4042
▪ 3020	▪ 4020	▪ 4050
▪ 3022	▪ 4021	▪ 4051
▪ 3030	▪ 4022	
▪ 3031		
▪ 3032		
▪ 3040		
▪ 3041		
▪ 3042		
▪ 3050		
▪ 3051		
▪ 3052		
▪ 3060		
▪ 3061		
▪ 3062		

Before you begin:

- You must have Administrator privileges to install back-end components such as the X12 templates.
- Exit all Windows programs, including any anti-virus applications.
- Verify your e*Gate registry host name, schema name, control broker logical name, and the administrator user name and password.
- Define which X12 templates you want to install.

To install the X12 templates on Windows

1. Log in on the workstation on which you want to install the X12 templates.

2 Insert the appropriate X12 installation CD into the CD-ROM drive. For more information, refer to Table 5.

If Autorun is enabled, the setup program automatically starts. Otherwise:

- On the task bar, click the **Start** button, and then click **Run**.
- In the **Open** field, type **D:\setup\setup.exe** where **D:** is your CD-ROM drive.

3 Follow the installation instructions until you come to the **Please choose the product to install** dialog box.

4 Select **e*Gate Integrator**, and then click **Next**.

5 Follow the on-screen instructions until you come to the second **Please choose the product to install** dialog box.

6 Select **Add-ons**, and then click **Next**.

7 Follow the on-screen instructions until you come to the **Select Components** screen.

8 Highlight (but do not check) **ETD Libraries**, and then click the **Change** button.

The **Select Sub-components** dialog box appears.

9 Select the X12 version or versions for which you need the template files.

10 Click **Continue** to return to the **Select Components** dialog box, and then click **Next**.

11 Follow the rest of the on-screen instructions to install the X12 templates.

Note: *Installing the X12 ETD library might take quite some time depending on the speed of the machine and the number of templates you install.*

For more information about e*Gate installation, see the *e*Gate Integrator Installation Guide*.

Note: *Do not change the default directory location for the X12 template files.*

To install the X12 templates on UNIX

To install the X12 templates on UNIX, you must have the e*Gate Registry host set up on UNIX also.

The procedure is essentially similar to Windows installation; however, you can only select one library at a time.

1 Follow the steps for the standard e*Gate installation on UNIX.

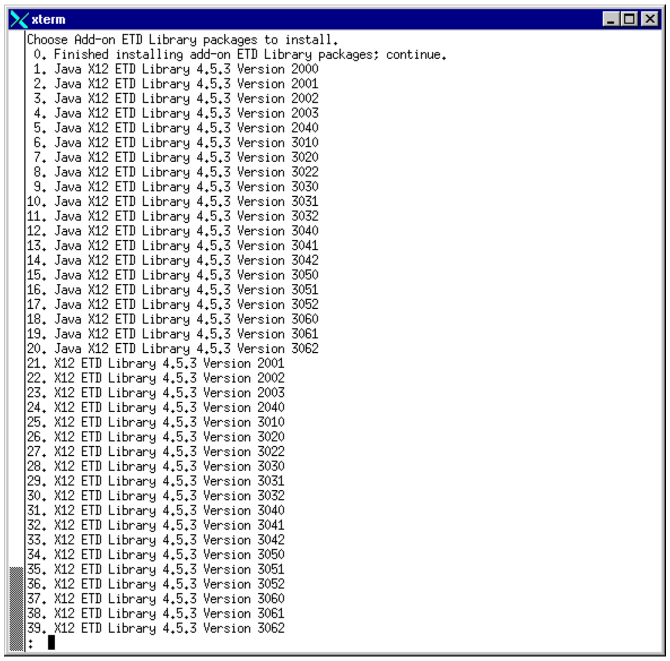
For more information, refer to the *e*Gate Integrator Installation Guide*.

2 At the prompt **Choose e*Gate Add-on Application**, enter the number assigned for the X12 Library for the version you want (scroll down the list to check the number).

An example of this list when installing the X12 ETD Library version 4.5.3 with e*Gate version 4.5.2 is shown in Figure 11.

Note: *With the UNIX installation, for e*Gate 4.5.2, if the library selection specifies Java—for example, Java X12 ETD Library 4.5.3 Version 3062—only the Java ETDs are installed. If the selection does not specify Java—for example, X12 ETD Library 4.5.3 Version 3062—both the Monk and Java ETDs are installed.*

Figure 11 UNIX Installation



```
Choose Add-on ETD Library packages to install.
 0. Finished installing add-on ETD Library packages; continue.
 1. Java X12 ETD Library 4.5.3 Version 2000
 2. Java X12 ETD Library 4.5.3 Version 2001
 3. Java X12 ETD Library 4.5.3 Version 2002
 4. Java X12 ETD Library 4.5.3 Version 2003
 5. Java X12 ETD Library 4.5.3 Version 2040
 6. Java X12 ETD Library 4.5.3 Version 3010
 7. Java X12 ETD Library 4.5.3 Version 3020
 8. Java X12 ETD Library 4.5.3 Version 3022
 9. Java X12 ETD Library 4.5.3 Version 3030
10. Java X12 ETD Library 4.5.3 Version 3031
11. Java X12 ETD Library 4.5.3 Version 3032
12. Java X12 ETD Library 4.5.3 Version 3040
13. Java X12 ETD Library 4.5.3 Version 3041
14. Java X12 ETD Library 4.5.3 Version 3042
15. Java X12 ETD Library 4.5.3 Version 3050
16. Java X12 ETD Library 4.5.3 Version 3051
17. Java X12 ETD Library 4.5.3 Version 3052
18. Java X12 ETD Library 4.5.3 Version 3060
19. Java X12 ETD Library 4.5.3 Version 3061
20. Java X12 ETD Library 4.5.3 Version 3062
21. X12 ETD Library 4.5.3 Version 2001
22. X12 ETD Library 4.5.3 Version 2002
23. X12 ETD Library 4.5.3 Version 2003
24. X12 ETD Library 4.5.3 Version 2040
25. X12 ETD Library 4.5.3 Version 3010
26. X12 ETD Library 4.5.3 Version 3020
27. X12 ETD Library 4.5.3 Version 3022
28. X12 ETD Library 4.5.3 Version 3030
29. X12 ETD Library 4.5.3 Version 3031
30. X12 ETD Library 4.5.3 Version 3032
31. X12 ETD Library 4.5.3 Version 3040
32. X12 ETD Library 4.5.3 Version 3041
33. X12 ETD Library 4.5.3 Version 3042
34. X12 ETD Library 4.5.3 Version 3050
35. X12 ETD Library 4.5.3 Version 3051
36. X12 ETD Library 4.5.3 Version 3052
37. X12 ETD Library 4.5.3 Version 3060
38. X12 ETD Library 4.5.3 Version 3061
39. X12 ETD Library 4.5.3 Version 3062
:
```

3 Enter the installation path, or press **Enter** to accept the default path (recommended).

4 Enter the hostname of the registry server (UNIX host).

The Monk files are installed and committed to the e*Gate Registry. When done, the following prompt is displayed:

```
Press Enter/Return to continue:
```

5 Press Enter or Return.

The Java files are installed and committed to the e*Gate Registry. When done, the prompt is again displayed:

```
Press Enter/Return to continue:
```

6 To install more than one X12 version, repeat steps 2 through 4 as needed. If you are done, press 0 to return to the previous menu.

3.3 X12 Files and Folders

This section outlines the folder structure created on your hard drive as a result of installation of the X12 templates, and the files copied into those folders.

3.3.1. X12 Folder Structure Created by Installation

By default, installation places the X12 templates in the locations shown in Table 6.

Table 6 X12 Template Locations

These files…	Are installed in this location…
Java	\<eGate>\server\registry\repository\default\etd\templates\x12
Monk	\<eGate>\server\registry\repository\default\monk_scripts\templates\x12

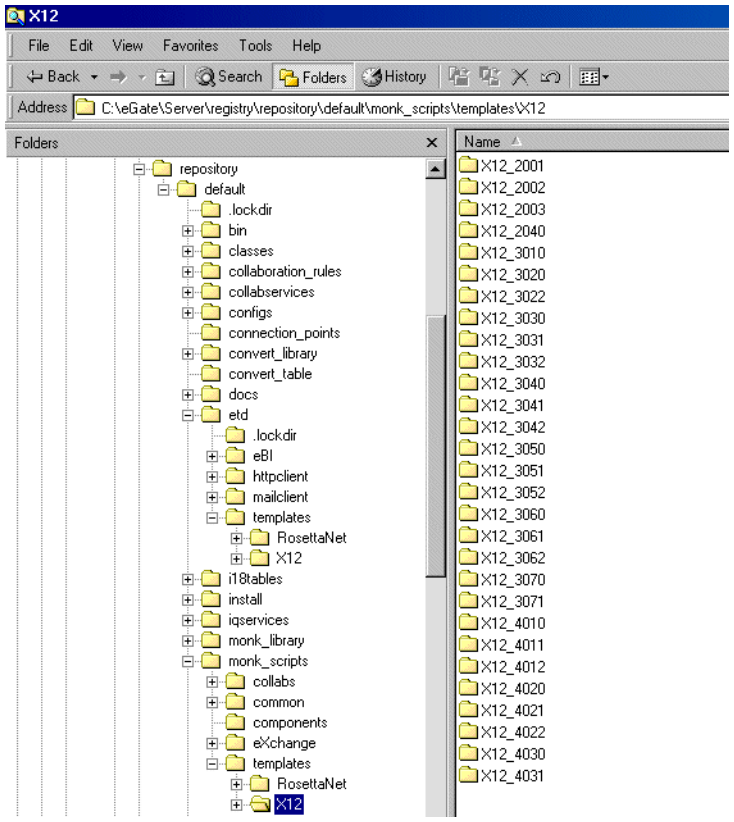
Installation commits the templates to the **default** schema on the Registry Host that you specified during the installation process.

Within the template directory, there is a subdirectory for each version of X12 that was selected during installation. Within each subdirectory, there is an **.ssc** file for each X12 transaction.

Under each version directory there is a **templates** subdirectory for segment templates. All messages are in the version directory. All segments are in the template directory, as illustrated in **Figure 14 on page 42**.

Figure 12 shows an example of the directory structure for the X12 templates when all versions have been installed.

Figure 12 X12 Version Folders



```
C:\eGate\Server\registry\repository\default\monk_scripts\templates\X12
```

```
repository
  default
    .lockdir
    bin
    classes
    collaboration_rules
    collabservices
    configs
    connection_points
    convert_library
    convert_table
    docs
    etd
      .lockdir
      eBI
      httpclient
      mailclient
      templates
        RosettaNet
        X12
    i18tables
    install
    iqservices
    monk_library
    monk_scripts
      collabs
      common
      components
      eXchange
      templates
        RosettaNet
        X12
```

```
X12_2001
X12_2002
X12_2003
X12_2040
X12_3010
X12_3020
X12_3022
X12_3030
X12_3031
X12_3032
X12_3040
X12_3041
X12_3042
X12_3050
X12_3051
X12_3052
X12_3060
X12_3061
X12_3062
X12_3070
X12_3071
X12_4010
X12_4011
X12_4012
X12_4020
X12_4021
X12_4022
X12_4030
X12_4031
```

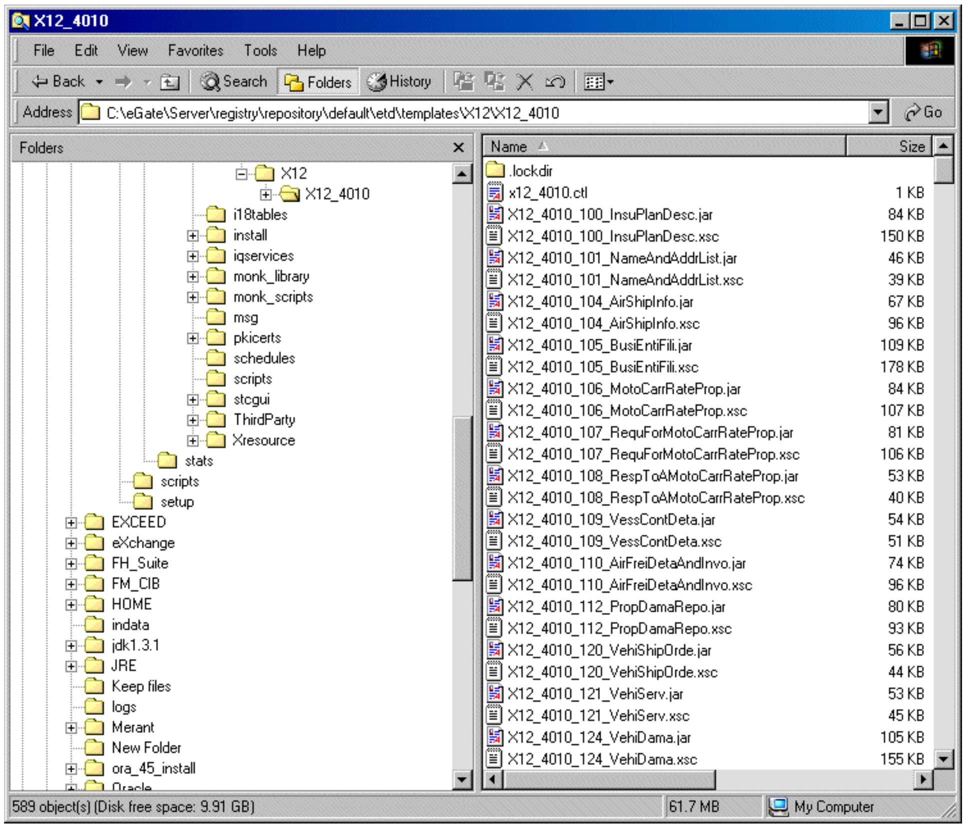
UNIX installation places the files in the same folder structure shown above.

3.3.2. X12 Files

Since there is a file for each X12 transaction and a file for each segment, installation of each version of X12 includes a large number of files.

Figure 13 shows some of the files that are installed for a specific version of X12 (Version 4010) Java.

Figure 13 Transaction Set Structures for X12 Version 4010 (Java)



Address: C:\eGate\Server\registry\repository\default\etd\templates\X12\X12_4010

Name	Size
.lockdir	
x12_4010.ctl	1 KB
X12_4010_100_InsuPlanDesc.jar	84 KB
X12_4010_100_InsuPlanDesc.xsc	150 KB
X12_4010_101_NameAndAddrList.jar	46 KB
X12_4010_101_NameAndAddrList.xsc	39 KB
X12_4010_104_AirShipInfo.jar	67 KB
X12_4010_104_AirShipInfo.xsc	96 KB
X12_4010_105_BusiEntiFili.jar	109 KB
X12_4010_105_BusiEntiFili.xsc	178 KB
X12_4010_106_MotoCarrRateProp.jar	84 KB
X12_4010_106_MotoCarrRateProp.xsc	107 KB
X12_4010_107_RequForMotoCarrRateProp.jar	81 KB
X12_4010_107_RequForMotoCarrRateProp.xsc	106 KB
X12_4010_108_RespToAMotoCarrRateProp.jar	53 KB
X12_4010_108_RespToAMotoCarrRateProp.xsc	40 KB
X12_4010_109_VessContDeta.jar	54 KB
X12_4010_109_VessContDeta.xsc	51 KB
X12_4010_110_AirFreiDetaAndInvo.jar	74 KB
X12_4010_110_AirFreiDetaAndInvo.xsc	96 KB
X12_4010_112_PropDamaRepo.jar	80 KB
X12_4010_112_PropDamaRepo.xsc	93 KB
X12_4010_120_VehiShipOrde.jar	56 KB
X12_4010_120_VehiShipOrde.xsc	44 KB
X12_4010_121_VehiServ.jar	53 KB
X12_4010_121_VehiServ.xsc	45 KB
X12_4010_124_VehiDama.jar	105 KB
X12_4010_124_VehiDama.xsc	155 KB

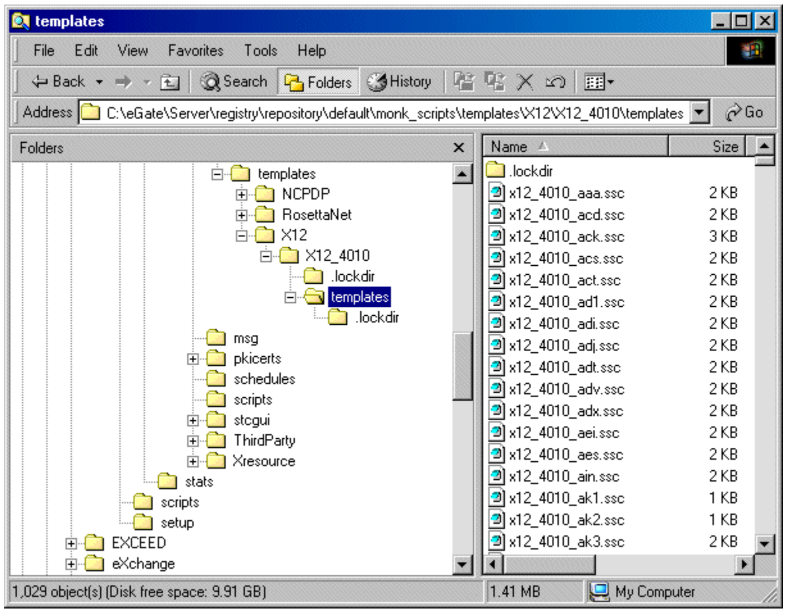
589 object(s) (Disk free space: 9.91 GB) 61.7 MB My Computer

There is an Event Type Definition file for each of the individual transactions that comprise the selected version of the standard. These files have the extension **xsc** for Java, **ssc** for Monk.

Note: *When building Collaboration Rules scripts with Java ETDs, if there is data mapped to a field in a Java template and there are optional fields on the same level with no data mapped to them, the output includes delimiters for the optional fields.*

Figure 14 shows the files in the Templates subdirectory for X12 Version 4010 Monk.

Figure 14 Segment Template Files for X12 Version 4010 Monk



Address: C:\eGate\Server\registry\repository\default\monk_scripts\templates\X12\X12_4010\templates

Name	Size
.lockdir	
x12_4010_aaa.ssc	2 KB
x12_4010_acd.ssc	2 KB
x12_4010_ack.ssc	3 KB
x12_4010_acs.ssc	2 KB
x12_4010_act.ssc	2 KB
x12_4010_ad1.ssc	2 KB
x12_4010_adi.ssc	2 KB
x12_4010_adj.ssc	2 KB
x12_4010_adt.ssc	2 KB
x12_4010_adv.ssc	2 KB
x12_4010_adx.ssc	2 KB
x12_4010_aei.ssc	2 KB
x12_4010_aes.ssc	2 KB
x12_4010_ain.ssc	2 KB
x12_4010_ak1.ssc	1 KB
x12_4010_ak2.ssc	1 KB
x12_4010_ak3.ssc	2 KB

1,029 object(s) (Disk free space: 9.91 GB)   1.41 MB

3.3.3. File Names

The file names for the X12 templates are designed to assist you in quickly locating the file you want.

Transaction Template File Names—Monk

The file name for each transaction template is comprised of the same set of elements in the same sequence. The file names are constructed as follows:

- X12_ (name of standard followed by underscore)
- 4010_ (name of version followed by underscore)
- 997 (name of message)
- .ssc (file extension)

Examples:

- The Monk file name for a 270, Eligibility Coverage or Benefit Inquiry, for version 4010 is **X12_4010_270.ssc**.
- The Monk file name for a 855, Purchase Order Acknowledgment, in version 3020 is **X12_3020_855.ssc**.

Segment Template File Names—Monk

Segment templates are located in the **\templates** subdirectory to the main directory for the X12 version.

The file name for each segment template is comprised of the same set of elements in the same sequence. Where the transaction segment template file names include the transaction number, the segment files include the segment ID.

The file names are constructed as follows:

- X12_ (name of standard followed by underscore)
- 4010_ (name of version followed by underscore)
- aaa (segment ID)
- .ssc (file extension)

Examples:

- The Monk file name for segment AAA, Request Validation, for version 4010 is **X12_4010_aaa.ssc**.
- The Monk file name for segment AK5, Transaction Set Response Trailer, in version 3020 is **X12_3020_ak5.ssc**.

Transaction Template File Names—Java

The file name for each transaction template is comprised of the same set of elements in the same sequence. The file names are constructed as follows:

- X12_ (name of standard followed by underscore)
- 4010_ (name of version followed by underscore)
- 997 (name of message)
- Abbreviation for the transaction name
- .xsc (file extension)

Examples:

- The Java file name for a 270, Eligibility Coverage or Benefit Inquiry, for version 4010 is **X12_4010_270_EligCoveOrBeneInqu.xsc**.
- The Java file name for an 855, Purchase Order Acknowledgment, in version 3020 is **X12_3020_855_PurcOrdeAckn.xsc**.

3.4 Customizing a Monk Message Structure

If you need to customize a Monk message structure, you can edit it in the e*Gate ETD Editor.

Note: *You cannot customize the Java message structures.*

To edit a message structure in e*Gate

1 With the **ETD Editor** window open, click **Open** on the Toolbar.

2 When the **Open Event Type Definition** dialog box opens, click **Templates**.

3 Navigate to the X12 directory.

4 Select a version directory (such as **X12_4010)** and click **Open**.

5 Select a message (such as **X12_4010_997.ssc**) and click **Open**.

The message opens in both the **Template Library** and the **Workspace** panes.

6 From the **Edit** menu, choose **Find**.

7 In the **Find** text box, type the name of the segment you want to find within the message, and click **Next**.

If the segment is present, it is highlighted.

8 To delete the segment, click either **Delete** or **Cut** on the Toolbar.

Note: *If you want to do further editing of the ETD, refer to the ETD Editor online Help system for instructions.*

9 When done, click **Save** to save your work.

Chapter 4

Working With the Java X12 ETDs

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

To test that your data is being mapped correctly by the ETD, and that the data is valid based on definitions and business rules, you can run validation within the Collaboration Rules component.

This chapter also provides information on using the custom Java methods provided within the ETDs, and other general information about using the X12 ETD Library.

4.1 Customizing a Java ETD

Currently SeeBeyond does not support the editing of pre-built Java ETDs. However, e*Gate offers a feature that allows you to convert existing Monk ETDs (**.ssc** files) to Java-enabled ETDs (**.xsc** files). This feature is the SSC Wizard.

To create a customized Java ETD

1. Create a corresponding Monk ETD, or use the Monk version of the Java ETD if available.
2. Customize the Monk ETD (**.ssc** file) using the e*Gate ETD Editor.
3. Convert the Monk ETD to a Java ETD using the e*Gate SSC Wizard.

   When the conversion is done, you have three files:

   - The original Monk ETD (**.ssc** file)
     Keep this file in case further customization is needed.
   - The Java version of the ETD (**.xsc** file)
   - The corresponding **.jar** file

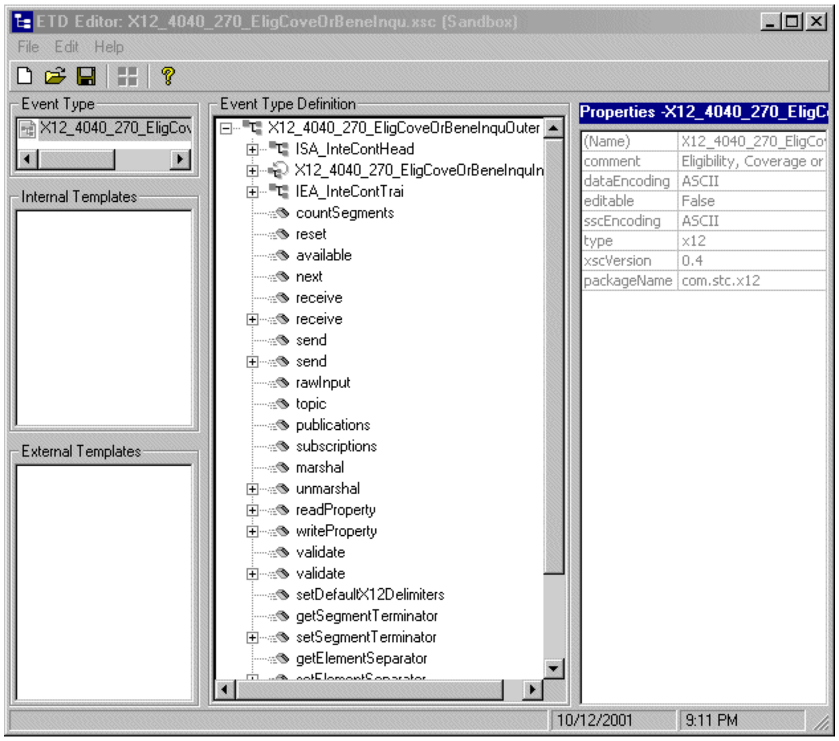
If you need to make further changes to the ETD, make the changes in the **.ssc** file and run the conversion again.

For specific instructions on using the e*Gate ETD Editor or the SSC Wizard, refer to the *e*Gate Integrator User's Guide*.

4.2 Viewing an X12 Java ETD in the ETD Editor

An example of a Java X12 270 transaction in the Java ETD Editor is shown in Figure 15.

Figure 15 Java X12 270 In the ETD Editor



```
X12_4040_270_EligCoveOrBeneInquOuter
  ISA_InteContHead
  X12_4040_270_EligCoveOrBeneInquIn
  IEA_InteContTrai
  countSegments
  reset
  available
  next
  receive
  receive
  send
  send
  rawInput
  topic
  publications
  subscriptions
  marshal
  unmarshal
  readProperty
  writeProperty
  validate
  validate
  setDefaultX12Delimiters
  getSegmentTerminator
  setSegmentTerminator
  getElementSeparator
```

(Name)	X12_4040_270_EligCo
comment	Eligibility, Coverage or
dataEncoding	ASCII
editable	False
sscEncoding	ASCII
type	x12
xscVersion	0.4
packageName	com.stc.x12

10/12/2001 9:11 PM

The ETD shown in Figure 15 is **x12_4040_270_EligCoveOrBeneInqu.xsc**. The root node is **x12_4040_270_EligCoveOrBeneInquOuter**. For each X12 ETD, the root node name is the same as the file name, but without the extension, and with **Outer** appended to the file name.

Some things to note about X12 Java ETDs:

- Bubble text labels are available for some of the items.
- In the **.xsc** file, the following naming conventions apply:
  - An element name begins with **E**
  - A segment loop name begins with **Loop**

4.3 Setting the Delimiters

The X12 ETDs must include some way for delimiters to be defined so that they can be mapped successfully from one ETD to another.

In X12, delimiters are specified in the interchange header segment (ISA).

The 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)

These delimiters can be set 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 Collaboration Editor by means of Java methods that are provided in the ETD files.

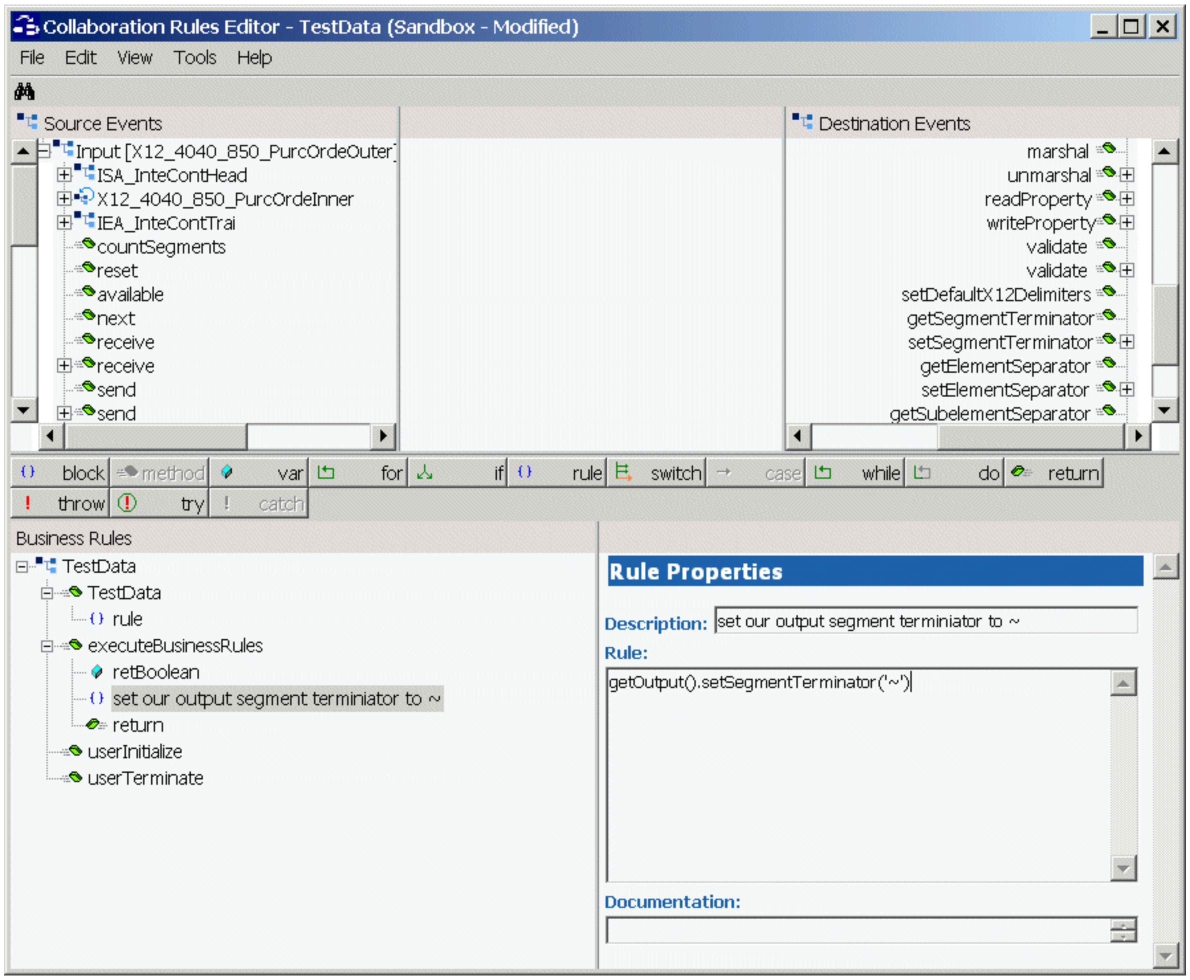
For specific information on the Java methods provided for the getting and setting of delimiters, refer to **"X12 ETD Library Java Methods" on page 55**.

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 ETD.

To set the delimiters for a Java X12 ETD in the Collaboration Rules Editor

1. Open the Collaboration in the Java Collaboration Rules Editor.
2. In the Business Rules section, add a rule.
3. Click on the method that you want to use.
4. Drag and drop to the Rule Properties section (an example is shown in Figure 16).

Figure 16 Setting Delimiters in a Java Collaboration Rules Component

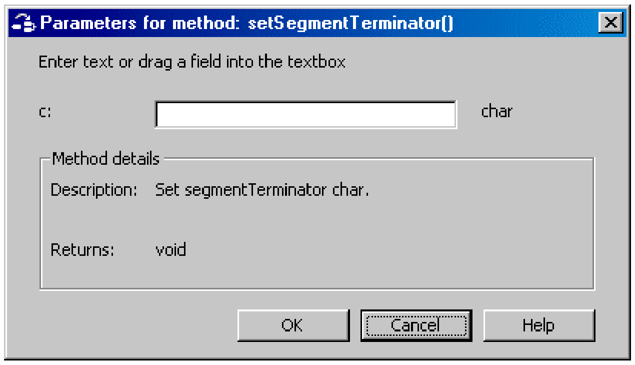


Description: set our output segment terminiator to ~

Rule:
```
getOutput().setSegmentTerminator('~')
```

5 The Parameters for Method dialog box appears (see Figure 17).

Figure 17 Parameters for Method Dialog Box



Parameters for method: setSegmentTerminator()

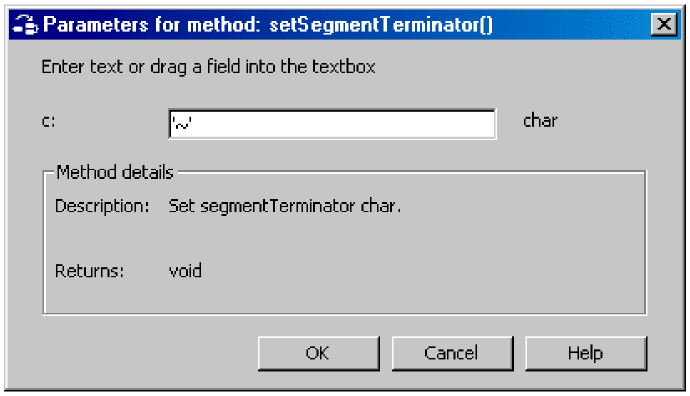
Enter text or drag a field into the textbox

c: char

Method details
Description: Set segmentTerminator char.
Returns: void

6 Set the delimiter value (an example is shown in Figure 18).

Figure 18 Parameters for Method Dialog Box Showing Delimiter Value



Parameters for method: setSegmentTerminator()

Enter text or drag a field into the textbox

c: '~' char

Method details
Description: Set segmentTerminator char.
Returns: void

7 Click **OK**.

8 Save the Collaboration Rules component.

Note: *You **must** specify the delimiters. You can do this either by setting individual delimiters to specific values, or by using the setdefaultX12delimiters Java method to set the defaults.*

4.4 Running Validation in the Java Collaboration Rules Component

An additional tool you can use for validating your data is to run one of the validation methods within the Collaboration.

The X12 ETD Library includes two Java methods provided for this purpose. They are as follows:

- validate
- validate(boolean)

For more information on these Java methods, refer to **"X12 ETD Library Java Methods" on page 55**.

4.4.1. Creating a Collaboration Rule to Validate a Java ETD

The elements that are part of an **.xsc** file can be dragged and dropped when two or more **.xsc** files are opened in the Collaboration Rules Editor (see the *e*Gate Integrator User's Guide* for more information). A field in the Source pane can be dragged to a field in the Destination Events pane. This action, when highlighted in the Business Rules pane, displays the rule in the Rule Properties pane.

The "validate" method nodes in an **.xsc** file can be used to validate an X12 message at run time. The methods return a string containing descriptions about any invalid data elements, segments, segment loops, envelopes, and so forth.

It also performs specific validations on the Interchange Group and Functional Group envelopes, and outputs any invalid information into the output string, as follows:

- Checks that the control numbers in the ISA and IEA segments match.
- It checks the number of transactions and verifies that against the transaction count value provided in the GE01 segment of the Functional Group trailer (GE).
- Validates the transaction count; checks the number of transactions and checks it against the count provided, says the transaction is invalid if they don't match.

The function returns a string. You can choose how to direct the output of the string; for example, to a log file.

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.*

4.5 Alternative Formats: ANSI and XML

All the Java X12 ETDs accept either standard ANSI X12 format or XML format as input, by default; no changes to the existing Collaborations are needed.

However, if you are using ETDs from version 4.5.1 or earlier and are using the Java ETDs, you must recompile all Collaborations that use these ETDs after installing the new version.

By default, output is ANSI. However, there are two Java Methods available for setting the output to XML:

- setXMLOutput (boolean isXML)

If the Collaboration is set to automatically publish (the default), set the argument to true to automatically publish XML output.

- marshal (boolean isXMLOutput)

  If the Collaboration is set to manual publication (via the **Manual Publish** check box in the Collaboration Rules component), set the argument to true to manually publish XML output.

4.5.1. XML Format for X12

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

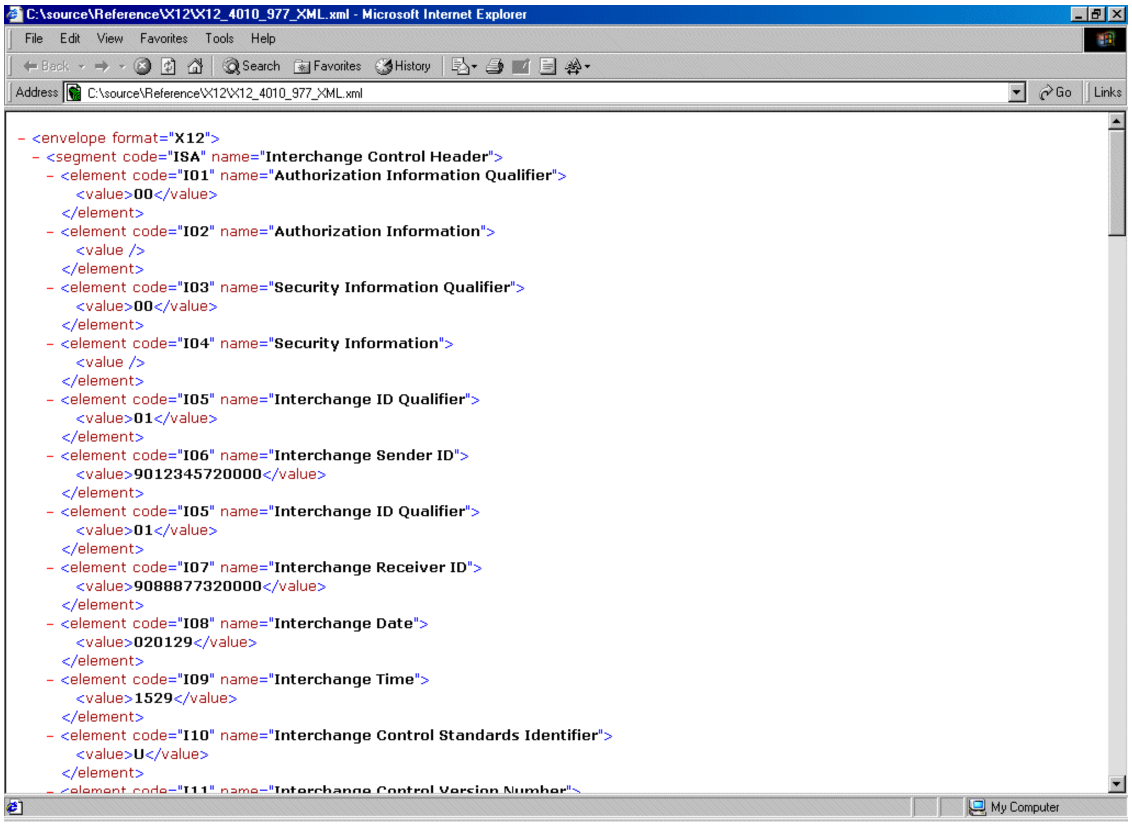
The XML X12 DTD is shown in Figure 19.

Figure 19 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 20 shows an X12 997 Functional Acknowledgment, in XML format.

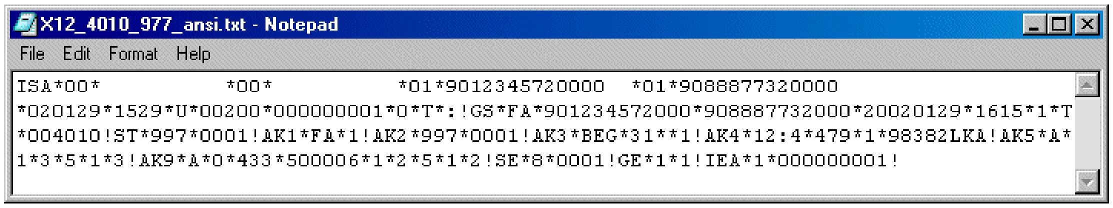
Figure 20 X12 997 Functional Acknowledgment—XML


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 21.

Figure 21 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!
```

4.5.2. Setting the Java Collaboration to XML Output

By default, output from a Collaboration that uses standard Events from the X12 ETD Library is in ANSI X12 format.

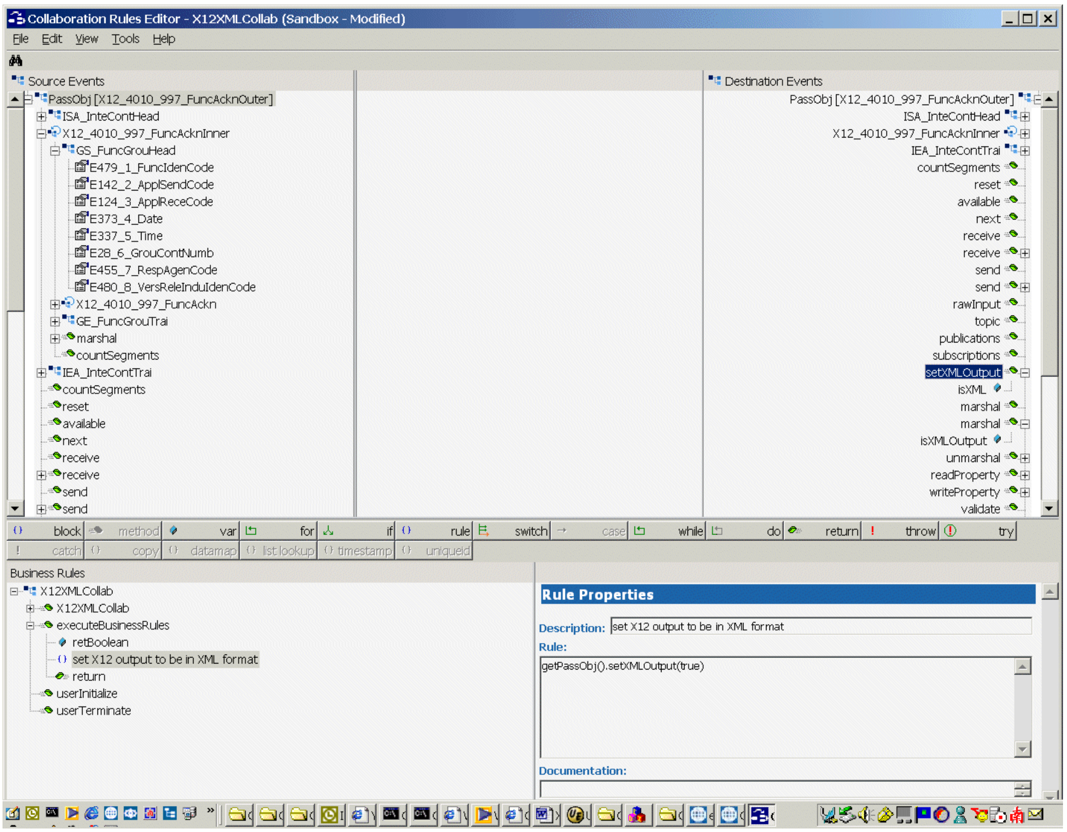
If you want to set the Collaboration to output XML format, use one of the following two Java methods:

- setXMLOutput (boolean isXML) with the argument set to true if the outbound X12 ETD is set to automatically publish.

- marshal (boolean isXMLOutput) with the argument set to true if the outbound X12 ETD is set to manually publish.

Figure 22 shows an X12 Collaboration. A rule is being added to the Collaboration to set the output to XML.

Figure 22 Setting the Output to XML in the Java X12 Collaboration



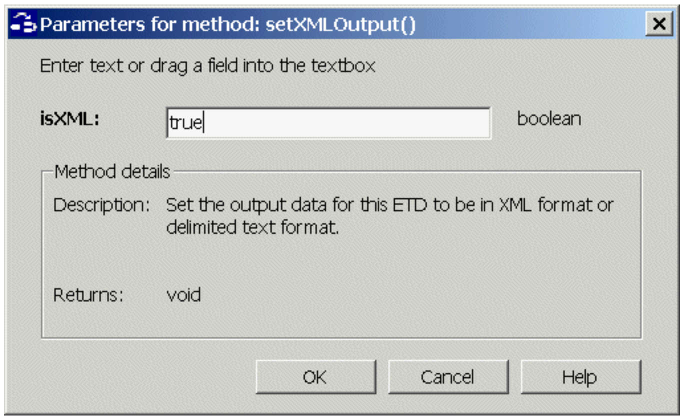
Description: set X12 output to be in XML format

Rule: getPassObj().setXMLOutput(true)

Documentation:

Figure 23 shows the parameter for setXMLOutput ( ) being set.

Figure 23 Specifying the Parameter for setXMLOutput ( )



Enter text or drag a field into the textbox

isXML: true boolean

Method details

Description: Set the output data for this ETD to be in XML format or delimited text format.

Returns: void

4.6 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, 010420 in the input file might be represented as 20010420 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.

Chapter 5

X12 ETD Library Java Methods

The X12 ETD Library contains Java methods that are used to extend the functionality of the ETDs. These methods allow you to get the X12 delimiters from the input ETD and set them appropriately for the output ETD; or to set the delimiters to the defaults.

The methods are:

- **setDefaultX12Delimiters** on page 55
- **getSegmentTerminator** on page 56
- **setSegmentTerminator** on page 57
- **getElementSeparator** on page 57
- **setElementSeparator** on page 58
- **getSubelementSeparator** on page 58
- **setSubelementSeparator** on page 59
- **getRepetitionSeparator** on page 60
- **setRepetitionSeparator** on page 60

The X12 ETD Library also includes the following custom Java methods for testing the validation Collaboration:

- **validate (no parameters)** on page 61
- **validate (boolean parameter)** on page 62

In addition, the library includes the following functions for setting the output of a Collaboration to XML:

- **setXMLOutput (boolean isXML)** on page 62
- **marshal (boolean isXMLOutput)** on page 63

setDefaultX12Delimiters

Description

Sets the default X12 delimiters.

Syntax

```
public void setDefaultX12Delimiters()
```

Parameters

None.

Constants

None.

Returns

void (none).

Throws

None.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
myETD.setDefaultX12Delimiters();
```

getSegmentTerminator

Description

Gets the segmentTerminator character.

Syntax

```
public char getSegmentTerminator()
```

Parameters

None.

Constants

None.

Returns

char
Returns the segment terminator character.

Throws

None.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
char segTerm=myETD.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.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
char c='~';
myETD.setSegmentTerminator(c);
```

getElementSeparator

Description

Gets the elementSeparator character.

Syntax

```
public char getElementSeparator()
```

Parameters

None.

Constants

None.

Returns

char
Returns the element separator character.

Throws

None.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
char elmSep=myETD.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.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
char c='+';
myETD.setElementSeparator(c);
```

getSubelementSeparator

Description

Gets the subelementSeparator character.

Syntax

```
public char getSubelementSeparator()
```

Parameters

None.

Constants

None.

Returns

char
Returns the getSubelement character.

Throws

None.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
char subeleSep=myETD.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.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
char c=':';
myETD.setSubelementSeparator(c);
```

getRepetitionSeparator

Description

Gets the RepetitionSeparator character.

Syntax

```
public char getRepetitionSeparator()
```

Parameters

None.

Constants

None.

Returns

char

Returns the getRepetitionSeparator character.

Throws

None.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
char repSep=myETD.getRepetitionSeparator();
```

setRepetitionSeparator

Description

Sets the RepetitionSeparator character.

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.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
char c='*';
myETD.setRepetitionSeparator(c);
```

validate (no parameters)

Description

Validates the ETD content in memory.

For example, if one of the nodes populated in the ETD has an inappropriate value, this method outputs a string such as the following.

```
X12_4010_850_PurcOrdeOuter.X12_4010_850_PurcOrdeInner[0].X12_4010_850
_PurcOrde[0].CUR_msk1_3_Curr.E280_3_ExchRate[0]: Value [0.2939] not
in the VALLIST list of [850/3///3] as specified in semantic rule [850/
3///3=[...:LOCALCODE:1:]]
```

If there are no problems with the ETD content, the output is a null string.

Syntax

```
public java.lang.String validate()
```

Parameters

None.

Constants

None.

Returns

String
A description of the errors in the data. If there are no errors, the string is null.

Throws

None.

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
string msg=myETD.validate();
```

validate (boolean parameter)

Description

Validates the ETD content, either immediately after unmarshaling or in memory.

When used with the parameter set to false, this method works in the same way as validate (with no parameters).

However, when the parameter is set to true, this method can be used to validate length information in the input data file.

For example, if the ETD expects a six-digit date and the input data provides an eight-digit date, this method outputs a string such as the following:

```
X12_3010_840_RequForQuotOuter.X12_3010_840_RequForQuotInner[0].GS_Fun
cGrouHead.E29_4_GrouDate: Its length of [8] is more than required max
of [6].
```

Syntax

```
public java.lang.String validate(boolean original)
```

Parameters

Name	Type	Description
original	boolean	If true, validates the ETD content right after unmarshaling. If false, validates the ETD in memory.

Constants

None.

Returns

String
A description of the errors in the data. If there are no errors, the string is null.

Throws

None

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
string msg=myETD.validate(true);
```

setXMLOutput (boolean isXML)

Description

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

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

Use this method when the X12 ETD 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

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
myETD.setXMLOutput(true);
```

marshal (boolean isXMLOutput)

Description

When used with the parameter set to true, this method generates the output byte array in XML format.

When used with the parameter set to false, this method generates the output byte array in ANSI format.

Use this method when the ETD is set to manual output. If the ETD is set to automatic output (the default), use setXMLOutput (boolean parameter) to achieve the same result.

Syntax

```
public byte[] marshal(boolean isXMLOutput)
```

Parameters

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

Constants

None.

Returns

byte []
The output in byte array format.

Throws

None

Examples

```
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter myETD=new com.stc.x12_401
0.X12_4010_850_PurcOrdeOuter();
......
......
byte[] output=myETD.marshal(true);
```

Appendix A

Implementation Tips

This appendix includes additional information that might be useful to you in an X12 implementation.

It provides information on how you can run a command-line Java ETD tester that is provided for the purpose. This allows you to test one or more input data files outside the e*Gate environment, to make sure the data is in the correct format to be successfully mapped by the ETD structures.

Note: *To simply test that an input data file is correctly mapped to an X12 ETD, you can also use the ETD Editor's "Run Test" utility. For more information, refer to the* ***e*Gate Integrator User's Guide****.*

A.1 Running the Command-Line Java Tester

The command-line utility allows you to test an input data file against the ETD. If the data is mapped correctly, the output file should very closely match the input data file, although there might be some insignificant differences.

Note: *The input file must include all envelope segments (ISA/IEA interchange control header and trailer, and GS/GE functional group header and trailer) as well as the ST/SE transaction set header and trailer segments.*

You must make adjustments to your classpath in order to run the utility.

For each X12 transaction, the ETD library includes two files, the **.xsc** file and the **.jar** file; for example for a Version 4010 850 (Purchase Order transaction) the ETD library includes **X12_4010_850_PurcOrde.xsc** and **X12_4010_850_PurcOrde.jar**.

The tester performs the following steps, in sequence:

1. Maps (unmarshals) the input data into the ETD structure.
2. Validates the data, and outputs a string description of any invalid data found.

   Specifically, it checks for correct data type, min/max values, optionality (whether a value is optional or required) and codelist (if there is a specific list of acceptable values, checks that the value is on the list). It also checks for matched control numbers and counts within the envelopes.

3 Outputs data (from the ETD mapped in the first step) into an output file that should mirror the content of the input file.

A.1.1. Setting the Classpath

The command-line utility requires certain files in the classpath. You can add them to the classpath environment variable before running the utility, or you can set them and run the utility in one action by using the "-classpath" option of the Java utility.

To change the classpath environment variable from a command line, type **set classpath=**.

The five .**jar** files required to be in the classpath for this utility to run are:

- The input file; for example, **X12_4010_PurcOrde.jar**
- **stcjcs.jar**. By default this is in **\<eGate>\client\classes**
- **xerces.jar** (used for XML parsing). By default this is in **\<eGate>\client\bin\java**
- **gnu-regexp-1.1.1.jar**. By default this is in **\<eGate>\client\bin\java**
- The shared segments/composites .**jar** file, **x12_nnnn_allsegscoms.jar**, for the X12 version you are working with. This is located in the templates folder for that version. For example, for version 4010, it would be:

  \<eGate>\server\registry\repository\default\etd\templates\x12\x12_4010\x12_4010_allsegscoms.jar

Note: *If you are running the utility on Windows, use semicolons to separate the strings and backslashes in the pathnames. If you are running on UNIX, use colons to separate the strings and forward slashes in the pathnames.*

A.1.2. Syntax

The basic syntax for the X12 command-line utility, with the classpath already set, is as follows:

```
java com.stc.x12_4010.[root node name from input file] [input file] [output file]
```

If you set the classpath and run the utility at the same time, it is as follows:

java -classpath [classpath settings of these files including full paths: input template .**jar** file, shared segments/composites .**jar** file for the version, **stcjcs.jar**, **xerces.jar**, **gnu-regexp-1.1.1.jar**] **com.stc.x12_4010.**<input ETD name, for example **X12_4010_850_PurcOrdeOuter**> <input file path and name> <output file path and name> [**A** or **X** (optional switch to indicate A for ANSI or X for XML)].

A.1.3. Example

An actual example for the version 4010 850 Purchase Order transaction, is shown below in two versions:

- With the classpath already set

- Setting the classpath and run the command both at the same time.

Running the utility with the classpath already set

if the classpath is already set and you are running the utility on a Windows platform, the command might be as follows:

```
java
com.stc.x12_4010.X12_4010_850_PurcOrdeOuter c:\X12_test\X12_4010_850.txt
output.dat A
```

Setting the classpath and running the utility

If you are setting the classpath and running the utility both at the same time, on a Windows platform, the command might be as follows:

```
java -classpath c:\egate\server\registry\repository\default\etd\templates\
x12\x12_4010\x12_4010_850_purcorde.jar;c:\egate\server\registry\repository\d
efault\etd\templates\x12\x12_4010\x12_4010_allsegscoms.jar;c:\egate\client\c
lasses\stcjcs.jar;c:\egate\client\bin\java\xerces.jar;c:\egate\client\bin\java\g
nu-regexp-1.1.1.jar com.stc.x12_4010.x12_4010_850_purcordeouter c:\_work\sef\
data\x12_4010_850.txt output.dat a
```

This example produces an output file named **output.dat** which you can open with a text editor to verify that it mirrors the content of the input file.

Index